Multigraph Modeling and Adaptive Large Neighborhood Search for the Vehicle Routing Problem with Time Windows

Hamza Ben Ticha, Nabil Absi, Dominique Feillet, Alain A. Quilliot

Multigraph Modeling and Adaptive Large Neighborhood Search for the Vehicle Routing Problem with Time Windows

Hamza Ben Ticha[a,*], Nabil Absi[a], Dominique Feillet[a], Alain Quilliot[b]

[a] *Ecole des Mines de Saint-Etienne and UMR CNRS 6158 LIMOS, CMP Georges Charpak F-13541 Gardanne, France*

[b] *LIMOS, Institut Supérieur d'Informatique de Modélisation et leurs Applications, ISIMA, Campus des Cèzeaux, Aubière Cedex, France*

Abstract

In this paper we propose a multigraph model and a heuristic for the Vehicle Routing Problem with Time Windows (VRPTW). In the classical VRPTW, travel information is commonly represented with a customer-based graph, where an arc is an abstraction of the best road-network path between two nodes. We consider the case when parallel arcs are added to this graph to introduce different compromises between travel time and cost. It has been shown in the literature that this multigraph modeling enables substantial gains in the solution quality, while highly complicating the problem. We develop an Adaptive Large Neighbourhood Search (ALNS) heuristic in which a special data structure and dynamic programming algorithms are used to efficiently address the multigraph setting. Computational experiments on several set of instances demonstrate the effectiveness of our solution method and the impact of alternative paths on the solution quality.

Keywords:
Vehicle Routing Problem with Time Windows, Multigraph, Large Neighborhood Search, Dynamic programming, Road-network information.

1. Introduction

Distribution is one of the most essential components in logistic systems. It is estimated that almost half of the logistic costs are due to distribution and, for some industries, this accounts for up to 70% of total costs [1]. The Vehicle Routing Problem (VRP), introduced by Dantzig and Ramser about sixty years ago [2], attempts to optimize distribution costs. The study of the VRP has been highly influential, as attested by the impressive number of publications on this topic [3].

Basically, vehicle routing problems compute a minimum-cost set of vehicle routes that start and end at a depot. Each customer has to be supplied exactly once and each route has to satisfy constraints such as vehicle capacity, customer time windows or route duration. In most studies, geographic information is expressed with a so-called customer-based graph, where nodes represent points of interest (customers, depot) and arcs symbolize a path between these nodes in the road-network. In many cases however, this model does not capture all the relevant roadways. Assume for example that a problem involves both travel times and travel distances. Given two customers, the min-time path in the road-network between these customers is not necessarily the same as the

*Corresponding author
Email address: hamza.ben-ticha@emse.fr (Hamza Ben Ticha)

min-distance path. Yet, only one of these two paths will be kept and represented by an arc in the customer-based graph. The other one will be lost. Also, many other efficient paths, with different trade-offs between time and distance, will also be forgotten. Clearly, this implies a loss of flexibility in route optimization and, possibly, an increase in travel costs. For example, one might sometimes prefer a fast but more costly connection, when delivery times are restricted, or the opposite during slack times.

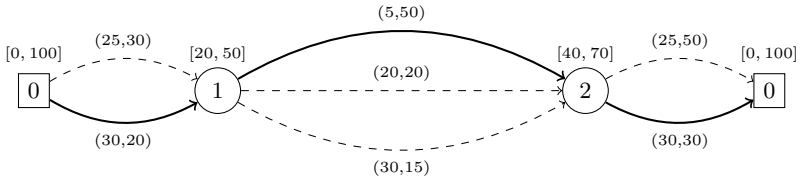
A few papers (Garaix et al. [4], Lai et al. [5], Ben Ticha et al. [6]) have analyzed the negative effect of the customer-based graph when arcs have several attributes (as time, distance and so on). Considering different transportation schemes in different geographical contexts, they all show significant increases in solution costs, compared to models that embed the complete road-network information. Ben Ticha et al. [7] have gone one step further and have reviewed the literature devoted to what they call *vehicle routing problems with road-network information*, *i.e.*, vehicle routing problems in which travel information is defined at the level of road segments. They exhibit several other limits of customer-based graphs. For example, these graphs are not suitable for complex criteria as carbon emissions, when speed is a decision variable: as the speed can be modified at any place in the road-network, paths cannot be precomputed [8]. Two alternatives to the customer-based graph have been proposed in the literature [7]. The first considers the customer-based graph and adds an arc for every efficient path that exists between two nodes. The graph is then referred as a multigraph. The second possibility is simply to ignore the customer-based graph and to rely on a graph that mimics the road-network. The first idea of using a multigraph is relatively recent. Exact solution schemes were investigated in Garaix et al. [4] and Ben Ticha et al. [6], for example. Some rare early papers have considered a road-network graph (*e.g.*, Orloff [9], Fleischmann [10], Cornuéjols et al. [11]). More recently, Letchford et al. [12] and Ben Ticha et al. [13] have investigated exact solution methods with branch-and-price algorithms. For both types of graph, the literature on heuristic methods is extremely limited.

In this paper, we focus on the modeling with a multigraph. Our objective is to develop a heuristic capable of obtaining adequate-quality solutions quickly. Subsequently, we deal with a standard routing problem that involves two attributes on arcs: the Vehicle Routing Problem with Time Windows (VRPTW). The VRPTW finds a minimum-cost set of vehicle routes that satisfy customer requests within their time windows. Information on travel time and on travel cost is associated with each arc. Travel time information is necessary to make sure that customers are served within their time windows. Travel cost determines the quality of solutions. Since we consider that this data is available on road segments, we call our problem the VRPTW with road-network information (VRPTW$_{RN}$).

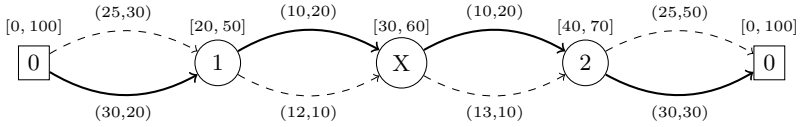
The remainder of this paper is organized as follows. In Section 2 we review the related literature. In Section 3, the VRPTW$_{RN}$ is formally introduced. The ALNS algorithm is described in Section 4. Computational experiments and analyses are detailed in Section 5.

2. Literature review

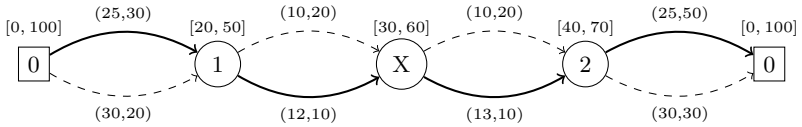
As far as we know, the first papers interested in vehicle routing problems with travel information supported by a multigraph are of Baldacci et al. [14] and Garaix et al. [4]. Both works were mainly interested in exact solution algorithms (namely, branch-and-price), but Garaix et al. [4] also investigated heuristic approaches. In particular, Garaix et al. [4] showed an important effect of the multigraph model. Even very simple operations as customer removals and insertions become difficult to evaluate. Indeed, these moves can affect the arc to select between consecutive customers, anywhere in the vehicle route. Garaix et al. [4] proved that for a given sequence of nodes, computing an optimal sequence of arcs is NP-hard. They called this problem the Fixed Sequence Arc Selection



(a) Initial route



(b) Same route after a customer is inserted, without changing the arcs



(c) Same route after the reoptimization of arc selection

Figure 1: Illustration of the insertion of a customer in a route defined on a multigraph

Problem (FSASP). To illustrate the difficulty of arc selection, let us consider the example presented in Figure 1. Figure 1(a) shows a vehicle route defined by node sequence $(0, 1, 2, 0)$. Time windows are shown above nodes. Parallel arcs between pair of nodes are provided, with their cost and travel time in parentheses, given in this order. Arcs that allow minimizing the cost of the route are represented with a thick line. Assume that we want to evaluate the insertion of customer X between customers 1 and 2. This can be done by selecting the less costly arcs that allow linking 1 with X and X with 2 without violating any time window. In this case, the obtained route is provided on Figure 1(b) and has a total cost equals to 80. However, a different sequence of arcs, shown on Figure 1(c), enables decreasing the route cost down to 75. In view of the difficulty of arc selection, Garaix et al. [4] solved their problem with a simple descent algorithm. Essentially, their method is composed of an initialization phase, based on greedy insertion, and an improvement phase, based on customer relocation. In both phases, the main mechanism is the customer insertion that requires solving an FSASP at each iteration. Garaix et al. [4] proposed solving these FSASP by dynamic programming. Experiments showed that this procedure is very time-consuming.

Lai et al. [5] faced the same difficulty and proposed to circumvent the complexity of the FSASP by computing arc sequences heuristically. Instead of solving the FSASP by dynamic programming, they applied a fast greedy method inspired from knapsack heuristics. Experiments were limited to multigraphs with two arcs in parallel. Wang and Lee [15] and Setak et al. [16] also developed heuristic methods for vehicle routing problems with a multigraph structure. However, in their case, at each time instant, an arc dominates other parallel arcs, which breaks the complexity of the FSASP. In addition to these works, one could cite another heuristic method developed by Caramia and Guerriero [17]. However, their context is long-haul freight distribution and the structure of the problem is very far from a vehicle routing problem. The multigraph was introduced to model the presence of multiple transportation modes and logistics operators. The authors proposed a heuristic composed of two phases: first, a set of efficient candidate paths is computed in the network; then, demands are assigned to transportation means.

The literature on the VRPTW is much more abundant. This problem has drawn the attention of many researchers and a large number of solution methods have been proposed in the literature

(Desaulniers et al. [18]). Baldacci et al. [19] reviewed the literature related to exact solution algorithms. Kallehauge [20] focused on mathematical formulations and polyhedral analyses. Construction heuristics and local-improvement methods were reviewed in [21] and metaheuristics were discussed in [22].

