INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 27 (2020) 112–137
DOI: 10.1111/itor.12623

A variable neighborhood search heuristic algorithm for the double vehicle routing problem with multiple stacks

Jonatas B. C. Chagas[a], Ulisses E. F. Silveira[b], André G. Santos[b] and Marcone J. F. Souza[a]

[a] *Departamento de Computação, Universidade Federal de Ouro Preto, Ouro Preto, Brazil*
[b] *Departamento de Informática, Universidade Federal de Viçosa, Viçosa, Brazil*
E-mail: jonatas.chagas@iceb.ufop.br [Chagas]; ulisses.silveira@ufv.br [Silveira]; andre@dpi.ufv.br [Santos]; marcone@iceb.ufop.br [Souza]

Received 9 February 2018; received in revised form 14 December 2018; accepted 14 December 2018

Abstract

This paper addresses the double vehicle routing problem with multiple stacks (DVRPMS) in which a fleet of vehicles must collect items in a pickup region and then travel to a delivery region where all items are delivered. The load compartment of all vehicles is divided into rows (horizontal stacks) of fixed profundity (horizontal heights), and on each row, the unloading process must respect the last-in-first-out policy. The objective of the DVRPMS is to find optimal routes visiting all pickup and delivery points while ensuring the feasibility of the vehicle loading plans. We propose a new integer linear programming formulation, which was useful to find inconsistencies in the results of exact algorithms proposed in the literature, and a variable neighborhood search based algorithm that was able to find solutions with same or higher quality in shorter computational time for most instances when compared to the methods already present in the literature.

Keywords: vehicle routing; pickup and delivery; loading constraints; mathematical formulation; variable neighborhood search

1. Introduction

The double vehicle routing problem with multiple stacks (DVRPMS) arose in its simplest form as the double traveling salesman problem with multiple stacks (DTSPMS), proposed by Petersen and Madsen (2009), when a software company that set up routes in its intermodal traffic encountered this problem with one of its customers.

In the DTSPMS, a single vehicle, which has its load compartment (container) divided into rows (horizontal stacks) of fixed depth (horizontal heights), must collect all items spread in a region known as pickup region and, henceforth, deliver all these collected items in another region, denominated delivery region. All items have the same size and shape. The items are stored in the

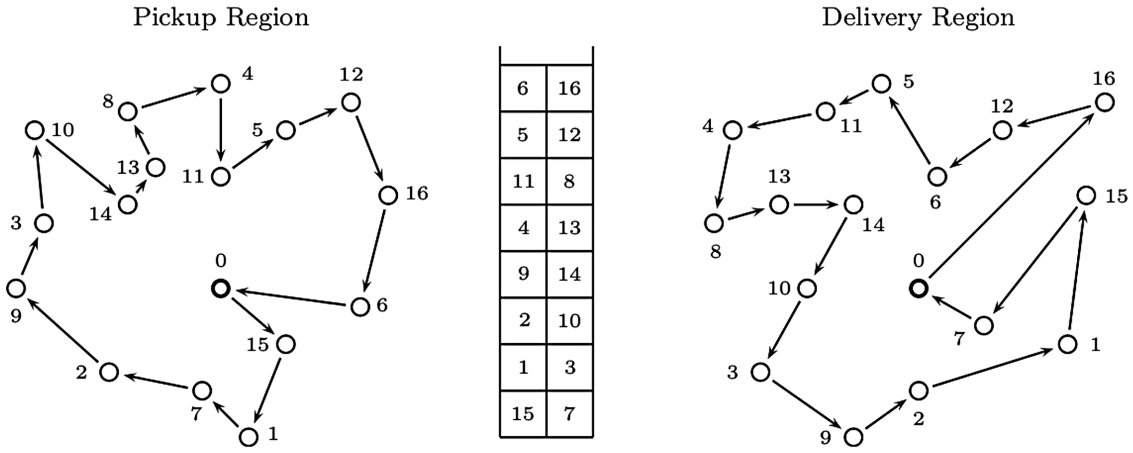

Fig. 1. DTSPMS example—adapted from Iori and Riera-Ledesma (2015).

stacks as they are collected. It is important to state that the items are not stored on top of each other, they are arranged in the same plane (container base) according to rows (horizontal stacks) and their depths (horizontal heights). The items' positions cannot be changed when they are already inside the container, that is, the items should be stationary until their unloading. The delivery must then respect the last-in-first-out (LIFO) policy. Thus, the delivery route is limited by the stack configurations set up by the pickup route.

