Publié par :
Published by:
Publicación de la:

Faculté des sciences de l'administration
2325, rue de la Terrasse
Pavillon Palasis-Prince, Université Laval
Québec (Québec) Canada G1V 0A6
Tél. Ph. Tel. : (418) 656-3644
Télec. Fax : (418) 656-7047

Disponible sur Internet :
Available on Internet
Disponible por Internet :

http://www4.fsa.ulaval.ca/la-recherche/publications/documents-de-travail/

DOCUMENT DE TRAVAIL 2017-003

An Adaptive Large Neighborhood Search for the Multi-Pickup and Delivery Problem with Time Windows

Salma NACCACHE
Jean-François CÔTÉ
Leandro C. COELHO

Document de travail également publié par le Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, sous le numéro CIRRELT-2017-25

Mai 2017

Dépôt legal – Bibliothèque et Archives nationales du Québec, 2017
Bibliothèque et Archives Canada, 2017

ISBN 978-2-89524-446-2 (PDF)

An Adaptive Large Neighborhood Search for the Multi-Pickup and Delivery Problem with Time Windows

Salma Naccache[1], **Jean-François Côté**[1,2*], **Leandro C. Coelho**[1,2]

[1] Interunivesrity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)
[2] Department of Operations and Decision Systems, 2325, rue de la Terrasse, Université Laval, Québec, Canada, G1V 0A6

**Corresponding author: jean-francois.cote@cirrelt.ca*

ABSTRACT

This article investigates multi-pickup and delivery problem with time windows where a set of vehicles is used to collect and deliver a set of items defined within client requests. In this paper we formally describe, model and solve this problem. An adaptive large neighbourhood search heuristic is developed to solve the studied problem. Several new insertion operators are developed to tackle the special precedence constraints. Computational results are reported on different types of instances to study the performance of the developed heuristics in different parameter settings.

Keywords: Vehicle routing problem, multi-pickup and delivery problem with time windows, sequential ordering problem.

Acknowledgments: This research was partly supported by grants 2014-05764 and 2015-04893 from Natural Sciences and Engineering Research Council of Canada (NSERC). This support is gratefully acknowledged.

1 Introduction

In many applications, vehicles must perform several sequential pickups of one or different commodities, and once all pickups are performed, the vehicle must deliver all of them to a given location. This type of problem arises, for example, in the online (phone or internet) food ordering, in which a request consists of a set of orders to be collected from multiple restaurants to be delivered to one single client location. Another application we have observed in practice arises in the collection of cash from parking tolls: an employee leaves the depot with a key that only allows access to the cash of some tolls to be dropped in a given delivery location. He can then visit the tolls in any order, but must visit them all before delivering all the cash, which must happen before he can have access to another key.

In this paper we consider a multi-pickup and delivery problem with time windows (MPDTW), in which a set of requests is satisfied by a capacitated vehicle fleet. In each request, items are required to be picked up from different locations to be shipped and unloaded at one common delivery location. In addition, a time window (TW) is associated with each node, such that pickups and deliveries can only be performed within the node's start and end times. The depot in which the vehicles are housed also contain TWs representing its opening hours. The goal is to obtain feasible vehicle tours fulfilling the requests for pickups and deliveries, while minimizing the overall costs associated with the routing of a set of requests.

In the MPDTW, a request must be fulfilled by a single vehicle. This means that all pickups and the corresponding delivery must be performed by a single tour, possibly combined with other requests. Moreover, vehicle tours have to be developed with respect to precedence constraints, while reducing the overall routing cost. The first set of precedence constraints is associated with the order in which nodes associated with a given request are visited. These constraints do not incur a direct precedence between the last visited pickup and delivery nodes. It is rather required for a vehicle fulfilling a given request to visit all its pickup locations before reaching the corresponding delivery node. The second set of precedence constraints is related to the departure from the depot: upon departure from the depot, the vehicle must perform a pickup. Finally the last set of precedence constraints concerns the end depot, where all requests assigned to the vehicle tour have to be performed before reaching the end depot node.

The MPDTW problem discussed in this paper shares some characteristics with problems previously studied in the literature, namely the pickup and delivery problem with time windows (PDPTW) and the sequential ordering problem (SOP). These are briefly reviewed next.

[30] review the main VRP problem families including VRP with time windows (VRPTW), pickup and delivery problems for goods or people transportation, stochastic VRPs, VRPs with profits and dynamic VRPs. [18] presents the main exact and heuristic algorithms developed for solving VRPs. Exact algorithms include branch-and-bound, dynamic programming, vehicle or commodity flow formulations and their respective solution algorithms, set partitioning formulations and algorithms. The classical heuristics are savings algorithms inspired from [4], set partitioning, cluster-first route-second heuristics in addition to local search based heuristics and improved heuristics used to post optimize VRP solutions. In [27], a tabu search based method is used for solving capacitated VRPs with delivery time

windows and heterogeneous fleet. [6] introduced a parallel iterated tabu search heuristic for solving the classical VRP, the periodic VRP, the multi-depot VRP and the site-dependant VRP. In order to solve the split-delivery vehicle routing problem with time windows, [7] proposes a new exact branch-and-price-and-cut method. [25] proposed a generic adaptive large neighborhood search (ALNS) algorithm for several classes of VRPs: capacitated, multi-depot with time windows, site-dependant and its open versions, in addition to the VRPTW. Finally, [24] presented a solution approach for the PDPTW, which is the closest formulation to the MPDTW investigated in this paper. A number of real-life applications of VRPs can be found in [5].

A similar problem is the special ordering set (SOP), which consists of building a Hamiltonian path in order to solve the asymmetric traveling salesman problem (ATSP) with precedence constraints: the visit of a given node has to be done after visiting a required set of direct and/or indirect predecessors. The SOP differs from the pickup and delivery problem as a node can have multiple direct predecessors [1, 15]. Moreover, a single node can be the direct predecessor of several other nodes, resulting into a tree-like route structure. This problem was introduced in [9] to design heuristics for production planning systems. It has been extended into the constrained SOP (CSOP) [10] to include additional precedence relationships between nodes, such as time windows, where a release date and a deadline are associated with each visited node.