In this work, we develop a heuristic following the framework of Adaptive Large Neighborhood Search (ALNS). ALNS was introduced by Ropke and Pisinger [23] to solve the Pickup and Delivery Problem with Time Windows. It was itself adapted from the Large Neighborhood Search heuristic (LNS) proposed by Shaw [24] to solve the VRPTW. ALNS has shown its efficiency for a large number of vehicle routing problems [3]. The method is based on a destroy and repair mechanism: subsets of customers are repeatedly removed from a solution and reinserted to form a new solution. The difficulty in our context is to manage removals and reinsertions efficiently.

3. Problem description and multigraph representation

We define the VRPTW$_{RN}$ using a directed graph $G_{RN} = (V_{RN}, A_{RN})$. V_{RN} contains the depot node 0 and nodes that represent road junctions. Among these nodes, a subset of size n represents customers. Arcs $(i, j) \in A_{RN}$ model road segments and are defined with a travel cost and a travel time. We associate with each customer i a demand d_i, a time window $[e_i, l_i]$ and a service time t_i. The depot also receives a time window $[e_0, l_0]$ that indicates the earliest starting and latest ending time of a vehicle tour. We consider a homogeneous fleet with K vehicles of loading capacity Q. The objective of the VRPTW$_{RN}$ is to compute a set of paths in G_{RN}, that start from the depot, return to the depot, satisfy time windows and vehicle capacity, so as to serve all the customers exactly once with a minimal total travel cost.

In order to tackle the VRPTW$_{RN}$, we introduce a directed multigraph $G = (V, A)$. $V = \{0, 1, \ldots, n\}$ is composed of node 0 for the depot and nodes 1 to n for the customers. A is defined as follows. For each pair $(i, j) \in V \times V$, we introduce a set $A_{(i,j)} = \{(i, j)^p, p = 1, \ldots, m_{ij}\}$ of parallel arcs, where m_{ij} is the number of efficient paths in G_{RN} between i and j. A path is efficient if it is not dominated with regards to travel time and cost; it is only considered if it is compatible with the time windows, *i.e.*, it allows reaching j on time (before l_j) when leaving i at time e_i. Given an arc $(i, j)^p$, we denote its travel cost by $c_{(i,j)^p}$ and its travel time by $t_{(i,j)^p}$. The VRPTW$_{RN}$ then equivalently consists in finding a set of paths in G, that start from the depot, return to the depot, satisfy customer time windows and vehicle capacity, and serve all the customers exactly once with a minimal total travel cost. In this paper, we assume that graph G and associated travel time and travel cost information are given as inputs. An efficient method to compute this data was proposed by Ben Ticha et al. [25].

4. Solution method

Our solution method follows the framework of ALNS. This framework is described by Algorithm 1. The algorithm is initialized with a solution s_{init} constructed using an adaptation of the Clarke and Wright algorithm [26], see Section 4.2. This solution is temporarily considered as the current solution (s_{curr}) and as the best solution (s_{best}). Then, at each iteration, a destroy and a repair operators are selected. These operators are chosen in sets D and R described in Sections 4.3 and 4.4, respectively, with a policy presented in Section 4.5. The destroy and the repair operators are successively applied to s_{curr} (Line 6), to produce a solution s. A simulated annealing mechanism, detailed in Section 4.6, decides whether the new solution becomes the current solution or not. Also, the best known solution is updated if needed (Line 9, $cost(.)$ is the cost of a solution). The

algorithm stops after a given number of iterations. Because finding a feasible VRPTW solution is NP-complete, we allow infeasible solutions in the algorithm. In the constructive algorithm that initializes the method, we do not consider the limit on the fleet size (see Section 4.2). Then, we try to recover a feasible solution, if needed, by limiting the degree of infeasibility during insertions (see Section 4.4). The main innovation in the algorithm stands in the management of arc selections in removal and insertion operations. We detail how we proceed in the next subsection.

Algorithm 1 Adaptive Large Neighborhood Search

1: compute an initial solution s_{init}
2: $s_{curr} \leftarrow s_{init}$
3: $s_{best} \leftarrow s_{curr}$
4: **while** the stopping criterion is not met **do**
5: select a destroy operator $d \in D$ and a repair operator $r \in R$
6: $s \leftarrow r(d(s_{curr}))$
7: **if** $accept(s, s_{curr})$ **then**
8: $s_{curr} \leftarrow s$
9: **if** $cost(s_{curr}) < cost(s_{best})$ **then**
10: $s_{best} \leftarrow s_{curr}$
11: **end if**
12: **end if**
13: **end while**
14: **return** s_{best}

4.1. Arc selection procedure

When applying destroy and repair operators, one has to repeatedly evaluate the feasibility and the cost of new sequences of nodes. As already explained, an exact evaluation necessitates to reoptimize the arc selection, *i.e.*, to solve an NP-hard problem: the FSASP. Garaix et al. [4] express the solution of the FSASP as a Shortest Path Problem with Resource Constraints. They apply a standard labeling dynamic programming procedure (see, *e.g.*, Irnich and Desaulniers [27]) that works as follows. Let us consider a sequence $\pi = (0, i_1, \ldots, i_{n_\pi}, 0)$. A label is defined with two attributes: cost and time. An initial label $(0, 0)$ is assigned to the first copy of the depot. This label is then extended to the next node in the sequence, using all parallel arcs $(0, i_1)^p$ $(p = 1, \ldots, m_{0i_1})$. It results in m_{0i1} labels associated with node i_1. These labels are then all extended to the next customer i_2, through all parallel arcs $(i_1, i_2)^p$, and the process is repeated until the end of the sequence is reached. When extending a label, the arc travel cost and time are added to the corresponding attribute of the label. Time windows are checked to eliminate infeasible labels and waiting times are added when necessary. Dominance rules are applied to eliminate dominated labels (see Algorithm 2 and next paragraphs for details). In their computational experiments, the authors show the limits of this method. Computing times are not compatible with a metaheuristic that requires the evaluation of a large number of sequences, as ALNS. For this reason it is critical for us to manage arc selection more efficiently. We follow the ideas initiated in Savelsbergh [28] and develop an improved procedure based on bidirectional search and incremental data.

Label preprocessing

This preprocessing is applied on the vehicle routes of the starting solution s_{init} of Algorithm 1 (Line 1). Each route can be represented by a sequence $\pi = (0, i_1, \ldots, i_{n_\pi}, 0)$. A dynamic programming algorithm similar to that of Garaix et al. [4] is applied to each sequence. An equivalent

algorithm, starting from the last node of the sequence and traversing arcs in backward is also applied. The label sets generated with these two algorithms are kept. The forward and backward labeling algorithms are fully described in Algorithms 2 and 3. The copy of the depot (ending the sequence) is renamed $n+1$ in these algorithms and for the subsequent subsections.

Algorithm 2 Forward labeling algorithm

$i \leftarrow 0$
$FL[i] \leftarrow (0, 0)$
while $i \neq n+1$ **do**
  $j \leftarrow next(\pi, i)$
  for all labels $(c, t) \in FL[i]$ **do**
    for all arcs $(i, j)^p \in A_{(i,j)}$ **do**
      if $t + t_i + t_{(i,j)^p} \leq l_j$ **then**
        $t' \leftarrow \max\{t + t_i + t_{(i,j)^p}, e_j\}$
        insert with dominance label $(c + c_{(i,j)^p}, t')$ in $FL[j]$
      end if
    end for
  end for
  $i \leftarrow next(\pi, i)$
end while
return FL

Algorithm 3 Backward labeling algorithm

$j \leftarrow n+1$
$BL[j] \leftarrow (0, l_{n+1})$
while $j \neq 0$ **do**
  $i \leftarrow previous(\pi, j)$
  for all labels $(c, t) \in BL[j]$ **do**
    for all arcs $(i, j)^p \in A_{(i,j)}$ **do**
      if $t - t_{(i,j)^p} - t_i \geq e_i$ **then**
        $t' \leftarrow \min\{t - t_{(i,j)^p} - t_i, l_i\}$
        insert with dominance label $(c + c_{(i,j)^p}, t')$ in $BL[i]$
      end if
    end for
  end for
  $j \leftarrow previous(\pi, j)$
end while
return BL

The outputs of these algorithms are lists of labels $FL[i]$ and $BL[i]$ associated with each node i in the sequence. Function $next(\pi, i)$ (resp., $previous(\pi, i)$) returns the node that follows (resp., precedes) node i in sequence π. The *insertion with dominance* of a label in a list of labels, is performed by comparing the new label with every label in the list. If the new label is dominated, the list is not modified. Otherwise, the label is added to the list and all dominated labels are removed. In order to optimize certain operations, lists $FL[i]$ and $BL[i]$ are implemented in non-decreasing order of the travel cost. When vehicle route are modified, this information is updated, so that it is always available.

Evaluation of the removal of a node from a sequence

To evaluate the removal of a node u between two nodes $a = previous(\pi, u)$ and $b = next(\pi, u)$ in a sequence π, we apply the following algorithm:

1. Extend every label in $FL[a]$ to b, using every arc in $A_{(a,b)}$ and following the extension scheme detailed in Algorithm 2. We call L_F the resulting label list.
2. Consider every pair $((c_F, t_F), (c_B, t_B))$ of labels in $L_F \times BL[b]$. A pair is feasible if $t_F \leq t_B$. Compute the cost $c_F + c_B$ of every feasible pair and return the minimal value.

The value returned by the algorithm is the best possible cost of sequence π with node u removed.

Update of incremental data after a removal

When a node u is removed from a sequence π, we need to update incremental data. First, we empty all sets $FL[i]$ for nodes i positioned after u in π, and sets $BL[i]$ for nodes i positioned before u. Then, we apply Algorithms 2 and 3 with a different initialization (Lines 1 and 2): i is initialized to $previous(u, \pi)$ in Algorithm 2, j is set to $next(u, \pi)$ in Algorithm 3. After this initialization, u is removed from the sequence and the main loop is executed normally for each algorithm.

Evaluation of the insertion of a node at a given position in a sequence

To evaluate the insertion of a node u between two nodes a and $b = next(\pi, a)$ in a sequence π, we apply the following algorithm:

1. Extend every label in $FL[a]$ to u, using every arc in $A_{(a,u)}$. We call L_F the resulting label list.
2. Extend every label in $BL[b]$ backwardly to u, using every arc in $A_{(u,b)}$. We call L_B the resulting label list.
3. Consider every pair $((c_F, t_F), (c_B, t_B))$ of labels in $L_F \times L_B$. A pair is feasible if $t_F \leq t_B$. Compute the cost $c_F + c_B$ of every feasible pair and return the minimal value.