The DTSPMS is a variation of the pickup and delivery traveling salesman problem (TSP) with multiple stacks (Cordeau et al., 2010; Côté et al., 2012; Sampaio and Urrutia, 2016), where the pickup and delivery operations must be completely separate. This is due to the fact that the DTSPMS arises in the context where the pickup and delivery regions are widely separated. In this way, all items in the pickup region must be gathered prior to any unloading in the delivery region. The transportation cost between the two regions is fixed and it is not considered as part of the optimization problem. The problem has applicability in deliveries where items are loaded and unloaded from the rear of the vehicle and item reallocation is prohibited due to the fact that the items are heavy and/or fragile or the handling of these items is dangerous.

The objective of the DTSPMS is to find two Hamiltonian cycles, one for the pickup region and the other for the delivery region, so that the sum of the distances traveled in both regions is the minimum possible, respecting the precedence constraints imposed by the vehicle's stacks.

Figure 1 depicts a feasible solution for the DTSPMS in a case involving 16 items. Each item is associated with a pickup client and a delivery client (item 1 is associated with pickup client 1 and with delivery client 1, item 2 is associated with pickup client 2 and with delivery client 2, and so on). The container of this vehicle is divided into two stacks of height 8, that is, the container has dimensions (2×8). The vehicle always starts its route in the pickup region at the vertex 0 (depot) and, in this example, the vehicle visits customer 15, collects and stores the item in the first stack, then visits customer 7, storing the item in the second stack, then customer 1 is visited and has his item stored on the first stack. The gathering continues as shown in the figure until all customers have been served and their items stored in the vehicle container. At the end, the vehicle returns to the depot from where the entire container is transported to the delivery region depot. In the delivery

region, the container is loaded into a vehicle that also always starts its route at the vertex 0 (depot) and, in this example, from the depot, the vehicle has only two possible customers to visit, since only items 6 and 16 are accessible from the top of the two stacks. As illustrated in the figure, the vehicle initially satisfies customer 16, unloads its item from the second stack, and customer 12 becomes available to be served. The process continues as illustrated, always satisfying a customer who has its item on top of any of the stacks. At the end of delivering, the vehicle returns to the depot in its respective region (in this case, the delivery region).

Petersen and Madsen (2009) proposed the DTSPMS, presented a mathematical formulation for the problem, and also proposed four heuristic approaches to solve it. The four heuristic approaches were based on the iterated local search (ILS), tabu search, simulated annealing (SA), and large neighborhood search (LNS) metaheuristics. These heuristics were tested on a set of instances that became the reference for future works. Among the proposed heuristics, the one based on LNS had better overall performance.

Since it was presented, the DTSPMS aroused great interest in the academic community, with several exact and heuristic approaches. Felipe et al. (2009) proposed a heuristic approach based on the variable neighborhood search (VNS) metaheuristic in which six neighborhood structures were used. Computational results showed that the VNS approach overcame the results presented by Petersen and Madsen (2009).

Petersen et al. (2010) proposed different exact mathematical formulations for the DTSPMS, including a branch-and-cut algorithm to solve the DTSPMS instances to optimality. Based on these results, it was determined that the difficulty of a given instance depends not only on the number of items, but also strongly depends on the height of the stacks.

Lusby et al. (2010) presented an exact method based on matching k-best tours for each of the regions separately. This method consists of repeatedly finding solutions for the two separate TSP (delivery region and pickup region) until a feasible loading plan is found. The results showed a significant superiority of this method compared to the previous method proposed by Petersen et al. (2010).