The SOP is used to model real-world problems within production planning in flexible manufacturing systems and for vehicle routing and transportation problems [11]. It has been applied to helicopter routing, job sequencing in flexible manufacturing, stacker cranes in automatic storage systems [2], single vehicle routing with pickup and delivery constraints [12, 26], multicommodity one-to-one pickup and delivery TSP problems [13, 17] and dial-a-ride problems in which items or people are picked up at some points and delivered to others [3]. According to [8] a variety of techniques based on restrictions, e.g., precedence constraints, are used in order to reduce the network size. Several approaches have been adopted to solve the SOP. [26] developed local search algorithms based on the k-exchange concept. [3] used time separation algorithms for solving problems arising in both scheduling and delivery routing problems. [2] used an integer program solved by a branch-and-cut. [15] proposed a sequential solution approach through a parallel version of the heuristic rollout algorithm, while [28] adopted a hybrid genetic algorithm. [1] used a lagrangian relaxation-based scheme for obtaining lower bounds on the optimal solution. Finally, [20] provided a multi-commodity flow formulation for the SOP and the CSOP.

While the MPDTW is associated with many model characteristics of the CVRP, the VRPTW, the PDPTW and the SOP, to our knowledge, this problem has not received any attention in the literature. This paper introduces a problem formulation of MPDTW taking into account the additional multi-pickup characteristics. Moreover, each vehicle route in a MPDTW can be interpreted as an ATSP, in which the distance between two nodes is different depending on the sequence in which the nodes are visited. For example given two locations l_1 and l_2, the travel time from l_1 to l_2 may be different from the travel time from l_2 to l_1. Travel times can differ because of road length, traffic congestion or driving speed limits. Given the precedence constraints on request nodes and on vehicle start and end depots, the vehicle routes are similar to a constrained ATSP. Moreover, we design and implement an ALNS algorithm tailored for the MPDTW. We propose new insertion operators handling the multi-pickup characteristics of the requests defined within

the problem. These operators are also new in the literature and can help solve other types of similar problems.

The remainder of this paper is organized as follows. Section 2 describes MPDTW and introduces a formulation inspired from the PDTW and the SOP. The general ALNS solution method is presented in Section 3 while Section 4 presents the new request insertion procedure. Computational results are reported in Section 5 and Section 6 concludes the paper with main findings and future research avenues.

2 Problem description

A problem instance of the MPDTW contains n requests and m vehicles. Let $P = \{1, ..., p\}$ be the set of pickup nodes, and $D = \{p+1, ..., p+n\}$ be the set of delivery nodes where $|D| = n$ and $p \geq n$. Let $R = \{r_1, ..., r_n\}$ be the set of requests to be routed. Each request $r \in R$ is represented by a set of pickup nodes $P_r \subseteq P$ and a delivery node $d_r \in D$. Let $N = P \cup D$ be the set of customer nodes. Let $r(i)$ be the request associated with node $i \in N$. Let $K = \{1, \ldots, m\}$ be the set of available vehicles.

The graph $G = (V, A)$ consists of the nodes $V = N \cup \{0, p+n+1\}$ where 0 and $p+n+1$ are the starting and ending depot. Each node $i \in V$ has a service time s_i and a time window $[a_i, b_i]$. Given the time window, a vehicle can visit node i if it arrives before the start time a_i, or within the time window such that the service of i is finished before b_i. A vehicle has to wait if its arrival time at i is earlier than the time window start a_i.

The set of arcs $A = V \times V$ minus arcs that lead to infeasible solutions: we omit arc (i, j) if i is a pickup node and j is its delivery node if $b_j > a_i + s_i + t_{ij}$. A distance $d_{ij} \geq 0$ and a travel time $t_{ij} \geq 0$ are associated with each arc $(i, j) \in A$. Let $A^+(i)$ and $A^-(i)$ be the sets of incoming and outgoing arcs from node $i \in V$.

Figure 1 illustrates two requests $R1$ and $R6$ and a route associated with vehicle k. For example in $R6$ two pickups nodes p_3 and p_4 have to be visited to collect items to be delivered to d_6. $R1$ and $R6$ are inserted in route k where precedence constraints are respected. Note that node p_1 is not directly visited after p_2 from the same request. Moreover, the delivery node of request R_1 is visited after all items have been collected from p_1 and p_2. The same applies for R_6.

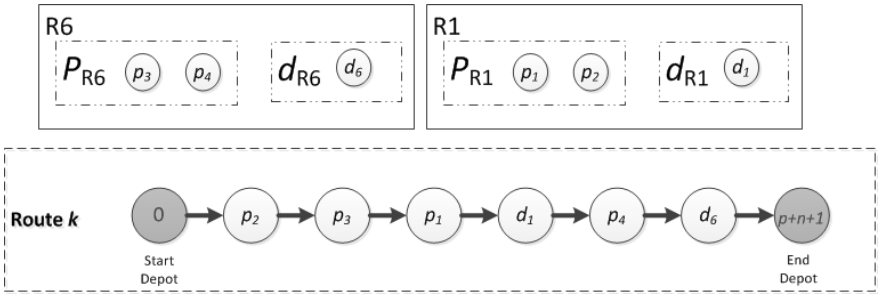

Figure 1: Requests R1 and R6 inserted into route k

A solution to the MPDTW is a set of feasible routes obtained by assigning all requests to the vehicles of the fleet. As the MPDTW is an extension of PDTW and SOP, it is thus an NP-Hard problem. The formulation described next is inspired from the PDTW

and the SOP. The problem can be mathematically formulated with the following decision variables:

- x_{ij}^k is 1 if arc (i, j) is traversed by route k, 0 otherwise;
- y_{rk} is 1 if request r is visited by vehicle k, 0 otherwise;
- S_i indicates the beginning of service at node $i \in V$.