The value returned by the algorithm is the best possible cost of sequence π with node u inserted after a. Note that the feasibility of the insertion with regards to vehicle capacity is evaluated upstream.

Update of incremental data after an insertion

When a node u is inserted between two nodes a and $b = next(\pi, a)$ in a sequence π, we need to update incremental data. First, we empty all sets $FL[i]$ for nodes i positioned after a in π, and sets $BL[i]$ for nodes i positioned before b. We also empty sets $FL[u]$ and $BL[u]$. Then, we apply Algorithms 2 and 3 with a different initialization (Lines 1 and 2): i is initialized to a in Algorithm 2, j is set to b in Algorithm 3. After this initialization, u is inserted in the sequence and the main loop is executed normally for each algorithm.

4.2. Initial solution

To provide an initial solution to our heuristic, we adapt the Clarke and Wright *savings* algorithm [26]. This algorithm was developed in the context of the VRP and works as follows. Consider a solution of the VRP and two routes π_1 and π_2 whose last and first customers are i and j, respectively. If the vehicle capacity allows it, a single route can be obtained by merging π_1 and π_2: after having reached i at the end of π_1, the vehicle goes to j and continues π_2. The impact on cost, the so-called saving sav_{ij}, can be precomputed and is given by:

$$sav_{ij} = c_{(i,0)} + c_{(0,j)} - c_{(i,j)} \tag{1}$$

The principle of the Clarke and Wright algorithm is to compute sav_{ij} for all pairs of customers (i, j), to sort them in a non-increasing order and to progressively merge routes when possible, according to this order. A pair (i, j) is eligible for a merging, and a saving sav_{ij} can be obtained, if three conditions holds: i is the last customer of a route, j is the first of a second route, the cumulated load of these two routes does not exceed the vehicle capacity. The algorithm is initialized with a solution composed of back-and-forth trips between the depot and a customer. We adapt this algorithm to take care of the time windows and of the parallel arcs between nodes. We implement the following modifications:

- At the initialization, every customer i is reached with the min-cost arc from the depot (*i.e.*, in $A_{(0,i)}$). Then, the min-cost arc in $A_{(i,0)}$ that enables returning to the depot on time is used.
- When computing the list of savings, much more combinations are introduced. A saving s_{ij}^{xyz} is evaluated for all pairs of customers (i, j), and all arcs $(i, 0)^x \in A_{(i,0)}$, $(0, j)^y \in A_{(0,j)}$ and $(i, j)^z \in A_{(i,j)}$.
- When evaluating the feasibility of merging two routes for a combination (i, j, x, y, z), some conditions are added:
  - arcs $(i, 0)^x$ and $(0, j)^y$ have to be selected in the current solution,
  - merging the routes that contain i and j with arc $(i, j)^z$ has to be compatible with time windows.

  The latter condition is checked with incremental data equivalent to that described in Section 4.1. Note that using this data also allows to reoptimize arc selection.

With the mechanism of savings, the Clarke and Wright algorithm tends to minimize the number of vehicle routes, but there is no guarantee that the provided solution respects the fleet size. In this case, attempts to recover feasibility will be carried out in the main loop of the ALNS algorithm (see Section 4.4).

4.3. Removal heuristics

We propose three removal heuristics, which differ in the way customers are selected. A removal heuristic takes as inputs a feasible solution s and a number ν of customers to be removed, and returns a set Π of feasible routes and a set $\mathcal{O}$ of ν removed customers.

Adapted Shaw removal heuristic

This heuristic was first proposed by Shaw [24] for the VRPTW and next adapted by Ropke and Pisinger [23] for the Pickup and Delivery Problem with Time Windows. The principle is to remove *similar* costumers. The rationale is to favor diversification when reinserting customers. Indeed, due to the tight structure of VRPTW solutions, removing very different customers might give no other choice than reinserting each customer at its original position. Given solution s, we evaluate the similarity $R_{ij}(s)$ between two customers i and j with the following measure (adapted from Shaw [24]):

$$R_{ij}(s) = \alpha_1 \min_{1 \leq p \leq m_{ij}} c_{(i,j)^p} + \alpha_2 |t_i(s) - t_j(s)| + \alpha_3 |d_i - d_j| + \alpha_4 \left(1 - \frac{|RC_i(s) \cap RC_j(s)|}{\min\{|RC_i(s)|, |RC_j(s)|\}}\right) + X_{ij}(s) \quad (2)$$

In this formula, $t_i(s)$ and $t_j(s)$ are the starting times of the service for customers i and j in solution s; $RC_u(s)$ is the set of positions where u can be inserted in s $(u = i, j)$; $X_{ij}(s) = 1$ if i and j are

served by the same vehicle in s, 0 otherwise. Parameters α_1 to α_4 are weights chosen in $[0, 1]$. At a given iteration of the ALNS algorithm, $R_{ij}(s)$ can be computed in constant time except for the term weighted by α_4. This term is particularly time consuming as it requires evaluating all the possible insertion positions, in all the routes of the current solution, for i and j, with the algorithm presented in Section 4.1. In Section 5.4, we conduct a sensitivity analysis that justifies using this term.

The adapted Shaw removal heuristic is detailed in Algorithm 4. It first randomly selects a customer and stores it in set $\mathcal{O}$. Then, for $\nu - 1$ iterations, it randomly selects a customer u in $\mathcal{O}$, finds a similar customer i and add i to $\mathcal{O}$. Customer i is selected among the customers still in the solution (*i.e.*, in set $\mathcal{I}$), according to measure $R_{ui}(s)$ and with some randomness controlled by a parameter γ_1: the higher γ_1, the more similar the customer. Once set $\mathcal{O}$ is computed, the routes of the solution are all stored in a set Π and the customers are successively removed. At this step, the removal procedure of Section 4.1 is used.

Algorithm 4 Adapted Shaw removal heuristic

1: $\mathcal{I} \leftarrow \{1, \ldots, n\}$, $\mathcal{O} \leftarrow \emptyset$
2: $i \leftarrow$ random customer in $\mathcal{I}$
3: $\mathcal{I} \leftarrow \mathcal{I} \setminus \{i\}$, $\mathcal{O} \leftarrow \mathcal{O} \cup \{i\}$
4: **while** $|\mathcal{O}| < \nu$ **do**
5: $\quad u \leftarrow$ random customer in $\mathcal{O}$
6: $\quad y \leftarrow$ random number in $[0, 1[$
7: $\quad r \leftarrow \lfloor y^{\gamma_1} \times |\mathcal{I}| \rfloor$
8: $\quad i \leftarrow \mathrm{r}^{th}$ most similar customer to u in $\mathcal{I}$ according to measure $R_{ui}(s)$
9: $\quad \mathcal{I} \leftarrow \mathcal{I} \setminus \{i\}$, $\mathcal{O} \leftarrow \mathcal{O} \cup \{i\}$
10: **end while**
11: $\Pi \leftarrow \{\pi : \pi \in s\}$
12: remove customers in $\mathcal{O}$ from their routes in Π
13: **return** $\mathcal{O}$ and Π

Random removal heuristic

As in [23], this removal heuristic simply selects ν customers randomly and insert them in set $\mathcal{O}$. The algorithm then constructs Π and returns $\mathcal{O}$ and Π as in Algorithm 4 (Lines 11–13).

Worst removal heuristic

This heuristic was introduced by Ropke and Pisinger [23]. Its principle is to remove the most costly customers from the solution. The heuristic is driven by a measure $\Delta_i^-(\Pi)$ that gives the difference between the cost of a set of routes Π and the cost of the same set with customer i removed. The evaluation of the removal is carried out with the procedure described in Section 4.1. In particular, arc selection on the modified route is reoptimized. Equivalently to the adapted Shaw removal heuristic, a random component is added, controlled with a parameter γ_2. The heuristic is detailed in Algorithm 5. Note that contrary to the adapted Shaw heuristic, customers are progressively removed and the measure that drives customer selection is updated accordingly.

4.4. Insertion heuristics

We propose four insertion heuristics. These heuristics take as inputs a set Π of feasible routes and a set $\mathcal{O}$ of customers not present in Π. Their output is a solution s (with a number of routes

Algorithm 5 Worst removal heuristic

$\mathcal{I} \leftarrow \{1, \ldots, n\}$, $\mathcal{O} \leftarrow \emptyset$
$\Pi \leftarrow \{\pi : \pi \in s\}$
compute $\Delta_i^-(\Pi)$ for all customers $i \in \mathcal{I}$
while $|\mathcal{O}| < \nu$ **do**
  $y \leftarrow$ random number in $[0, 1[$
  $r \leftarrow \lfloor y^{\gamma_2} \times |\mathcal{I}| \rfloor$
  $i \leftarrow$ rth most costly customer in $\mathcal{I}$ according to measure $\Delta_i^-(\Pi)$
  $\mathcal{I} \leftarrow \mathcal{I} \setminus \{i\}$, $\mathcal{O} \leftarrow \mathcal{O} \cup \{i\}$
  remove i from Π and call π^* the modified route
  compute $\Delta_j^-(\Pi)$ for all customers $j \in \pi^*$
end while
return $\mathcal{O}$ and Π

potentially larger than the fleet size). Each heuristic iteratively inserts a customer from $\mathcal{O}$ in Π, until $\mathcal{O}$ is empty and Π thus becomes a feasible solution. The heuristics differ in the order in which the customers are selected in $\mathcal{O}$ and in the way they are inserted in Π.

If insertions result in a set Π with $|\Pi| > \max\{K, |s_{curr}|\}$, the insertion procedure is stopped and a next iteration of the ALNS is started from the same current solution s_{curr}. This condition allows to manage infeasible solutions for a number of iterations. However, the degree of infeasibility (number of routes in excess with regards to K) is not allowed to increase. As soon s_{curr} becomes feasible, infeasibility is not allowed anymore.