Carrabs et al. (2010) developed a branch-and-bound algorithm for the double traveling salesman problem with two stacks (DTSP2S), a special case of the DTSPMS in which the vehicle has exactly two stacks. The results showed that the branch-and-bound algorithm performed better than the other exact approaches in the literature (Lusby et al., 2010; Petersen et al., 2010) in terms of computational time and the number of global optima. According to Carrabs et al. (2010), for the exact approaches proposed by Lusby et al. (2010) and Petersen et al. (2010), the difficulty of an instance depends on the capacity of the stacks, consequently, it depends on the number of items and on the number of stacks in the container. In this case, the performance of the algorithms improves when the number of stacks increases. This is probably due to the fact that the construction of the routes in the pickup region and delivery region becomes less restricted.

Casazza et al. (2012) studied the theoretical properties of the DTSPMS, analyzing the structure of DTSPMS solutions in two separate components: routes and loading plan. It was shown that some DTSPMS subproblems can be solved in polynomial time, considering specific cases of the problem.

Alba Martínez et al. (2013) proposed improvements to the branch-and-cut algorithm of Petersen et al. (2010), adding new valid inequalities (cut planes) that allowed greater efficiency. The results showed that the new algorithm overcame the exact methods that existed in the literature to that date.

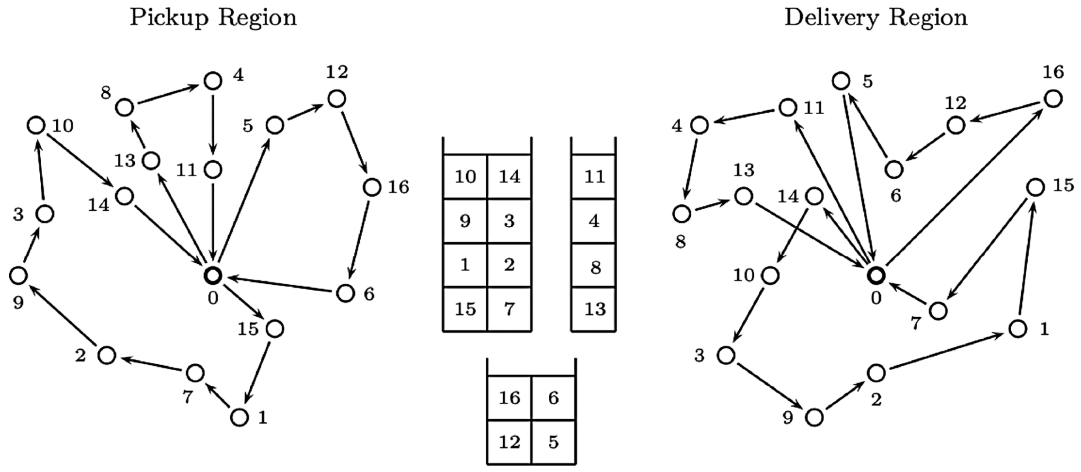

Fig. 2. DVRPMS example (completely filled containers)—adapted from Iori and Riera-Ledesma (2015).

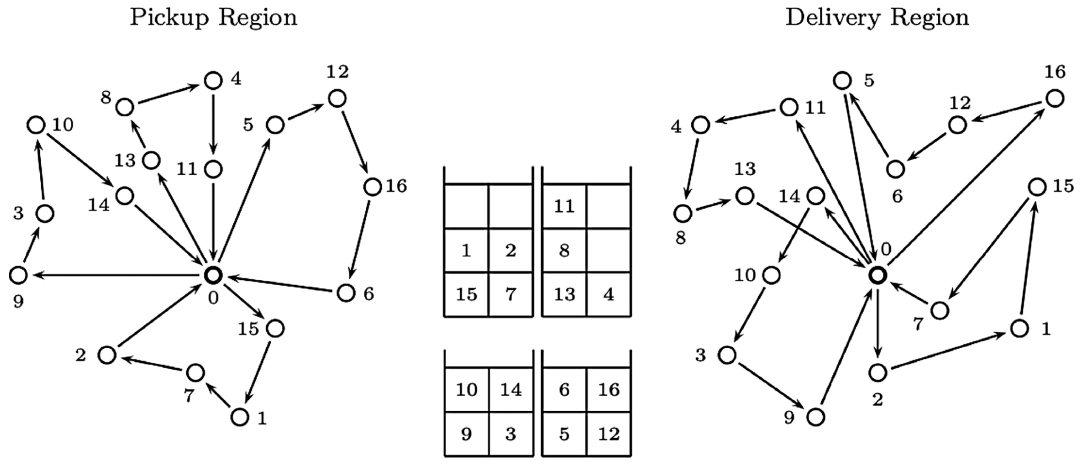

Fig. 3. DVRPMS example (noncompletely filled containers)—adapted from Iori and Riera-Ledesma (2015).