Then, the MPDTW can be formulated as follows:

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ij}^k \tag{1}$$

$$\text{s.t.} \sum_{j \in A^+(i)} x_{ij}^k = y_{r(i)k} \qquad k \in K, i \in N \tag{2}$$

$$\sum_{j \in A^-(i)} x_{ji}^k = y_{r(i)k} \qquad k \in K, i \in N \tag{3}$$

$$\sum_{j \in A^+(0)} x_{0j}^k = 1 \qquad k \in K \tag{4}$$

$$\sum_{k \in K} y_{rk} = 1 \qquad r \in R \tag{5}$$

$$a_i \leq S_i \leq b_i \qquad i \in V \tag{6}$$

$$S_j \geq S_i + s_i + t_{ij} - M(1 - \sum_{k \in K} x_{ij}^k) \qquad (i,j) \in A \tag{7}$$

$$S_{d_r} \geq S_i + s_i + t_{id_r} \qquad i \in P_r, r \in R \tag{8}$$

$$x_{ij}^k \in \{0, 1\} \qquad (i,j) \in A, k \in K \tag{9}$$

$$y_{rk} \in \{0, 1\} \qquad r \in R, k \in K. \tag{10}$$

The objective function (1) is to minimize the overall transportation cost. Constraints (2) and (3) are degree constraints. They also ensure that all nodes of a request belong to the same vehicle. Constraints (4) ensure that all vehicles are used in the solution. Constraints (5) force a request to be served by only one vehicle. Constraints (6) and (7) guarantee schedule feasibility with respect to time windows. The precedence order is preserved via constraints (8). Constraints (9) and (10) impose the nature and the domain of the variables. M is usually a big enough number and can be set to the maximal return time at the depot b_{p+n+1}.

3 Adaptive large neighborhood search heuristic for MPDTW

The ALNS heuristic is an extension of the large neighbordhood search (LNS) introduced by [29]. The ALNS framework presented in [24] and [25] was applied to five variants of VRPs, namely the VRPTW, the CVRP, the multi-depot VRP, the site-dependant VRP and the open VRP. The ALNS has been recently applied to a wide variety of VRPs. For instance

it was used to solve the two-echelon VRP (2E-VRP) [14, 16], two 2-echelon multi-trip VRO with satellite synchronization [14], the location routing problem, which is modeled as a 2E-VRP [16], VRPs with stochastic demands and weight-related costs [22], VRPs arising in an integrated single-vendor multi-buyer inventory-transportation synchronized supply chain [19], and a real-life multi-depot multi-period VRP with a heterogeneous fleet [23]. Due to the use of ALNS in particular for solving PDPTW [23], we anticipate the efficiency of an approach exploiting it to solve the MPDTW. We adapt ALNS for solving the MPDTW as follows.

This section presents our ALNS heuristic to solve the MPDTW. Its pseudocode is presented in Algorithm 1.

Algorithm 1 Adaptive large neighborhood search

Require: S: initial solution

Require: RO: set of removal operators

Require: IO: set of insertion operators

1: $S_{best} \leftarrow S$
2: **while** stop criterion not met **do**
3: $S' \leftarrow S$
4: $q \leftarrow$ Generate a random number of requests to remove
5: $ro \leftarrow$ Select at random an operator from RO
6: $io \leftarrow$ Select at random an operator from IO
7: Remove q requests from S' by applying ro
8: Insert removed requests into S' by applying io
9: **if** $f(S') < f(S_{best})$ **then**
10: $S_{best} \leftarrow S'$
11: **end if**
12: **if** $acccept(S', S)$ **then**
13: $S \leftarrow S'$
14: **end if**
15: **end while**

The initial solution S is generated through a simple construction heuristic in which requests are progressively inserted within an available vehicle, at its minimum insertion cost position. In line 1, the best solution is initialized with the initial solution. Then, the algorithm iterates from lines 2 to 14 until a stop criterion is met, namely a number of iterations that is set as an input.

Each iteration starts by creating a temporary solution S' from solution S. Then a random number q of requests to remove is drawn between $min\{0.1\alpha, 30\}$ and $min\{0.4\alpha, 60\}$ (line 4) in line with to the specifications of the ALNS framework detailed in [24], while we set α as the number of customer nodes already assigned to a route.

Request removal and insertion operators ro and io are then randomly selected from the sets RO and IO in lines 5 and 6, through two distinct roulette wheel selection [24]. The selection of an operator depends on its score: the score of an operator is incremented each time it is used and it improves the solution, to increase the probability for selecting it in the next iterations. In line with [24] and [25] we consider two removal operators within the set RO, namely the *Shaw removal* and the *random removal* heuristics, while we allow IO to include a basic greedy heuristic and regret heuristics with different regret degrees. In particular, we consider four regret heuristics as follows:

- The regret degree is set to three and no insertion noise is applied,
- The regret degree is set to the number of available vehicles m and no insertion noise is applied,
- The regret degree is set to three and the use of the insertion noise is allowed,
- The regret degree is set to the number of available vehicles m and the use of the insertion noise is allowed.

The insertion noise value is used in Algorithm 3 and is discussed in details in Section 4.

Solution S' is destroyed by removing q requests by applying ro (line 7). The solution is then repaired by the application of io (line 8). The condition in line 9 states that if the objective function is improved in S', in comparison with S_{best}, it is set as the current best solution. At the end of the iteration, if a simulated annealing acceptance criterion is verified (line 12), the solution S is set to S'. The acceptance criterion is such as that a candidate solution S' is accepted given the current solution S with a probability $e^{-(f(S')-f(S))/T}$, where T is the temperature that decreases at each iteration according to a standard exponential cooling rate [24].

The ALNS heuristic presented in Algorithm 1 describes a general approach to explore the neighborhoods of an MPDTW solution through removing and inserting requests until an stop criterion is verified. Each insertion operator (i.e., greedy or regret) handles multiple nodes from each request r while looking for best insertion positions within a route. For example, in Figure 2, the insertion of request $R3$ requires inserting the nodes p_5, p_6, p_7 and d_3 within route k previously presented in Figure 1. The resulting route will be explained later.