Greedy insertion heuristic

This heuristic follows a best insertion policy. We compute for each customer $i \in \mathcal{O}$ and for each route $\pi \in \Pi$, the insertion cost $\Delta_i^+(\pi)$. This cost is computed with the algorithm presented in Section 4.1, applied for all insertion positions. $\Delta_i^+(\pi)$ is set to the cost of the best insertion. Once these values are obtained, we compute best insertion costs in Π: $\Delta_i^+(\Pi) = \min_{\pi \in \Pi} \Delta_i^+(\pi)$.

A customer i that minimizes $\Delta_i^+(\Pi)$ is inserted in Π. The insertion costs are then updated and the procedure is repeated until all the customers have been inserted. The insertion is carried out with the procedure detailed in Section 4.1. When updating insertion costs, we only recompute values $\Delta_i^+(\pi)$ for the modified route.

Regret insertion heuristic

This heuristic is similar to the greedy insertion heuristic, except that it introduces a look-ahead strategy. Given $i \in \mathcal{O}$, if we denote by π^* the route in Π that allows to reach a minimum insertion cost for i (*i.e.*, $\Delta_i^+(\pi^*) = \Delta_i^+(\Pi)$), a regret $R_i^+(\Pi)$ is defined as follows:

$$R_i^+(\Pi) = \min_{\pi \in \Pi \setminus \{\pi^*\}} \Delta_i^+(\pi) - \Delta_i^+(\Pi) \tag{3}$$

Contrary to the best insertion heuristic, the customer inserted in Π at a given iteration is the one with a maximum regret $R_i^+(\Pi)$. All other steps of the method are kept the same.

Non-myopic insertion heuristic

This heuristic also extends the greedy insertion heuristic. Given $i \in \mathcal{O}$, if we denote by Π' the set of routes that would be obtained after the best insertion of i in Π, an *impact value* $I_i^+(\Pi)$ is

defined as follows:

$$I_i^+(\Pi) = \Delta_i^+(\Pi) + \sum_{j \in \mathcal{O} \setminus \{i\}} (\Delta_j^+(\Pi') - \Delta_j^+(\Pi)) \tag{4}$$

The rationale behind this measure is to take account of the impact that the insertion of i can have on future insertions. The difference $\Delta_j^+(\Pi') - \Delta_j^+(\Pi)$ evaluates this impact for the remaining customers $j \in \mathcal{O} \setminus \{i\}$. The customer inserted in Π at a given iteration is the one with a minimum impact $I_i^+(\Pi)$. All other steps of the method are the same as in the greedy insertion heuristic. Note that to compute $\Delta_j^+(\Pi')$, we only need to compute $\Delta_j^+(\pi')$ for the route π' that would result from the best insertion of i. However, this heuristic might appear particularly time consuming as it involves many calls to the evaluation of customer insertions. In Section 5.4, we conduct a sensitivity analysis showing that it however contributes positively to the solution quality.

Simple insertion heuristic

The aim of this heuristic is to help diversifying the search. At each iteration, the customer i taken from $\mathcal{O}$ is randomly selected. If the number of routes in Π is lower than the fleet size K, a new route $(0, i, 0)$ is added to Π. Otherwise, a route π is randomly selected in Π and the insertion of i in π is tried. For that matter, a best insertion policy is applied and the procedure described in Section 4.1 is used. If the insertion fails, another route is selected, and so on until the insertion is done.

4.5. Adaptive strategy for the selection of removal and insertion heuristics

In Sections 4.3 and 4.4, we introduced three removal and four insertion heuristics. We now explain how heuristics are selected at each iteration of the ALNS algorithm. Because it is difficult to determine *a priori* which removal and insertion strategies would be more efficient, we follow the adaptive control strategy introduced by Ropke and Pisinger [23]. The principle is to assign a weight w_i $(i = 1, \ldots, 6)$ to each heuristic and to periodically adjust these weights according to the successes of the heuristic. The selection of removal and insertion heuristics is then made using a roulette wheel mechanism based on these weights. Weight evolution is managed as follows:

1. All weights are initialized to the same value at the beginning of the search.
2. The concept of segment is introduced to decide of when updating weights. A segment represents a fixed number of ALNS iterations. An update is performed at the end of each segment.
3. The update is based on a score reached on the segment for the different heuristics. The score sc_i of heuristic i $(i = 1, \ldots, 6)$ is set to zero at the beginning of the segment. At each iteration, the scores of the selected removal and insertion heuristics are increased by a value that depends on the quality of the solution s obtained:
   - μ_1 if s is a new global best solution;
   - μ_2 if s is accepted and improves the current solution;
   - μ_3 if s is accepted but has a total cost worse than the current solution;
   - 0 otherwise.
4. Given the scores sc_i, the weights are updated with the following formula:

$$w_i \leftarrow w_i \times (1 - r) + r \times \frac{sc_i}{\eta_i} \tag{5}$$

   where η_i is the number of times heuristic i has been selected on the segment and r is a *reaction* factor in $[0, 1]$ that controls how the score reacts to the effectiveness of the heuristics.

4.6. Acceptance criteria

To avoid getting trapped early in a local optimum, a simulated annealing mechanism is implemented. It consists of accepting a deteriorating solution s with a probability

$$\exp\left(-\frac{cost(s) - cost(s_{curr})}{T}\right) \tag{6}$$

where s_{curr} is the current solution, $cost(.)$ is the cost of solutions and T is the temperature. Improving solutions are always accepted. The temperature starts at a value T_{start}, fixed so that a solution 5% worse than the initial solution s_{init} has a probability 50% of being selected. Then, the temperature is decreased at every iteration by a factor γ_3, with $0 < \gamma_3 < 1$.

5. Computational experiments

In this section, we describe our experimental computations. We first present, in Section 5.1, the benchmark instances that we use. In Section 5.2, we then explain how ALNS parameters have been tuned. In Section 5.3, we evaluate the performance of the ALNS heuristic and the impact of road-network information on solution quality. In section 5.4, we present some sensitivity analyses to justify the integration of some components in the method.

Algorithms are implemented in C++. Tests are run on an Intel CORE i5 2.6 GHz computer with 4GB of memory.

5.1. Benchmark instances

In our experiments, we use four classes of instances provided by Ben Ticha et al. [6].

SOL. A first class consists of 90 instances derived from a subset of Solomon's VRPTW benchmark instances [29]: 45 instances with 25 customers and 45 instances with 50 customers. To generate these instances, Ben Ticha et al. [6] first modify travel times. For that matter, they draw random numbers correlated with Euclidean distances. Three correlation degrees are used: no-correlation (NC), weak correlation (WC) and strong correlation (SC). Multigraphs are then constructed by computing the set of efficient paths between every pair of nodes. Other parameters are not modified. Note that these instances are not stricly VRPTW$_{RN}$ instances as the multigraphs are not computed from road networks.

LET. A second class of 30 instances was initially provided by Letchford et al. [12]. These instances are generated from sparse graphs that simulate urban road networks. Four graphs are used, with different sizes $|V_{RN}| \in \{25, 50, 75, 100\}$ for the node set. The probability that a node is also a customer is 66%. Travel costs are given by the Euclidean distance and travel times are defined in correlation with these costs. Three different levels of correlation are used: NC, WC and SC. Customer time windows are narrow (NTW) or wide (WTW).

NEWLET. A third class of 45 instances was generated by Ben Ticha et al. [6] using the same procedure as Letchford et al. [12] but decreasing the density of customers. Three series of five road-network graphs are constructed: five with 25 customers and 100 nodes, five with 50 customers and 100 nodes, five with 50 customers and 200 nodes. For each graph, three degrees of correlations are defined for travel times: NC, WC, SC.

AIX. A fourth class of 12 instances was generated by Ben Ticha et al. [6] using real spatial data from the region of Aix-en-Provence (south of France). The first graph (Z1) represents the urban area and has 5,437 nodes. The second graph (Z2) includes the city and its surroundings, and has 19,500 nodes. Each arc comes with two attributes: length and maximal speed. These two attributes are used to define travel costs (length) and travel times (length divided by speed). Six instances are generated from each graph: two instances with 25 customers, two instances with 50 customers and two instances with 75 customers.

This yields a total of 177 instances. For more details on instance characteristics, readers are referred to [6].

5.2. Parameter tuning

We perform a first set of experiments to adjust parameters of the ALNS algorithm (see the list of these parameters and the selected values in Table 1). To this aim, we select a subset of 27 representative instances: SOL instances r101, r105, c103, c104, rc101, rc105 with 50 customers and NEWLET instances 1, 2 and 3 with 50 customers and 100 nodes; these 9 instances are considered for the three correlation levels NC, WC and SC. This total of 27 instances represents 15% of the benchmark instances. Furthermore, two out of the four instance classes are not represented. We believe that it permits to avoid overlearning from the tuning.

With these instances, we proceed as follows. We first tune the parameters of the adapted Shaw removal heuristic. We apply the ALNS scheme limited to this removal heuristic and to the greedy insertion heuristic. We successively focus on one of the parameters and try a number of values for this parameter. For each value, the tuning instances are solved five times; the value that provides the best average solution quality is kept.

We apply the same methodology for the worst removal heuristic. Other parameters are fixed in the same way, one by one, but using the complete ALNS scheme instead of using single removal and insertion heuristics.

Table 1: Parameter values

Operator	Parameter	Selected value
Shaw removal	Weight associated with cost: α_1	4
	Weight associated with service time: α_2	5
	Weight associated with demand: α_3	3
	Weight associated with insertion positions: α_4	10
	Randomness degree: γ_1	6
Worst removal	Randomness degree: γ_2	5
Adaptive strategy	Initial weights	100
	Gain for a new global best solution: μ_1	500
	Gain for an improving solution: μ_2	200
	Gain for an accepted non-improving solution: μ_3	150
	Reaction factor: r	0.1
Acceptance method	Cooling rate: γ_3	0.99975

5.3. Computational results

In this section, we evaluate the performance of the ALNS algorithm. We compare the solutions obtained with this algorithm to optimal solutions, when these solutions are available. Optimal solution values are reported by Ben Ticha et al. [6] and computed using a branch-and-price algorithm. We also compare the ALNS algorithm to two other heuristic schemes: MC and MT. In both schemes, a customer-based graph is constructed from the multigraph by keeping at most one arc

between every pair of nodes. In MC, the min-cost arc is kept. In MT, the min-time arc is kept. Then, in both cases, the resulting VRPTW is solved exactly with a branch-and-price algorithm. Note that these two schemes are heuristic because the customer-based graphs do not capture all the available information. Note also that these comparisons also give insights on the interest of defining travel information at the road-network level instead of using customer-based graphs.