More recently, the exact algorithm proposed by Barbato et al. (2016) was able to solve instances involving containers of two stacks, which were not previously solved in the literature. To the best of our knowledge, it is the last report on the DTSPMS.

Iori and Riera-Ledesma (2015) proposed the DVRPMS, a generalization of the DTSPMS. The DVRPMS has the same characteristics and constraints as the DTSPMS, except that now there is a fleet of vehicles available to meet the demand of the customers. According to the authors, the DVRPMS was motivated by the fact that not always a single vehicle is enough to transport all items. In addition, using multiple vehicles can be interesting even if all products could be transported by a single vehicle, as the addition of more vehicles causes an increase in the flexibility of the loading/unloading process and this can lead to a reduction in the operating costs of transport.

Figures 2 and 3 show two feasible solutions for the same instance involving 16 requests. Figure 2 represents a solution using three heterogeneous vehicles with containers of size (2×4), (1×4), and

(2×2), where all containers are completely filled. Figure 3 represents a solution using four vehicles with containers of (2×4), (2×4), (2×2), and (2×2), but not all containers spaces are filled. In both examples, for each vehicle are associated a route in the pickup region, a route in the delivery region, and a loading plan of the container. The pickup and delivery route for each vehicle respects the LIFO policy on the respective container.

Besides proposing the DVRPMS, Iori and Riera-Ledesma (2015) presented three exact algorithms: branch-and-cut, branch-and-price, and branch-and-cut-and-price. As stated by the authors, the three exact algorithms had different behavior and efficiency with respect to different instances, created to evaluate the quality of their algorithms.

To the best of our knowledge, only two works have addressed the DVRPMS by heuristic algorithms. Silveira et al. (2015) proposed three methods based on ILS, SA, and variable neighborhood descent metaheuristics, while Chagas et al. (2016) proposed another method based on the SA metaheuristic, which outperformed all heuristic algorithms proposed by Silveira et al. (2015) and was able to find several solutions with the same quality as those found by Iori and Riera-Ledesma (2015).

Chagas and Santos (2016) introduced the double vehicle routing problem with multiple stacks and heterogeneous demand (DVRPMSHD), a generalization of the DVRPMS, which occurs when customers have heterogeneous demands and the demand of each customer cannot be divided among two or more vehicles. The authors also proposed a simple branch-and-price algorithm that was able to solve only instances of up to 15 customer requests. Posteriorly, a simple and effective heuristic based on the SA metaheuristic was proposed by the same authors in (Chagas and Santos, 2017). Their algorithm was able to overcome several results found by the branch-and-price algorithm.

In this paper, we address the DVRPMS, propose an integer linear programming (ILP) formulation and a heuristic algorithm based on the VNS metaheuristic to solve it.

The remainder of the paper is organized as follows. Section 2 formally describes the DVRPMS and presents our ILP formulation. In Section 3, the details of the proposed heuristic algorithm is described. The computational experiments are reported in Section 4 in which we make a comparative analysis between our methods here proposed and the ones already described in the literature. Finally, in Section 5 we present our conclusions and emphasize the contributions of this paper.

2. Problem definition

As introduced in Iori and Riera-Ledesma (2015), the DVRPMS can be formally described as follows. Let $I = \{1, 2, \ldots, n\}$ be the set of customer requests carried by the vehicles in the pickup and delivery regions. Let also $V_c^P = \{1^P, 2^P, \ldots, n^P\}$ be the set of customers related to the pickup regions and $V_c^D = \{1^D, 2^D, \ldots, n^D\}$ the corresponding customers in the delivery region. Following the region dependencies, each request $i \in I$ corresponds to its $i^T \in V_c^T$ vertex, where T refers to any of the two regions.