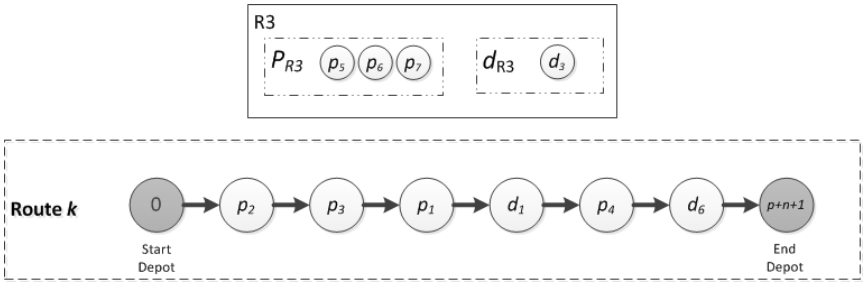

Figure 2: Request R3 to insert into route k

The insertion of a request within a route requires a procedure to insert the set of different pickup and delivery nodes. In addition to the insertion of a delivery node d_r at its best position, the algorithm has to precisely determine the insertion position of each node $p \in P_r$. As the request insertion procedure handles multiple pickup nodes previously to the delivery node, it needs to specify an order in which the pickup nodes are processed by the algorithm. To this end, we define different pickup selection methods namely: *cheapest first*, *most expensive first*, *simple* and *random*. The general procedure for inserting a request within a route, in addition to each of the pickup selection methods are described in Section 4.

4 Request insertion procedure

The general procedure for inserting a request r in a route k is described in Algorithm 2. This general procedure requires subroutines described in Sections 4.1 and 4.2.

Algorithm 2 Insert a request in a vehicle route

Require: r : Request to insert

Require: k: Current route

1: $pickups \leftarrow P_r$
2: **while** $pickups <> \emptyset$ **do**
3: $p_i \leftarrow selectAPickup(pickups, method)$
4: $BestPosition(p_i, k) \leftarrow$ Insert p_i at its best insertion position in k
5: $pickups \leftarrow pickups \backslash \{p\}$
6: **if** $BestPosition(p_i, k) = null$ **then**
7: **return** Request insertion infeasible
8: **end if**
9: **end while**
10: $BestPosition(d_r, k) \leftarrow$ Insert d_r at its best insertion position in k
11: **if** $BestPosition(p_i, k) = null$ **then**
12: **return** Request insertion infeasible
13: **end if**

In line 1 of Algorithm 2, *pickups* initially contains all the pickup nodes P_r. While there are nodes in *pickups*, the algorithm selects in line 3 a pickup node $p_i \in P_r$ through one of the pickup selection methods described in details in Section 4.2. In line 4, the best insertion position of p_i in k ($BestPosition(p_i,k)$) is found by applying Algorithm 3, and then it is removed from *pickups* in line 5. Once all the pickups are inserted, d_r is inserted in line 10 at its best insertion position in k. If the insertion of a node $i \in \{P_r \cup \{d_r\}\}$ is not possible, the algorithm is interrupted and exited according to lines 6–8 and 11–13, where the value $BestPosition(i, k)$ is set to *null*.

Algorithm 3 Insert i at its best position in k

Require: i : Node to insert

Require: k: Current route

$\Delta_i^{k*} \leftarrow \infty$

$BestPosition(i,k) \leftarrow null$

$prev \leftarrow$ the depot node 0

$next \leftarrow$ first customer of route k

while $prev <> p+n+1$ **do**

  $t \leftarrow vehicleArrivalTimeAt(k,i)$

  if $t > b_i$ **then**

    Exit Algorithm 3

  end if

  Set t_{next} the actual arrival time at $next$

  $t'_{next} \leftarrow$ arrival time at $next$ if i is inserted before

  $addedDuration \leftarrow t'_{next} - t_{next}$

  if $t_{next} > b_{next}$ OR $addedDuration > slack_{next}$ **then**

    $prev \leftarrow next$

    $next \leftarrow$ next customer after $next$

    Go to While

  end if

  $NewCost \leftarrow C_{prev,i} + C_{i,next} - C_{prev,next} + generateRandomNoise()$

  if $NewCost < \Delta_i^{k*}$ **then**

    Insert i after $prev$ in current solution to obtain S'

    if S' is feasible **then**

      $BestPosition(i,k) \leftarrow prev$

      $\Delta_i^{k*} \leftarrow NewCost$

    end if

  end if

  $prev \leftarrow next$

  $next \leftarrow$ next customer after $next$

end while

return $BestPosition(i,k)$

4.1 Node insertion at its best position in route

In order to determine the best insertion position of node i in route k, Algorithm 3 iterates through all the nodes already inserted in the route starting from the depot node 0 until its last customer location. At each step, the algorithm temporarily inserts i after a node j from k, computing the increase in routing costs, Δ_i^k. Assume that Algorithm 3 is processing the pickup node p_6 from request R_3, given the nodes already inserted in route k and illustrated in Figure 2. The algorithm would test the insertion of p_6 after 0, p_2, p_3, p_1, d_1, p_4 and d_6.

In Algorithm 3, the best insertion cost of i in k (Δ_i^{k*}) is initially set to an infinite value in line 1, while the value of the best insertion position of node i is initially set to *null* in line 2. At the first iteration of the algorithm, node i is inserted between the depot node 0 and the first customer of route k. To this end, the previous node *prev* of i is set to the depot node at line 3, whereas the next node *next* to be visited after i is set in line 4. Line 6 computes the arrival time of k at i, which is used in lines 7–9 to verify that the vehicle would visit i before b_i given the current insertion. In line 10, t_{next} is set to the vehicle arrival time at *next* assuming that k is not yet including i. Then, in line 11 the vehicle arrival time at *next*, namely t'_{next} is computed, assuming that i is inserted in k. The added duration computed in line 12 is used in lines 13–17 to verify that if node i is visited, the vehicle would visit *next* before the its operation end time b_{next}. Also, the added duration is compared with the slack at *next*, that indicates to which extent the vehicle's visit to *next* would be postponed. If one of these conditions is violated, the algorithm moves to the next iteration, to test the next insertion position in line 16.

If all the conditions are verified, the insertion cost of i in k at the position under evaluation ($NewCost$) is computed in line 18 to which a random noise is added. If the application of a noise is allowed, $generateRandomNoise()$ generates a random value in $[-addedNoise, +addedNoise]$, where $addedNoise$ is a value that is proportional to the maximum value in the problem's distance matrix. In line 19, the temporary $NewCost$ is compared against Δ_i^{k*} to determine whether the insertion of i in k improves the solution. If the condition is verified, the feasibility of the new solution is tested by first inserting i in k to obtain S' in line 20. If S' generates a finite cost, the feasibility of the new solution is verified (line 21) and the best known insertion position of i in k is set to *prev* in line 22. Then, Δ_i^{k*} is set to the value of $NewCost$ in line 23.