For each instance, the ALNS algorithm is applied 10 times and we report both the best and average solution costs and solution times. Computing times for the branch-and-price algorithms are limited to 7,500 seconds. Tables 2, 3, 4, 5 and 6 report the results obtained on instances of class SOL with 25 nodes, SOL with 50 nodes, LET, NEWLET and AIX, respectively. In these tables, Column BKS provides the value of best known solutions, *i.e.*, the best among the solutions found with the branch-and-price algorithm of Ben Ticha et al. [6], the 10 found by ALNS, and the two found with MC and MT. Values in bold indicate that the solution is known to be optimal. Columns Gap(%) give the percentage gap between the solution returned by each heuristic method and BKS, computed as follows:

$$Gap = \frac{solution\ cost\ with\ the\ heuristic - best\ known\ solution\ cost}{best\ known\ solution\ cost} \times 100 \quad (7)$$

For methods MC and MT, values are in italic when the branch-and-price algorithm (applied on the customer-based graph) did not finish in 7,500 seconds. Columns CPU(s) indicate the computing times in seconds, for the different methods. For a better readability, computing times are not reported for methods MT and MC. Basically, they have the same order of magnitude as those reported for optimal solutions. These values can be found in [6]. Also CPU times are replaced by – when the exact branch-and-price algorithm was not able to find the optimal solution in 7,500 seconds.

The first columns of the tables precise the instance characteristics. Column Corr indicates the correlation degree: NC, WC or SC. Column Instance gives the instance name. For class LET, the first number is $|V_{RN}|$ and the second number is n; instance names finish with the type of time windows: NWT or WTW. For NEWLET instances, $|V_{RN}|$ and n are provided in Columns $|V_{RN}|$ and n, respectively. The number of customers n is also reported in Column n for AIX instances. Finally, when several instances have the same characteristics, the instance index is given in Column Inst.

Evaluation of the ALNS heuristic

Tables 2 to 6 demonstrate the effectiveness of the ALNS heuristic. Regarding the best run, optimal solutions are found for 108 out of the 148 instances for which the optimal solution is known. The average gap on the remaining instances is 0.4% and the maximal gap is 1.8%. On average, the ALNS algorithm is a little bit less effective, the average gap for this algorithm is 1.1%. As expected, the smaller the customer set, the better the results: all instances with 25 customers are solved optimally. Conversely, the effectiveness of the method is comparable for the four classes of instances, which tends to demonstrate its robustness.

Comparisons with the MC and MT heuristics are clearly in favor of the ALNS. ALNS best finds better or equivalent solution for 163 out of 177 instances against MC and for 173 instances against MT. ALNS avg. finds better or equivalent solution for 124 instances against MC and for 133 instances against MT. Furthermore, the customer-based graph constructed in MC does not admit any feasible solution for 8 instances.

Computing times are globally better for the ALNS heuristic. The behavior of the branch-and-price algorithms are very unpredictable. Instances of the same class and with the same characteristics can be solved in a few seconds or not be solved in two hours. On the contrary, computing

Table 2: Results for class SOL (instances with 25 customers)

			OPT	ALNS best	ALNS avg.		MC	MT
Corr	Instance	BKS	CPU(s)	Gap(%)	CPU(s)	Gap(%)	Gap(%)	Gap(%)
NC	r101-025	**690.4**	0.6	0.0	8.1	0.1	0.0	85.7
	r102-025	**588.7**	1.7	0.0	10.4	0.6	1.0	37.5
	r103-025	**491.3**	12.7	0.0	12.1	1.2	0.0	48.1
	r104-025	**507.3**	31	0.0	15.7	0.2	0.0	33.3
	r105-025	**642.8**	2.3	0.0	9.7	0.2	1.6	55.9
	c101-025	279.2	–	0.0	13.7	0.0	0.1	133.4
	c102-025	238.6	–	0.0	22.8	0.0	*10.6*	111.2
	c103-025	**202.0**	2,197	0.0	27.1	0.0	10.8	80.4
	c104-025	195.1	–	0.0	43	0.0	0.0	78.8
	c105-025	**224.0**	54.6	0.0	19.2	0.0	3.5	116.6
	rc101-025	**671.1**	0.7	0.0	7.6	0.0	10.3	69.9
	rc102-025	**558.0**	10.9	0.0	11.8	0.0	12.6	53.7
	rc103-025	**545.9**	613.9	0.0	15.4	1.3	2.2	50.8
	rc104-025	**420.4**	2,728	0.0	18.8	0.3	5.1	25.2
	rc105-025	**575.7**	9.8	0.0	8.4	0.4	3.8	38.7
WC	r101-025	**682.0**	0.2	0.0	7.8	0.2	0.0	10.9
	r102-025	**572.6**	1.2	0.0	7.1	0.1	0.0	6.5
	r103-025	**476.2**	2.3	0.0	7.1	0.1	0.0	6.0
	r104-025	**481.0**	4.7	0.0	8.2	0.3	0.0	3.8
	r105-025	**601.0**	0.9	0.0	6.8	0.0	0.0	11.3
	c101-025	**250.7**	206.5	0.0	6.7	0.4	4.8	18.1
	c102-025	229.9	–	0.0	11.3	0.0	*1.0*	14.6
	c103-025	**199.1**	480.4	0.0	14	0.1	0.0	26.4
	c104-025	192.8	–	0.0	10	0.0	0.0	6.6
	c105-025	**216.6**	27.3	0.0	7.8	0.0	0.0	30.2
	rc101-025	**561.1**	7.7	0.0	5.7	0.0	8.6	11.6
	rc102-025	**552.4**	983.5	0.0	8.1	0.2	13.6	3.9
	rc103-025	**461.8**	713.3	0.0	8.8	1.6	2.5	3.4
	rc104-025	**398.4**	835	0.0	9.9	0.3	0.2	2.7
	rc105-025	**555.4**	2.8	0.0	6.4	0.0	1.5	8.4
SC	r101-025	**684.7**	0.2	0.0	7.9	0.0	0.0	0.0
	r102-025	**570.8**	0.5	0.0	6.7	0.0	0.0	1.1
	r103-025	**458.3**	0.9	0.0	5.2	0.0	1.8	0.0
	r104-025	**420.2**	4.2	0.0	5.6	0.0	0.0	0.6
	r105-025	**549.3**	0.8	0.0	5.4	0.0	0.0	0.1
	c101-025	**216.6**	6.4	0.0	4.7	0.0	0.0	1.8
	c102-025	**193.1**	5.5	0.0	5.2	0.0	0.0	0.0
	c103-025	**193.1**	96.4	0.0	6.1	0.0	0.0	1.1
	c104-025	**189.7**	1,717.6	0.0	6.4	0.0	0.0	0.0
	c105-025	**194.1**	0.5	0.0	4.1	0.0	0.0	0.0
	rc101-025	**507.5**	13.9	0.0	5.4	0.0	4.9	0.8
	rc102-025	**443.6**	397.7	0.0	5.6	0.0	0.0	0.0
	rc103-025	**342.2**	5.9	0.0	5.8	0.2	0.0	0.1
	rc104-025	**314.9**	13.1	0.0	5.9	0.1	0.0	0.0
	rc105-025	**457.6**	21.7	0.0	5.8	0.2	0.0	0.2

NOTE : – *indicates that the corresponding branch-and-price algorithm could not solve the instance in 7,500 seconds*

Table 3: Results for class SOL (instances with 50 customers)

			OPT	ALNS best	ALNS avg.		MC	MT
Corr	Instance	BKS	CPU(s)	Gap(%)	CPU(s)	Gap(%)	Gap(%)	Gap(%)
NC	r101	**1,317.3**	17.6	1.1	29.6	2.3	0.4	72.7
	r102	**1,148.3**	52.8	0.1	36.7	2.0	0.5	45.6
	r103	**952.7**	593.7	1.0	59.4	2.2	2.3	34.1
	r104	**770.5**	6,798.9	1.3	137	2.5	3.2	*68.1*
	r105	**1,162.8**	26.9	1.8	36.9	2.9	0.6	75.6
	c101	599.2	–	0.0	36.9	0.3	*9.9*	123.2
	c102	506.0	–	0.0	74.7	0.5	*22.4*	120.1
	c103	426.2	–	0.0	126.2	2.5	*17.2*	72.3
	c104	394.2	–	0.0	413.9	2.4	*13.5*	*146.6*
	c105	511.0	–	0.0	67.3	0.2	*13.3*	140.1
	rc101	**1,375.8**	108	0.2	27.5	0.9	15.0	68.0
	rc102	**1,164.1**	321.6	0.9	40.5	2.2	9.5	50.6
	rc103	**1,063.1**	6,821.9	1.1	56.5	2.8	0.9	37.7
	rc104	829.3	–	0.0	89.5	1.6	0.5	*75.3*
	rc105	**1,229.0**	2,358.4	0.9	37.1	1.6	9.4	49.7
WC	r101	**1,179.4**	1.2	0.3	26.8	1.4	3.3	29.0
	r102	**1,075.1**	6.3	0.6	28.7	1.5	1.6	10.6
	r103	**948.2**	65.4	0.3	32.5	2.6	1.7	11.4
	r104	**769.3**	1,304.3	0.4	47.1	3.0	0.0	6.6
	r105	**1,062.3**	17	0.5	27	1.7	0.4	9.4
	c101	535.4	–	0.0	27.3	1.5	*5.3*	*69.6*
	c102	468.1	–	0.0	51.8	1.0	*22.8*	*53.5*
	c103	402.0	–	0.0	66.5	4.1	*16.9*	*81.8*
	c104	372.7	–	0.0	143.2	2.1	*20.4*	*60.3*
	c105	486.0	–	0.0	42.3	0.8	*4.0*	23.3
	rc101	**1,222.2**	110.3	0.0	23	0.6	7.8	7.8
	rc102	1,172.4	–	0.0	29.1	1.4	5.3	*33.2*
	rc103	996.2	–	0.0	31.4	2.3	3.4	*50.5*
	rc104	892.2	–	0.0	39.7	1.7	*25.1*	*17.2*
	rc105	**1,034.4**	68.4	0.0	26.1	0.5	9.6	8.4
SC	r101	**1,085.7**	1	0.1	23.9	1.0	1.2	1.0
	r102	**929.8**	6.8	0.0	23.6	0.6	0.0	0.7
	r103	**827.1**	71.1	0.0	24.7	1.1	0.0	0.1
	r104	718.8	–	0.0	28.8	1.8	*0.0*	*13.5*
	r105	**932.7**	14.9	0.2	23.5	1.0	0.8	0.2
	c101	**405.4**	106.4	0.0	21.1	0.0	0.0	0.5
	c102	**366.9**	53.8	0.0	25.7	0.0	0.0	2.0
	c103	**368.8**	699.9	0.0	29	0.6	0.0	1.1
	c104	365.4	–	0.0	46.6	1.7	*28.1*	*21.2*
	c105	**367.9**	12	0.0	18.5	0.0	0.0	0.5
	rc101	**990.9**	2,385.9	0.0	21.4	0.2	0.0	1.0
	rc102	916.9	–	0.0	22.5	1.0	*38.9*	*31.9*
	rc103	871.4	–	0.0	22.4	1.4	*26.1*	*25.9*
	rc104	714.4	–	0.0	30	0.8	*24.8*	*14.2*
	rc105	**940.9**	6,544.4	0.0	20.6	1.1	0.0	0.7