It is possible to represent the DVRPMS as a directed graph $G = (V, E)$, where V is the set of vertices given by $V = V^P \cup V^D$, where $V^P = \{0^P\} \cup V_c^P$ and $V^D = \{0^P\} \cup V_c^D$. The members 0^P and 0^D are the depots for the pickup and delivery regions and members V_c^P and V_c^D are, respectively, the sets of vertices excluding the depots for the pickup and delivery regions. Likewise, the set of arcs is given by $E = E^P \cup E^D$, where $E^P = \{(i^P, j^P) \in V^P \times V^P \mid i^P \neq j^P\}$ and $E^D = \{(i^D, j^D) \in$

$V^D \times V^D \mid i^D \neq j^D\}$. For each arc $(i, j) \in E^P$ and $(i, j) \in E^D$, there is an associated routing cost of c_{ij}^P and c_{ij}^D, respectively.

Let K be the set of vehicles available to meet the transportation requirements. The loading compartment (container) of each vehicle $k \in K$ is divided into R_k stacks, all with the same height L_k. All vehicles in the K set must start and end their routes in the depots defined in each region. The vehicles must collect all items from customers located in the pickup region, store those items in their container, and then deliver the items to the respective customers located in the delivery region.

The routes of each vehicle must satisfy the LIFO policy in all its stacks, that is, if a client located at vertex i^P (pickup region) is visited before the client located at vertex j^P (pickup region), and the requested item j is stored in the same stack in which the i item was stored, then the request client j must be visited (vertex j^D) before the client of the requisition i (vertex i^D) in the delivery region.

The objective of the DVRPMS is to serve all $|I|$ requests, so that the total distance traveled by the $|K|$ vehicles is the smallest possible one.

The DVRPMS can be formally modeled as a binary ILP problem. In the rest of this section, we describe an ILP formulation that was based on the mathematical formulation described by Chagas and Santos (2016) for the DVRPMSHD, which in turn was based on the mathematical formulation of the DTSPMS proposed by Petersen and Madsen (2009) and on the mathematical formulation for the DVRPMS proposed by Iori and Riera-Ledesma (2015). The variables used in the ILP formulation are described below:

- x_{ij}^{kT} : binary variable that gets 1 if the vehicle k crosses the arc (i, j) in the region T, and 0 otherwise.
- y_{ij}^{T} : binary variable that gets 1 if the vertex i is visited before the vertex j in region T, and 0 otherwise.
- w_i^k : binary variable that gets 1 if the vehicle k carries out the requested item i, and 0 otherwise.
- z_{ir}^k : binary variable that gets 1 if the item referring to the request i is stored in the rth stack of the vehicle k, and 0 otherwise.

With these variables, we can describe the following ILP formulation for the DVRPMS:

$$\min \sum_{k \in K} \sum_{T \in \{P,D\}} \sum_{(i,j) \in E^T} c_{ij}^T \cdot x_{ij}^{kT} \tag{1}$$

s.t.

$$\sum_{j \in V^T} x_{0j}^{kT} = 1 \qquad k \in K, T \in \{P, D\} \tag{2}$$

$$\sum_{i \in V^T} x_{i0}^{kT} = 1 \qquad k \in K, T \in \{P, D\} \tag{3}$$

$$\sum_{i \in V^T \setminus \{j\}} x_{ij}^{kT} = w_j^k \qquad k \in K, T \in \{P, D\}, j \in V_c^T \tag{4}$$

$$\sum_{j \in V^T \setminus \{i\}} x_{ij}^{kT} = \sum_{j \in V^T \setminus \{i\}} x_{ji}^{kT} \qquad k \in K, T \in \{P, D\}, i \in V_c^T \tag{5}$$

$$\sum_{k \in K} w_i^k = 1 \qquad i \in I \tag{6}$$

$$\sum_{i \in I} z_{ir}^k \leq L_k \qquad k \in K, r = 1..R_k \tag{7}$$

$$\sum_{r=1}^{R_k} z_{ir}^k = w_i^k \qquad k \in K, i \in I \tag{8}$$

$$y_{ij}^T + y_{ji}^T = 1 \qquad T \in \{P, D\}, i \in V_c^T, j \in V_c^T \setminus \{i\} \tag{9}$$