The next insertion position is updated in lines 26 and 27. After iterating through all customer nodes in k, the algorithm returns the best insertion position of i in k in line 29.

4.2 Pickup selection

Here we present the *simple* and *random* pickup selection methods. We also describe in details the algorithms of our *cheapest first* and *most expensive first* pickup selection methods. The *simple* pickup selection method selects the pickup nodes according to the order in which it is saved in the computer's memory or within the datafile describing the problem. The example depicted in Figure 3 illustrates how the pickups from the request R_3 are inserted within the route k with the *simple* method. The algorithm starts by inserting P_5 at its best insertion position, then P_6 and P_7. In the *random* method the index of a pickup node from *pickups* is randomly generated in $[1..|pickups|]$. In the

example illustrated in Figure 4, the *random* method results into inserting P_7, P_5 then P_7 at their respective best insertion positions.

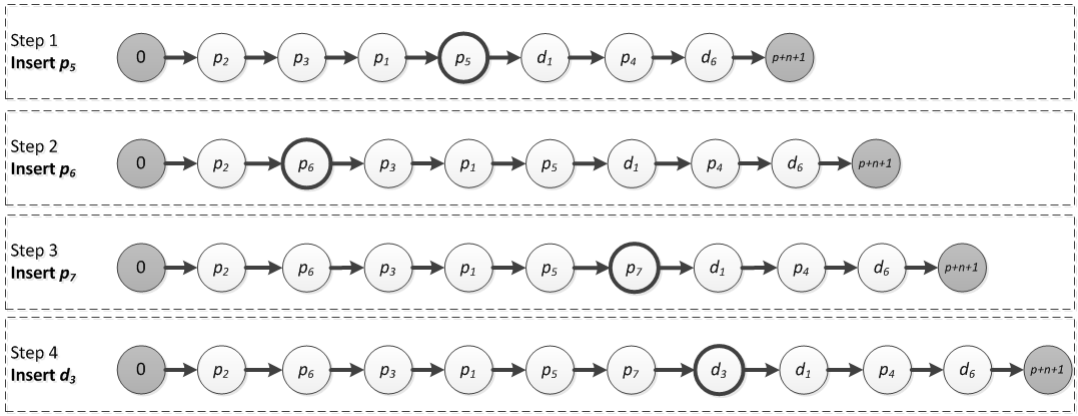

Figure 3: Insertion of R3 in route k with the simple method

Algorithm 4 contains the pseudocode of the *cheapest first* method. The algorithm is initiated by setting the best insertion cost Δ to an infinite value in line 1. Then, Δ_p^{k*} is computed for each $p \in pickups$ and the algorithm returns the index of the node that has the lowest best insertion cost. In Figure 5, at the first iteration node P_6 is inserted from R_3's pickup nodes, as it results into ${\Delta^8_{P_6}}^* = 501$ versus ${\Delta^8_{P_5}}^* = 1027$ and ${\Delta^8_{P_5}}^* = 728$. In the second iteration of the algorithm, P_7 is inserted as ${\Delta^8_{P_7}}^* = 728$ is superior to the ${\Delta^8_{P_5}}^* = 1125$, both updated at iteration 2 given the insertion of p_6 in route k . Finally P_5 is inserted at its best insertion position at Step 3.

Algorithm 5 contains the pseudocode of the *most expensive first* method, which is similar to Algorithm 4.

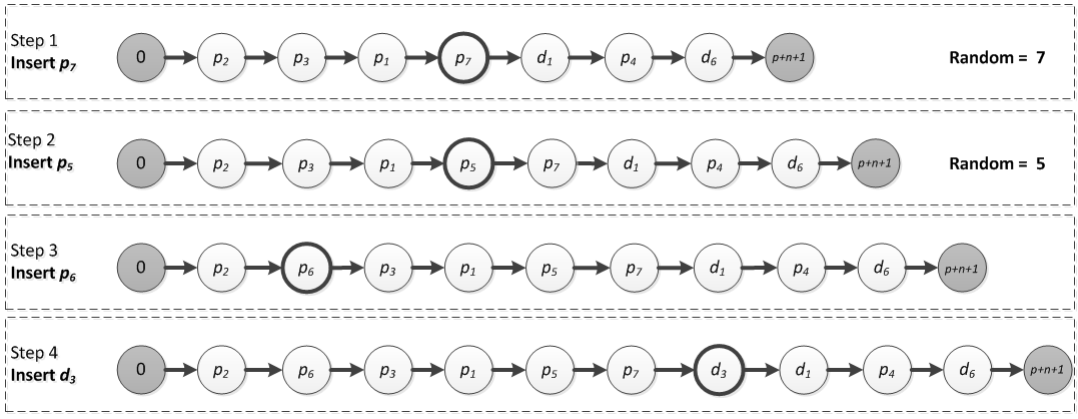

Figure 4: Insertion of R3 in route k with the random method

Algorithm 4 *selectAPickup*(*pickups*, cheapest first)

$\Delta \leftarrow \infty$
for all $p \in pickups$ **do**
  $\Delta_p^{k*} \leftarrow$ Compute the best insertion cost of p in k
  if $\Delta_p^{k*} < \Delta$ **then**
    $\Delta \leftarrow \Delta_p^{k*}$
    $i \leftarrow p$
  end if
end for
return i

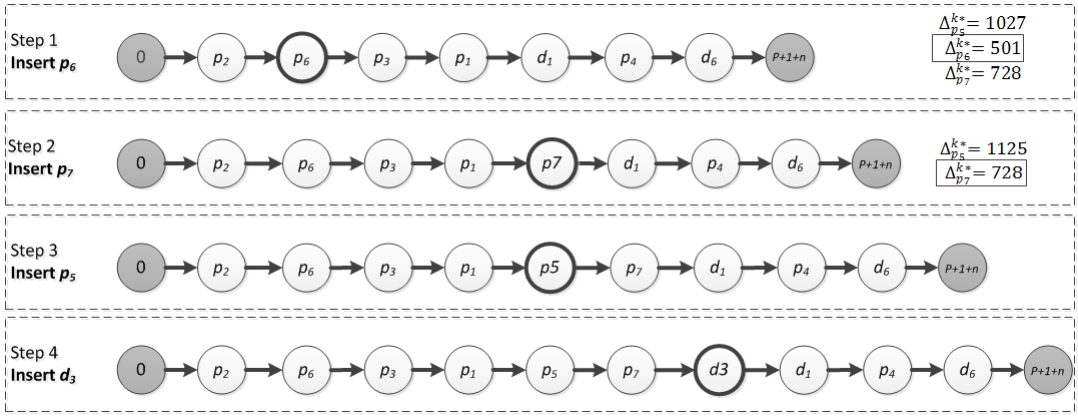

Figure 5: Insertion of R3 in route k with the cheapest first method

Algorithm 5 *selectAPickup*(*pickups*, most expensive first)