NOTE : *– indicates that the corresponding branch-and-price algorithm could not solve the instance in 7,500 seconds*

Table 4: Results for **LET** instances

			OPT	ALNS best	ALNS avg.		MC	MT
Instance	Corr	BKS	CPU(s)	Gap(%)	CPU(s)	Gap(%)	Gap(%)	Gap(%)
25_16_NTW	NC	**1,252**	0.1	0.0	1.8	0.0	0.0	8.6
25_16_WTW	NC	**1,252**	0.1	0.0	2.1	0.0	0.0	8.6
25_16_NTW	WC	**1,252**	0.1	0.0	1.6	0.0	0.0	1.0
25_16_WTW	WC	**1,252**	0.1	0.0	1.7	0.0	0.0	1.0
25_16_NTW	SC	**1,252**	0.1	0.0	1.5	0.0	0.0	0.0
25_16_WTW	SC	**1,252**	0.1	0.0	1.6	0.0	0.0	0.0
50_33_NTW	NC	**2,137**	0.5	0.0	5.6	0.0	2.2	2.6
50_33_WTW	NC	**2,072**	398	0.0	6.6	0.0	0.0	1.1
50_33_NTW	WC	**2,293**	1.8	0.0	6.3	2.4	0.0	0.1
50_33_WTW	WC	2,095	–	0.0	7.6	0.0	0.0	0.1
50_33_NTW	WC	**2,453**	0.6	0.0	7.1	0.0	Infeasible	5.6
50_33_WTW	WC	**2,169**	50.6	0.0	9.2	0.7	Infeasible	4.2
50_33_NTW	SC	**2,438**	19.4	0.0	6	0.0	0.0	0.3
50_33_WTW	SC	**2,104**	533.9	0.0	6.3	0.1	0.0	0.0
75_50_NTW	NC	**3,346**	0.7	0.0	14.8	0.0	Infeasible	9.2
75_50_WTW	NC	**3,233**	152.3	0.0	16.9	3.8	2.6	4.8
75_50_NTW	WC	**3,277**	1.4	0.0	15.7	0.0	0.0	1.2
75_50_WTW	WC	2,999	–	0.0	20.2	2.9	0.1	0.2
75_50_NTW	WC	**3,169**	3.2	0.0	18.4	2.2	2.9	13.3
75_50_WTW	WC	**2,951**	353.5	1.7	25	5.1	1.7	10.2
75_50_NTW	SC	**3,266**	1.1	0.0	12.9	0.3	0.0	0.1
75_50_WTW	SC	**2,949**	5,305.9	1.2	14.6	5.2	0.0	0.0
100_66_NTW	NC	**3,379**	64.7	0.0	27.2	2.7	Infeasible	7.8
100_66_WTW	NC	**3,184**	550.1	1.8	30.8	7.7	5.7	9.9
100_66_NTW	WC	**3,373**	6.2	0.0	23.8	2.7	Infeasible	5.8
100_66_WTW	WC	**3,223**	4,391.6	0.0	27.7	6.7	Infeasible	7.0
100_66_NTW	WC	**3,308**	13	0.0	27.2	0.4	Infeasible	8.6
100_66_WTW	WC	**3,153**	593.6	1.0	34	5.6	Infeasible	8.8
100_66_NTW	SC	**3,319**	4.5	0.0	20.9	3.7	0.0	0.9
100_66_WTW	SC	3,215	–	0.0	25.9	8.0	16.5	20.1

NOTE *: – indicates that the corresponding branch-and-price algorithm could not solve the instance in 7,500 seconds*

times are rather regular for the ALNS. They are relatively high even for small instances, but increase slowly with the size of the instances. For example, SOL instances are solved in 10 seconds on average when $n = 25$, and 35 seconds when $n = 50$.

Impact of road-network information

Garaix et al. [4], Ben Ticha et al. [6] and Lai et al. [5] have presented extensive computational results that show the improvements achieved when travel information is defined at the road-network level. Tables 2 to 6 consolidate these findings on a dozen of larger (or more difficult) instances that could not be solved with the branch-and-price method developed in Ben Ticha et al. [6].

On these instances, the gaps observed for the heuristic methods based on customer-based graphs oscillate a lot. They vary between 0.0% and 38.9% for MC, between 0.0% and 146.6% for MT. On average, they are respectively equal to 4.4% for MC and 19.1% for MT.

5.4. Sensitivity analyses

In this section, we present some sensitivity analyses. We carry out these tests to check the impact of some components of the ALNS algorithm. We also aim at identifying the respective contributions of the removal and insertion heuristics during the search. We limit these tests to instances of class SOL and of class NEWLET with $|V_{RN}| = 100$.

Table 5: Results for **NEWLET** instances

					OPT	ALNS best	ALNS avg.		MC	MT
$\|V_{RN}\|$	n	Corr	Inst	BKS	CPU(s)	Gap(%)	CPU(s)	Gap(%)	Gap(%)	Gap(%)
100	25	NC	1	**1,828.7**	6.3	0.0	7.4	0.0	6.8	3.6
			2	**2,109.6**	1.4	0.0	7.5	0.4	0.0	11.9
			3	**2,200.9**	5.7	0.0	9.1	0.0	2.4	15.6
			4	**2,139.5**	2.8	0.0	6.8	0.0	0.4	6.0
			5	**1,869.2**	2.5	0.0	10.5	0.0	1.2	11.8
		WC	1	**1,742.8**	2	0.0	5.5	0.0	0.0	3.4
			2	**1,510.2**	71.5	0.0	8	0.0	1.7	5.9
			3	**2,056.3**	3.1	0.0	6.1	0.7	2.6	0.4
			4	**1,749.7**	0.8	0.0	5.1	0.0	0.0	2.7
			5	**2,173.0**	10.4	0.0	6.6	0.3	3.2	3.3
		SC	1	**2,075.4**	21.1	0.0	4.5	0.0	0.0	0.4
			2	**2,108.0**	1.3	0.0	4.7	1.0	0.0	0.2
			3	**1,770.8**	9.7	0.0	6.2	1.3	0.0	2.1
			4	**2,029.1**	0.6	0.0	4.8	0.0	0.0	0.0
			5	**2,108.2**	0.6	0.0	5.5	0.0	0.0	0.7
	50	NC	1	**2,563.3**	242	0.7	23.3	1.6	2.0	9.6
			2	**3,320.5**	2,296.5	0.0	31.2	0.6	4.9	2.5
			3	**2,729.1**	1,045.6	0.1	28.3	1.1	4.2	3.3
			4	**2,616.4**	399.9	0.0	29.2	0.5	4.2	7.5
			5	**2,948.6**	82.2	0.0	26.4	2.3	2.8	7.1
		WC	1	**2,626.8**	104.8	0.0	22	0.8	0.0	5.7
			2	**2,890.1**	245.5	0.0	19.8	0.5	3.2	3.2
			3	**2,516.7**	79.1	0.2	19.3	0.5	3.2	1.9
			4	**2,398.3**	42.2	0.0	19.1	0.0	0.0	4.9
			5	**2,427.2**	291.3	0.0	19.9	0.5	0.0	2.0
		SC	1	**3,177.5**	1,120.1	1.0	16.6	1.7	0.3	0.0
			2	**3,116.5**	424.2	0.4	15.6	1.4	0.0	0.1
			3	**3,174.3**	21.4	0.1	15.7	1.9	4.1	0.1
			4	**2,977.5**	44.9	0.0	16.8	0.4	0.0	0.5
			5	**3,352.2**	5.4	0.0	15.5	0.8	0.0	0.4
200	50	NC	1	**4,125.8**	1,659.3	0.3	56.8	1.6	8.3	6.6
			2	**4,000.5**	191.3	0.0	48.3	0.6	5.0	12.8
			3	**4,277.9**	1,045.1	0.3	39.2	2.5	4.2	8.0
			4	**4,068.4**	6,775.5	0.5	60.7	3.6	5.9	6.7
			5	4,674.7	–	0.0	52.2	1.3	4.0	9.4
		WC	1	**4,358.5**	4,378.7	0.8	40.8	3.0	3.6	6.6
			2	**3,894.4**	181.7	0.0	32	1.3	3.6	6.3
			3	**4,050.5**	651.6	0.6	38.1	3.1	4.2	5.4
			4	**3,683.4**	695.4	0.0	44.2	2.7	4.1	6.9
			5	**4,327.3**	453.4	0.0	42.3	2.6	9.7	9.0
		SC	1	4,539.9	–	0.0	28.5	0.5	3.8	0.7
			2	**4,416.8**	3,892.8	0.2	26.4	4.4	0.0	1.0
			3	**4,282.3**	203.3	0.1	23.2	2.3	0.0	0.3
			4	**3,719.8**	81.8	0.0	22.3	2.3	0.0	0.6
			5	**3,765.7**	67.5	0.0	20	1.7	0.0	0.2