$$y_{il}^T + y_{lj}^T \leq y_{ij}^T + 1 \qquad T \in \{P, D\}, l \in V_c^T, i \in V_c^T \setminus \{l\}, j \in V_c^T \setminus \{i, l\} \tag{10}$$

$$x_{ij}^{kT} \leq y_{ij}^T \qquad k \in K, T \in \{P, D\}, i \in V^T, j \in V^T \setminus \{i\} \tag{11}$$

$$y_{ij}^P + z_{ir}^k + z_{jr}^k \leq 3 - y_{ij}^D \qquad k \in K, i \in I, j \in I \setminus \{i\}, r = 1..R_k \tag{12}$$

$$\sum_{j \in I} j \cdot x_{0j}^{kP} \leq \sum_{j \in I} j \cdot x_{0j}^{k'P} \qquad k \in K, k' \in K \mid k < k', R_k = R_{k'}, L_k = L_{k'} \tag{13}$$

$$x_{ij}^{kT} \in \{0, 1\} \qquad k \in K, T \in \{P, D\}, (i, j) \in E^T \tag{14}$$

$$y_{ij}^T \in \{0, 1\} \qquad T \in \{P, D\}, i \in V^T, j \in V^T \setminus \{i\} \tag{15}$$

$$z_{ir}^k \in \{0, 1\} \qquad k \in K, i \in I, r = 1..R_k \tag{16}$$

$$w_i^k \in \{0, 1\} \qquad k \in K, i \in I. \tag{17}$$

The objective function of the problem is defined by Equation (1), which minimizes the total distance traveled by the vehicles. Constraints (2) and (3) ensure that each vehicle starts and ends its route in the depot of each region. Note that for instances where the total capacity of the vehicles is greater than the total customer demand, a vehicle k may not be used, in this case, the variable x_{00}^{kT} gets 1 $\forall\, T \in \{P, D\}$, indicating that the vehicle did not serve any customer request. Constraints (4) ensure that each request j is served by a vehicle k only if the vehicle k reaches the vertex j. Constraints (5) guarantee that the same vehicle must arrive and leave a vertex that represents the location of a client. Constraints (6) ensure that each request is served by one and only one vehicle. Constraints (7) ensure that the capacity of the stacks is not extrapolated. Constraints (8) ensure that the item of a request served by a vehicle k must be stored in its container. Constraints (9) and (10) establish a visitation order between all vertex pairs in both regions and ensure a transitivity in this order, respectively. In other words, if i precedes l and l precedes j, then i precedes j. Constraints (11) ensure that if an arc (i, j) is traversed by a vehicle, then i is strictly visited before j. Constraints (12) indicate the restrictions regarding the LIFO policy applied in all stacks of all vehicles. If the item related to the requests i and j are stored in the same stack r of vehicle k, being i visited before j in the pickup region, then i cannot be visited before j in the delivery region. Constraints (13) break the resulting symmetry from the formulation, imposing a lexicographic order on the routes of

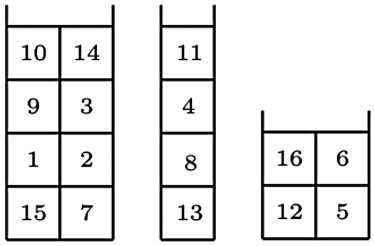

Fig. 4. DVRPMS solution representation example (Chagas et al., 2016).

vehicles with the same container configuration. And, finally, constraints (14)–(17) define the scope and domain of the decision variables.

Although this formulation does not use advanced optimization methods (e.g., branch-cut, branch-and-price, among others), it may be used as an alternative to the exact methods proposed by Iori and Riera-Ledesma (2015).

3. Heuristic approach

Since the DVRPMS is a complex combinatorial problem, for some large instance none of the exact methods proposed by Iori and Riera-Ledesma (2015) and neither our ILP formulation could solve the problem within an acceptable time. To work around this difficulty, implementing heuristic algorithms to solve large-scale instances is inevitable. Considering that the VNS metaheuristic is widely used in the literature to solve combinatorial problems (Hansen and Mladenović, 2001), we also have been motivated to propose a method based on VNS to solve the DVRPMS. The remainder of this section is intended to describe it in detail.