$\Delta \leftarrow 0$
for all $p \in pickups$ **do**
  $\Delta_p^{k*} \leftarrow$ Compute the best insertion cost of p in k
  if $\Delta_p^{k*} > \Delta$ **then**
    $\Delta \leftarrow \Delta_p^{k*}$
    $i \leftarrow p$
  end if
end for
return i

5 Computational experiments

In this section we describe our computational experiments. Section 5.1 introduces the characteristics of the MPDTW problem test instances and the different ALNS parameter values. The results of detailed and extensive computational experiments are then presented in Section 5.2.

5.1 Test instances and ALNS parameters

The test instances were built upon existing PDTW instances from [21]. For each PDTW instance, pickup and delivery nodes are first separated into distinct lists. Then, two main instances are created having respectively short (maximum of 4 nodes) and long (maximum of 8 nodes) requests. These are created and categorized as follows.

The developed heuristic is tested against a set of instances classified according to the characteristics presented in Table 1. Each instance type is characterized by a time window type, the maximum length of the requests and the number of nodes (instance size). An instance is defined as *Without*, with *Normal* or with *Large* TWs. For each instance, the minimum size of a request is equal to two, as it includes one delivery point. Depending on the instance type, it can include at most 4 (*Short* requests) or 8 (*Long* requests) pick-up and delivery nodes. Finally, our instances contain 50, 100 or 400 nodes. For each of the 18 instance types, we have generated five instances, resulting into a total of 90 instances in our test bed. For example, the instances of type L_4_50 represent *Large* TW, *short* requests, 50 nodes and are denoted L_4_50_1 to L_4_50_5.

	Without TW		Normal TW		Large TW	
Instance Size	Short Requests	Long Requests	Short Requests	Long Requests	Short Requests	Long Requests
50	W_4_50	W_8_50	N_4_50	N_8_50	L_4_50	L_8_50
100	W_4_100	W_8_100	N_4_100	N_8_100	L_4_100	L_8_100
400	W_4_400	W_8_400	N_4_400	N_8_400	L_4_400	L_8_400

Table 1: Instance types

According to the type of requests within an instance, the length k of a request is randomly generated between 2 and 4 for short requests and between 2 and 8 for long requests. $k-1$ nodes are randomly drawn from the list of pickup nodes, while a delivery node is randomly selected from the delivery node list.

Then for each window type (*Without* TW, *Normal* TW and *Long* TW) and for each instance size (50, 100 or 400 nodes), a request is randomly selected. The TW of the nodes within the request is modified according to the window type:

- The TW is deleted, if the nodes in the instance to generate are *Without* TW,
- The TW is kept the same as in the original PDTW instance, to obtain an instance with *Normal* TW,
- The TW is increased by 300 for an instance with *Large* TW.

The feasibility of the request is tested by inserting it in a route through the cheapest first insertion method. The request is rejected if it is not feasible. Request generation for

a given instance is repeated until reaching the desired instance size. The procedure just described above is executed five times to obtain the 90 test instances.

In order to assess the performance of our ALNS heuristic for solving the MPDTW we use the following parameters for the algorithm. Each test presented in Table 2 defines an initial temperature, the number of iterations required to fulfil the algorithm's stopping criterion in addition to the pickup selection method.

Test type (T)	Test #	Initial temperature	Iteration #	Pickup selection method
1	1	0	25000	Cheapest first
	2			Most expensive first
	3			Random
	4			Simple
2	5	3000	25000	Cheapest first
	6			Most expensive first
	7			Random
	8			Simple
3	9	3000	100000	Cheapest first
	10			Most expensive first
	11			Random
	12			Simple

Table 2: ALNS test parameters

5.2 Computational results

The experiments reported here have been performed on a desktop computer equipped with a 2.67 GHz Intel processor operating under the Scientific Linux 6.3. For each test reported in Table 2 ($t = 1$ to 12), the algorithm is executed three times for each test instance. The aim of the replication of the algorithm is to account for the variability induced by the randomness at several stages of the heuristic, such as the noise included in the computation of a node's insertion cost within a route or the selection of an operator. The average values of the solution cost and time for each method, each instance and each test type T=1 to 3 are computed and used to assess the performance of each method.

The statistics reported in Tables 3 and 6 are used to evaluate the performance of our ALNS heuristic for solving the MPDTW, while exploiting each of the pickup selection methods discussed in Section 3.

In order to provide easily interpretable performance indicators, the average deviation from the best known solution is reported in Table 3 as *Avg Cost* for each test (from 1 to 12). The results are aggregated according to the instance TW type, that generate indicators highly representative of the results, i.e., the average deviation of costs obtained by (t, i) in a test T are close to the aggregated average cost obtained by t for all the instances having the same window type as i. The *Freq* indicator reported in Table 3 is computed for each test and each instance group as the percentage of instances for which the method obtained a solution with the lowest cost. Finally *Avg Sol Time* is the average solution time in seconds spent by a test t for solving instances with a given TW type.