NOTE *: – indicates that the corresponding branch-and-price algorithm could not solve the instance in 7,500 seconds*

Table 6: Results for **AIX** instances

				OPT	ALNS best	ALNS avg.		MC	MT
	n	Instance	BKS	CPU(s)	Gap(%)	CPU(s)	Gap(%)	Gap(%)	Gap(%)
Z1	25	1	**44,931**	1.7	0.0	8.4	0.0	3.6	8.4
		2	**44,574**	0.8	0.0	7.7	0.0	8.6	6.7
	50	1	**79,925**	13.4	0.2	31	0.6	2.4	5.5
		2	**84,722**	18.8	0.1	30.4	1.0	1.6	4.6
	75	1	**110,718**	131.4	0.8	65.9	1.8	0.5	5.4
		2	**101,700**	73.4	1.7	67.1	2.5	0.7	6.3
Z2	25	1	**123,592**	1.2	0.0	5.8	0.0	7.1	11.1
		2	**192,625**	1.1	0.0	7.4	0.0	1.7	9.6
	50	1	**271,836**	22.7	0.0	28.8	1.0	0.1	10.5
		2	**362,426**	13.3	0.3	30	0.9	2.3	9.0
	75	1	**390,642**	174.1	1.4	57.9	2.2	11.7	4.3
		2	**374,845**	102.6	1.4	54.2	2.8	0.9	4.6

NOTE *: – indicates that the algorithm has not terminated within 7,500 seconds*

Evaluation of insertion positions in the adapted Shaw removal heuristic

The adapted Shaw removal heuristic is based on values $R_{ij}(s)$ that measure the similarity between customers i and j in a solution s. $R_{ij}(s)$ is composed of four terms (see Equation 2). In Section 4.3, we underlined the negative impact that the last term (evaluation of insertion positions) might have on computing times.

To evaluate this impact, we apply the ALNS algorithm with the following modifications:

- The portfolio of removal heuristics is limited to the adapted Shaw removal heuristic;
- The portfolio of insertion heuristics is limited to the greedy insertion heuristic;
- The similarity measure includes ($\alpha_4 > 0$) or not ($\alpha_4 = 0$) the term evaluating insertion positions. When $\alpha_4 > 0$, it is defined as detailed in Table 1.

Each instance is solved five times with the two methods. Tables 7 and 8 report aggregated results for SOL and NEWLET instances, respectively. Column Gap(%) provides the percentage gaps with best known solution values. Column CPU(s) gives CPU times in seconds. These two values are reported for the best run (best gap out of five and best CPU time out of five) and on average.

Table 7: Sensitivity analysis on the similarity measure for SOL instances

		Best run				Average (5 runs)			
		$\alpha_4 = 0$		$\alpha_4 \neq 0$		$\alpha_4 = 0$		$\alpha_4 \neq 0$	
n	Corr	Gap(%)	CPU(s)	Gap(%)	CPU(s)	Gap(%)	CPU(s)	Gap(%)	CPU(s)
25	NC	0.6	10.6	0.5	10.6	1.3	11.1	0.9	11.1
	WC	0.3	4.9	0.2	4.9	0.9	5.1	0.6	5.1
	SC	0.2	2.9	0.1	2.9	0.8	3.0	0.5	3.0
50	NC	6.1	49.9	4.5	50.2	8.2	51.7	6.6	52.4
	WC	3.5	19.7	3.2	19.8	4.8	21.4	4.1	21.9
	SC	1.7	9.3	1.2	9.3	2.6	9.6	2.2	9.7

Table 8: Sensitivity analysis on the similarity measure for NEWLET instances

		Best run				Average (5 runs)			
		$\alpha_4 = 0$		$\alpha_4 \neq 0$		$\alpha_4 = 0$		$\alpha_4 \neq 0$	
n	Corr	Gap(%)	CPU(s)	Gap(%)	CPU(s)	Gap(%)	CPU(s)	Gap(%)	CPU(s)
25	NC	0.5	5.4	0.3	5.3	0.5	5.5	0.3	5.5
	WC	0.0	4.0	0.0	4.0	1.0	4.3	0.8	4.3
	SC	0.6	3.1	0.6	3.1	0.8	3.2	0.7	3.3
50	NC	4.7	16.9	4.1	16.8	5.2	17.7	4.2	17.4
	WC	2.0	11.2	1.4	11.2	4.4	11.8	3.4	12.0
	SC	6.3	9.2	4.6	9.5	7.1	9.9	6.3	10.1

From Tables 7 and 8, we can observe that considering insertion positions ($\alpha_4 > 0$) in the similarity measure improves significantly the quality of solutions for a very limited additional amount of computing times. Incidentally, it illustrates the efficiency of the evaluation methods described in Section 4.1.

Non-myopic insertion heuristic

The non-myopic insertion heuristic computes values $I_i^+(\Pi)$ to evaluate the insertion of a customer i in a list of routes Π. The computation of $I_i^+(\Pi)$ necessitates executing many times the evaluation method described in Section 4.1 (see Equation 4). To evaluate the impact of this insertion heuristic, we run the ALNS algorithm with or without the heuristic. Each instance is solved five times with the two methods. Tables 9 and 10 provide aggregated results for SOL and NEWLET instances, respectively. Column headings are the same as in Tables 7 and 8.

Table 9: Sensitivity analysis on the non-myopic insertion heuristic for SOL instances

		Best run				Average (5 runs)			
		With non-myopic		Without non-myopic		With non-myopic		Without non-myopic	
n	Corr	Gap(%)	CPU(s)	Gap(%)	CPU(s)	Gap(%)	CPU(s)	Gap(%)	CPU(s)
25	NC	0.0	12.1	0.2	8.7	0.1	16.3	0.6	8.9
	WC	0.0	7.4	0.1	4.6	0.0	8.4	0.3	4.7
	SC	0.0	4.7	0.0	3.1	0.0	5.7	0.1	3.2
50	NC	0.9	66.6	2.7	34.9	2.2	84.6	3.8	35.8
	WC	0.3	25.6	1.7	17.1	1.6	42.8	2.6	17.5
	SC	0.0	12.9	0.5	8.9	0.6	25.5	1.2	9.4

Table 10: Sensitivity analysis on the non-myopic insertion heuristic for NEWLET instances

		Best run				Average (5 runs)			
		With non-myopic		Without non-myopic		With non-myopic		Without non-myopic	
n	Corr	Gap(%)	CPU(s)	Gap(%)	CPU(s)	Gap(%)	CPU(s)	Gap(%)	CPU(s)
25	NC	0.0	6.7	0.0	5.0	0.0	7.4	0.0	5.1
	WC	0.0	4.6	0.0	3.9	0.2	4.9	0.2	3.9
	SC	0.0	3.6	0.0	3.2	0.2	4.2	0.5	3.3
50	NC	0.1	22.6	5.0	15.1	1.0	24.2	5.0	15.4
	WC	0.1	14.2	2.6	11.2	0.4	15.6	4.2	11.5
	SC	0.3	11.1	3.8	9.7	1.2	13.2	5.3	10.0

Tables 9 and 10 show the important impact of the non-myopic insertion heuristic on solution quality. Without this heuristic, solution costs can sometimes be increased by more than 5%. This heuristic has however also an impact on computing times, which are sometimes more than doubled.

Contribution of the different removal and insertion heuristics

In Tables 11 and 12, we report the contribution of each insertion-removal combination in the ALNS. Each column corresponds to a combination, with the name of the removal heuristic on the first row and the name of the insertion heuristic on the second. For each combination, four criteria are analyzed: the number of accepted solutions that improved the best solution (row *Best*), the number of accepted solutions that improved the current solution (row *Improving*), the number of accepted solutions that did not improve the current solution (row *Non-Improving*) and the total computing time used by each combination along the search (row *Computing time*). These criteria are expressed in percentage (each row reaches 100%).

Table 11: Contribution of removal-insertion combinations for SOL instances

		Random				Worst				Shaw			
n	Indicator	Greedy	Regret	Non-myopic	Simple	Greedy	Regret	Non-myopic	Simple	Greedy	Regret	Non-myopic	Simple
25	Best	5.1 %	10.7 %	8.1 %	0.0 %	10.3 %	22.6 %	15.5 %	0.0 %	4.8 %	13.7 %	9.2 %	0.0 %
	Improving	7.2 %	11.8 %	10.4 %	0.0 %	10.1 %	15.3 %	13.0 %	0.0 %	7.8 %	12.9 %	11.5 %	0.0 %
	Non-Improving	10.6 %	8.8 %	9.5 %	0.1 %	14.0 %	13.1 %	13.4 %	0.2 %	11.2 %	9.0 %	9.9 %	0.1 %
	Computing time	5.5 %	6.4 %	16.8 %	2.1 %	6.2 %	7.1 %	19.0 %	2.3 %	6.5 %	7.3 %	18.3 %	2.5 %
50	Best	6.1 %	8.7 %	8.0 %	0.0 %	11.9 %	21.2 %	16.1 %	0.0 %	7.3 %	11.5 %	9.2 %	0.0 %
	Improving	8.0 %	10.7 %	9.6 %	0.0 %	11.4 %	15.2 %	13.3 %	0.0 %	8.9 %	12.1 %	10.8 %	0.0 %
	Non-Improving	10.4 %	8.6 %	8.8 %	0.1 %	15.2 %	13.5 %	13.4 %	0.2 %	11.1 %	9.0 %	9.6 %	0.1 %
	Computing time	6.4 %	7.0 %	14.3 %	2.9 %	7.3 %	8.2 %	16.6 %	3.3 %	7.3 %	8.0 %	15.5 %	3.1 %

Table 12: Contribution of removal-insertion combinations for NEWLET instances

		Random				Worst				Shaw			
n	Indicator	Greedy	Regret	Non-myopic	Simple	Greedy	Regret	Non-myopic	Simple	Greedy	Regret	Non-myopic	Simple
25	Best	4.8 %	8.4 %	4.7 %	0.0 %	13.3 %	22.0 %	18.3 %	0.0 %	9.0 %	6.3 %	6.5 %	0.0 %
	Improving	8.4 %	9.4 %	10.0 %	0.0 %	11.6 %	15.3 %	15.0 %	0.0 %	9.7 %	9.6 %	11.0 %	0.0 %
	Non-Improving	10.1 %	9.9 %	8.8 %	0.1 %	14.6 %	14.6 %	13.9 %	0.1 %	9.1 %	9.9 %	8.8 %	0.0 %
	Computing time	6.5 %	7.1 %	13.6 %	3.2 %	7.5 %	8.3 %	16.5 %	3.8 %	7.3 %	7.8 %	14.9 %	3.5 %
50	Best	7.2 %	5.3 %	9.6 %	0.0 %	13.5 %	16.8 %	24.1 %	0.0 %	6.7 %	3.1 %	13.7 %	0.0 %
	Improving	8.1 %	6.8 %	11.6 %	0.0 %	13.1 %	15.0 %	18.1 %	0.0 %	8.5 %	5.9 %	12.9 %	0.0 %
	Non-Improving	9.0 %	9.7 %	8.7 %	0.1 %	14.2 %	14.7 %	14.7 %	0.1 %	9.4 %	11.0 %	8.3 %	0.1 %
	Computing time	5.4 %	5.5 %	16.8 %	2.5 %	6.5 %	6.9 %	19.4 %	2.9 %	6.4 %	6.7 %	17.9 %	3.0 %

The main observation that can be made with these tables is that almost all heuristics contribute to the improvement of solutions. Except for the simple insertion heuristic, the two tables report significant percentages for all the heuristics on all indicators. Fortunately, the time consumed by the simple insertion heuristic is very limited. Probably, the learning mechanism is able to identify quickly that this heuristic is not effective and gives a small probability to its selection. Among the removal heuristics, the worst removal method is specially effective. It consistently permits to find around 50% and around 40% of the best and improving solutions, respectively. This improvements are furthermore obtained with a computational effort that only slightly exceeds the ones of the two other removal heuristics. Regarding insertion heuristics (simple insertion excluded), the ranking is not as clear. The regret heuristic tends to be the most effective, except for NEWLET instances of size 50, where it is the worst. The non-myopic heuristic is globally better than the random heuristic, but its computing times are higher than those of the two other heuristics. Globally, the main conclusion is still that the three heuristics are important.

6. Conclusion

Due to their numerous applications, and strong correlation to the bottom line, vehicle routing problems are critical to industry and have drawn the attention of many researchers. In many real-life circumstances, different criteria have to be considered when defining transportation plans: operational costs, traveling times or energy consumption, for example. Therefore, it is imperative to capture travel information at the road-network level. Modeling travel information with customer-based graphs may indeed furnish infeasible routes or overestimate cost. Efficient heuristic solution approaches that solve vehicle routing problems with this degree of information are however missing in the literature. Hence, we propose an ALNS algorithm, with the objective of filling this gap. We considered the VRPTW$_{RN}$ and introduced a multigraph, that captures all efficient paths between pairs of points of interest (depot, customers). The presence of parallel arcs introduces computational challenges, especially when exploring the neighborhood of a given solution: elementary operations like customer removal or insertion induce the solution of an NP-hard problem. To handle this difficulty, we proposed an incremental data structure and developed a procedure based on dynamic programming. We conducted an extensive experimental study on several set of instances with different characteristics. Numerical results showed the ability of the heuristic to find near-optimal solutions in a reasonable amount of time. In addition, results confirm the gains provided by road-network travel information compared to traditional solution approaches based on customer-based graphs. An alternative to the multigraph is to tackle directly vehicle routing problems with road-network graphs. A future study could be to investigate heuristic solution schemes on these graphs.

Acknowledgements

We warmly thank the reviewers for their suggestions that have helped improve the quality of this paper. The first author was supported by the Labex IMobS3, by the European Fund for Regional Development (FEDER Auvergne region) and by the Auvergne Region.

References

References

[1] B. De Backer, V. Furnon, P. Prosser, P. Kilby, P. Shaw, Local search in constraint programming: Application to the vehicle routing problem, in: Proc. CP-97 Workshop Indust. Constraint-Directed Scheduling, Schloss Hagenberg Austria, 1997, pp. 1–15.

[2] G. B. Dantzig, J. H. Ramser, The truck dispatching problem, Management Science 6 (1) (1959) 80–91.

[3] P. Toth, D. Vigo (Eds.), Vehicle Routing: Problems, Methods, and Applications, 2nd Edition, Vol. 18 of MOS-SIAM series on optimization, SIAM, Philadelphia, 2014.

[4] T. Garaix, C. Artigues, D. Feillet, D. Josselin, Vehicle routing problems with alternative paths: An application to on-demand transportation, European Journal of Operational Research 204 (1) (2010) 62–75.

[5] D. S. Lai, O. C. Demirag, J. M. Leung, A tabu search heuristic for the heterogeneous vehicle routing problem on a multigraph, Transportation Research Part E: Logistics and Transportation Review 86 (2016) 32–52.

[6] H. Ben Ticha, N. Absi, D. Feillet, A. Quilliot, Empirical analysis for the vrptw with a multigraph representation for the road network, Computers & Operations Research 88 (2017) 103–116.

[7] H. Ben Ticha, N. Absi, D. Feillet, A. Quilliot, Vehicle routing problems with road-network information: State of the art, Networks 72 (3) (2018) 393–406. doi:10.1002/net.21808.

[8] J. Qian, R. Eglese, Fuel emissions optimization in vehicle routing problems with time-varying speeds, European Journal of Operational Research 248 (3) (2016) 840–848.

[9] C. Orloff, A fundamental problem in vehicle routing, Networks 4 (1) (1974) 35–64.

[10] B. Fleischmann, A cutting plane procedure for the travelling salesman problem on road networks, European Journal of Operational Research 21 (3) (1985) 307–317.

[11] G. Cornuéjols, J. Fonlupt, D. Naddef, The traveling salesman problem on a graph and some related integer polyhedra, Mathematical programming 33 (1) (1985) 1–27.

[12] A. N. Letchford, S. D. Nasiri, A. Oukil, Pricing routines for vehicle routing with time windows on road networks, Computers & Operations Research 51 (2014) 331–337.

[13] H. Ben Ticha, N. Absi, D. Feillet, A. Quilliot, T. Van Woensel, A branch-and-price algorithm for the vehicle routing problem with time windows on a road network, Networks-doi:10.1002/net.21852.

[14] R. Baldacci, L. D. Bodin, A. Mingozzi, The multiple disposal facilities and multiple inventory locations rollon–rolloff vehicle routing problem, Computers & Operations Research 33 (9) (2006) 2667–2702.

[15] H. Wang, Y. Lee, Two-stage particle swarm optimization algorithm for the time dependent alternative vehicle routing problem, Journal of Applied & Computational Mathematics 3 (4) (2014) 1–9. doi:10.4172/2168-9679.1000170.

[16] M. Setak, Z. Shakeri, A. Patoghi, A time dependent pollution routing problem in multi-graph, International Journal of Engineering-Transactions B: Applications 30 (2) (2017) 234–242.

[17] M. Caramia, F. Guerriero, A heuristic approach to long-haul freight transportation with multiple objective functions, Omega 37 (3) (2009) 600–614.

[18] G. Desaulniers, O. B. G. Madsen, S. Ropke, The vehicle routing problem with time windows, in: P. Toth, D. Vigo (Eds.), Vehicle Routing: Problems, Methods, and Applications, 2nd Edition, Vol. 18 of MOS-SIAM series on optimization, SIAM, Philadelphia, 2014, Ch. 5, pp. 119–159.

[19] R. Baldacci, A. Mingozzi, R. Roberti, Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints, European Journal of Operational Research 218 (1) (2012) 1–6.

[20] B. Kallehauge, Formulations and exact algorithms for the vehicle routing problem with time windows, Computers & Operations Research 35 (7) (2008) 2307–2330.

[21] O. Bräysy, M. Gendreau, Vehicle routing problem with time windows, part I: Route construction and local search algorithms, Transportation Science 39 (1) (2005) 104–118.

[22] O. Bräysy, M. Gendreau, Vehicle routing problem with time windows, part II: Metaheuristics, Transportation Science 39 (1) (2005) 119–139.

[23] S. Ropke, D. Pisinger, An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows, Transportation Science 40 (4) (2006) 455–472.

[24] P. Shaw, Using constraint programming and local search methods to solve vehicle routing problems, in: International conference on principles and practice of constraint programming, Springer, 1998, pp. 417–431.

[25] H. Ben Ticha, N. Absi, D. Feillet, A. Quilliot, A solution method for the multi-destination bi-objectives shortest path problem, Tech. Rep. EMSE CMP-SFL 2017/5, Ecole des Mines de Saint Etienne, CMP, Gardanne, France (2017).

[26] G. Clarke, J. W. Wright, Scheduling of vehicles from a central depot to a number of delivery points, Operations Research 12 (4) (1964) 568–581.

[27] S. Irnich, G. Desaulniers, Shortest path problems with resource constraints, in: G. Desaulniers, J. Desrosiers, M. M. Solomon (Eds.), Column Generation, Springer, New York, 2005, pp. 33–65.

[28] M. W. P. Savelsbergh, Local search in routing problems with time windows, Annals of Operations Research 4 (1) (1985) 285–305.

[29] M. M. Solomon, Algorithms for the vehicle routing and scheduling problems with time window constraints, Operations Research 35 (2) (1987) 254–265.