According to the *Freq* indicator in Table 3, we observe that as the ALNS temperature and iteration count are increased, the solution methods tend to generate better solutions more often. Besides, the *Cheapest first* and *Random* insertion sequence methods obtain solutions with the lowest overall costs, while a poor performance is obtained by *Most*

Test # (t)	Pickup Selection method	Without TW			Normal TW			Large TW		
		Avg. Cost (%)	Freq. (%)	Avg. Sol. Time (s)	Avg. Cost (%)	Freq. (%)	Avg. Sol. Time (s)	Avg. Cost (%)	Freq. (%)	Avg. Sol. Time (s)
1		2.0	20.0	753.7	0.4	20.0	632.9	0.5	30.0	673.2
5	Cheapest	0.8	46.7	768.7	0.3	23.3	642.8	0.4	43.3	690.2
9	first	0.8	43.3	3229.2	0.2	43.3	2693	0.3	50.0	2915.4
Average		1.2	36.7	1583.9	0.3	28.9	1322.9	0.4	41.1	1426.3
2	Most	1.7	20.0	687.6	6.7	40.0	600.7	59.2	33.3	625.2
6	expensive	1.4	20.0	757.2	6.7	50.0	634.2	59.0	30.0	673.1
10	first	1.1	33.3	3106.3	6.6	36.7	2876.8	59.0	30.0	2805.4
Average		1.4	24.4	1517	6.7	42.2	1370.5	59.1	31.1	1367.9
3		0.9	30.0	734.7	0.2	26.7	618	0.4	26.7	649.2
7	Random	0.6	43.3	734.8	0.1	40.0	615.5	0.3	50.0	652.9
11		0.5	50.0	3377.7	0.0	50.0	2665.2	0.2	50.0	2828.6
Average		0.7	41.1	1615.7	0.1	38.9	1299.5	0.3	42.2	1376.9
4		1.5	33.3	707	37.3	26.7	612.1	52.7	33.3	639.6
8	Simple	1.0	43.3	693.1	37.1	36.7	601.5	52.6	40.0	642.1
12		1.3	43.3	2937.3	37.0	40.0	2393.5	52.4	40.0	2667.1
Average		1.3	40.0	1445.8	37.1	34.4	1202.3	52.5	37.8	1316.3

Table 3: Method performance by type of time windows

expensive first and *Simple* methods.

For instances without TWs we observe that for $T = 1$, the *Random* insertion method performs better than *Cheapest first* in terms of solution costs and best solution frequency (30% in $t = 3$ versus 20% in $t = 1$). Besides, when the initial temperature is set to 3000 the performance of *Cheapest first* improves in tests $t = 5$ and $t = 9$, as the value of *Avg Cost* decreases below 1%.

For the instances containing normal TWs, we note that regardless of the ALNS initial temperature and iteration count, the *Cheapest first* and *Random* methods obtain the lowest average deviations from best cost (respectively 0.3% and 0.1% on average). Regardless of the parameters, the *Random* method yields the best solution more often for 38.9% of the instances on average versus 28.9% for the *Cheapest first*.

Finally, for instances with large TWs, the results indicate that the *Cheapest first* and *Random* methods perform equally well with averages of 0.4% and 0.3% are respectively obtained for *Avg Cost*. In addition, the *Random* method leads to solutions with the best costs while solving 42.2% of the instances followed closely by the *Cheapest first* method, for which $Freq = 41.1\%$. Finally, in the settings $T = 3$ both methods obtain the best costs for 50% of the instances.

We observe that the *Most expensive first* and the *Simple* methods produce high average values for *Avg Cost* when solving instances with large and normal TWs, besides obtaining results that are comparable to the other methods for instances without them. This is explained by the very high cost of the solutions for some particular instances as indicated in Tables 4 and 5. For instance, for tests 4, 8 and 12 the *Simple* method generated solutions with average costs that deviate 1014.8% from the best solution obtained for the instance L_8_50_4, 549% for L_8_100_4 and around 1107% for N_8_50. Also, for tests 2 , 6 and 10, the *Simple* method generates solutions whose average costs deviate 1014.8% for instance L_8_50_4 , around 549% for L_8_100_4 , around 199% for L_8_400_4 and around 191% for N_8_400_3. Analyses of the algorithm behaviour in these special cases in which the *Most*

expensive first or the *Simple* methods are implemented show that the initial solutions generated by the sequential insertion and used as the input of the ALNS are very costly in comparison with the initial solutions obtained when the *Cheapest first* or the *Random* methods are implemented.

Instance Type	Instance #	Test 4 (%)	Test 8 (%)	Test 12 (%)
L_8_50	4	1014.8	1014.8	1014.8
L_8_100	4	548.7	549.4	549.4
N_8_50	4	1099.4	1106.2	1106.7

Table 4: High deviation from the minimum cost for the *Simple* method

Instance Type	Instance #	Test 2 (%)	Test 6 (%)	Test 10 (%)
L_8_50	4	1014.8	1014.8	1014.8
L_8_100	4	548.7	549.4	549.4
L_8_400	4	199.8	197.9	199.7
N_8_400	3	191.8	191.5	190.4

Table 5: High deviation from the minimum cost for the *Most expensive first* method

According to Table 3 the best solution times are obtained when the *Simple* method is implemented, followed by the *Random*, the *Most expensive* and the *Cheapest first* methods. According to our analyses, the *Random* method achieves the best trade-off between the solution cost and the running time. However, when the quickest *Simple* method is implemented, it may provide poor quality solutions in some special cases. Moreover, with the *Cheapest first* insertion method, solution costs are comparable to the ones generated by the *Random* method, and it requires the highest solution times. Finally the use of the *Most expensive first* method is the less promising option for solving the MPDTW, as it is one of the slowest alternatives and may provide poor quality solutions in some special cases of instances.

Finally, average solution times in seconds per instance type are reported in Table 6. Overall, irrespective of the implemented pickup selection methods, the results are not affected by the initial temperature of the algorithm, as they are similar between test types $T = 1$ and $T = 2$ where the ALNS iteration number is set to 25000 and the initial temperature is increased from 0 to 3000. Besides when the iteration number is increased from 25000 to 100000 in $T = 3$, the solution time is proportionally increased.

6 Conclusion

This paper described a new ALNS implementation applied for the MPDTW problem. Our new operators are specifically designed to tackle this problem, and can help other local search algorithms for similar problems, such as the PDP and the SOP, among others. We have also modeled the problem via an integer programming formulation, and designed a complete benchmark set of instances inspired from the PDTW ones. The reported results show that the computational times of our ALNS implementation are at their lowest while generating solution with lowest costs when the *Random* pickup selection method is implemented, in comparison with the other three methods. This means that selecting the

Instance Type	T=1	T=2	T=3
W_50_4	30.4	30.5	121.8
W_50_8	36.3	36.4	147.2
W_100_4	107.5	105.0	402.9
W_100_8	184.3	184.5	765.7
W_400_4	1288.2	1279.5	4987.3
W_400_8	2757.6	2794.7	12550.9
N_50_4	27.9	28.2	116.1
N_50_8	34.5	34.8	141.3
N_100_4	111.0	110.8	465.2
N_100_8	201.1	204.1	898.1
N_400_4	1246.4	1274.0	5298.6
N_400_8	2118.8	2089.0	9023.3
L_50_4	30.0	30.1	121.0
L_50_8	35.5	35.7	144.2
L_100_4	112.1	112.0	447.6
L_100_8	209.8	212.5	903.2
L_400_4	1277.9	1290.8	5194.4
L_400_8	2278.6	2306.5	10014.4

Table 6: Average solution time per instance type and per test type (seconds)

sequence of nodes to insert in a solution in random order tends to be more beneficial than using other criteria such as lowest or highest costs, or to to use the sequence in which these nodes appear.

We have provided the first solutions for this problem, and we hope that it motivates other researchers in developing competing algorithms and on developing methods to provide lower bounds for this problem. Moreover, this work could be extended by considering an heterogeneous fleet with different capacities or with special loading condition features.

Acknowledgments

This research was partly supported by grants 2014-05764 and 2015-04893 from the Canadian Natural Sciences and Engineering Research Council. This support is gratefully acknowledged.

References

References

[1] A. Alonso-Ayuso, P. Detti, L. F. Escudero, and M. T. Ortuño. On dual based lower bounds for the sequential ordering problem with precedences and due dates. *Annals of Operations Research*, 124(1-4):111–131, 2003.

[2] N. Ascheuer, M. Jünger, and G. Reinelt. A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints. *Computational Optimization and Applications*, 17(1):61–84, 2000.

[3] E. Balas, M. Fischetti, and W. R. Pulleyblank. The precedence-constrained asymmetric traveling salesman polytope. *Mathematical Programming*, 68(1-3):241–265, 1995.

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

[5] J. Coelho, L.C. J. Renaud and G. Laporte. Road-based goods transportation: a survey of real-world logistics applications from 2000 to 2015. *INFOR: Information Systems and Operational Research*, 54(2):79–96, 2016.

[6] J.-F. Cordeau and M. Maischberger. A parallel iterated tabu search heuristic for vehicle routing problems. *Computers & Operations Research*, 39(9):2033–2050, 2012.

[7] G. Desaulniers. Branch-and-price-and-cut for the split-delivery vehicle routing problem with time windows. *Operations Research*, 58(1):179–192, 2010.

[8] G. Desaulniers, J. Desrosiers, A. Erdmann, M. M. Solomon, and F. Soumis. VRP with pickup and delivery. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, pages 225–242. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[9] L. F. Escudero. An inexact algorithm for the sequential ordering problem. *European Journal of Operational Research*, 37(2):236–249, 1988.

[10] L. F. Escudero and A. Sciomachen. Local search procedures for improving feasible solutions to the sequential ordering problem. *Annals of Operations Research*, 43(7):397–416, 1993.

[11] A. Ezzat, A. M. Abdelbar, and D. C. Wunsch. An extended eigenant colony system applied to the sequential ordering problem. In *Swarm Intelligence (SIS), 2014 IEEE Symposium on*, pages 1–7. IEEE, 2014.

[12] M. T. Fiala Timlin and W. R. Pulleyblank. Precedence constrained routing and helicopter scheduling: heuristic design. *Interfaces*, 22(3):100–111, 1992.

[13] L. Gouveia and M. Ruthmair. Load-dependent and precedence-based models for pickup and delivery problems. *Computers & Operations Research*, 63:56–71, 2015.

[14] P. Grangier, M. Gendreau, F. Lehuédé, and L.-M. Rousseau. An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. *European Journal of Operational Research*, 254(1):80–91, 2016.

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

[16] V. C. Hemmelmayr, J.-F. Cordeau, and T. G. Crainic. An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. *Computers & Operations Research*, 39(12):3215–3228, 2012.

[17] H. Hernández-Pérez and J.-J. Salazar-González. The multi-commodity one-to-one pickup-and-delivery traveling salesman problem. *European Journal of Operational Research*, 196(3):987–995, 2009.

[18] G. Laporte. Fifty years of vehicle routing. *Transportation Science*, 43(4):408–416, 2009.

[19] Y. C. E. Lee, C. K. Chan, A. Langevin, and H. W. J. Lee. Integrated inventory-transportation model by synchronizing delivery and production cycles. *Transportation Research Part E: Logistics and Transportation Review*, 91:68–89, 2016.

[20] A. N. Letchford and J.-J. Salazar-González. Stronger multi-commodity flow formulations of the (capacitated) sequential ordering problem. *European Journal of Operational Research*, 251(1):74–84, 2016.

[21] H. Li and A. A. Lim. A metaheuristic for the pickup and delivery problem with time windows. *International Journal of Artificial Intelligence Tools*, 12(2):160170, 2001.

[22] Z. Luo, H. Qin, D. Zhang, and A. Lim. Adaptive large neighborhood search heuristics for the vehicle routing problem with stochastic demands and weight-related cost. *Transportation Research Part E: Logistics and Transportation Review*, 85:69–89, 2016.

[23] S. Mancini. A real-life multi depot multi period vehicle routing problem with a heterogeneous fleet: Formulation and adaptive large neighborhood search based matheuristic. *Transportation Research Part C: Emerging Technologies*, 70:100–112, 2016.

[24] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, 2007.

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

[26] M. W. P. Savelsbergh. An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operational Research*, 47(1):75–85, 1990.

[27] F. Semet and E. Taillard. Solving real-life vehicle routing problems efficiently using tabu search. *Annals of Operations Research*, 41(4):469–488, 1993.

[28] D.-I. Seo and B.-R. Moon. A hybrid genetic algorithm based on complete graph representation for the sequential ordering problem. In *Genetic and Evolutionary Computation Conference*, pages 669–680, 2003.

[29] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 417–431, 1998.

[30] P. Toth and D. Vigo, editors. *Vehicle Routing*. Monographs on Discrete Mathematics and Applications. MOS-SIAM Series on Optimization, Philadelphia, 2014.