3.1. Solution representation

The solution representation defined by Chagas et al. (2016) was used in this paper, where a solution for the DVRPMS is represented by the items of the $|I|$ customer requests, which are distributed and allocated in the containers of the $|K|$ vehicles in the fleet. This representation defines the loading plan of the items in each container and, consequently, defines the loading/unloading constraints that must be obeyed in order to respect the LIFO policy.

Figure 4 shows an example of representation for a solution involving 16 transport requests and 3 vehicles with containers of dimensions (2×4), (1×4), and (2×2). As seen in the figure, the representation of the solution only informs the designation of the requests and the container loading plan for each vehicle. For each vehicle, the routes in the pickup and delivery regions are obtained by the evaluation functions described in Section 3.2.

3.2. Evaluation function

As previously mentioned, the solution representation does not report the routes performed by each vehicle. We assign this task to the evaluation function, which is responsible to evaluate

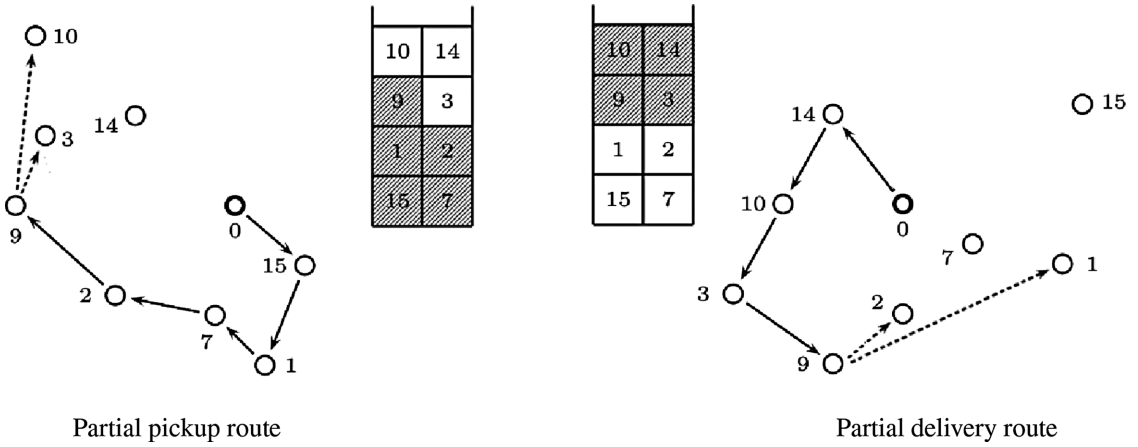

Fig. 5. Greedy phase of the evaluation heuristic.

and determine a route in the pickup region and another in the delivery region for each vehicle from its container (solution representation) and thus to return the total distance traveled by the vehicles.

Finding the shortest route in the pickup region (or the delivery region) with LIFO constraints imposed by a known loading plan can be seen as the traveling salesman problem with precedence constraints (TSPPC), proposed by Savelsbergh and Sol (1995), where the precedence constraints are given by the loading plan. According to Moon et al. (2002), TSPPC belongs to the class of problems $\mathcal{NP}$-hard problems, therefore the optimal solution to the problem cannot be obtained within a reasonable computational time when large instances are considered.

In this work, we developed two different evaluation functions. At first, we consider the evaluation function, denoted as $f_{opt}(\cdot)$, that consists in applying the dynamic programming algorithm described by Casazza et al. (2012) in each container $k \in K$ to determine the shortest pickup (delivery) route that satisfies the LIFO constraints imposed by the loading plan.

From preliminary tests, we concluded that evaluating all solutions explored by the VNS algorithm using the evaluation function $f_{opt}(\cdot)$ is impracticable due to its exponential complexity, that is, though the number of stacks is small, during the execution of the algorithm a countless number of solutions are evaluated. Therefore, we proposed a simple heuristic evaluation, denoted as $f_{heur}(\cdot)$, in order to find good-quality routes in shorter computational time. This heuristic evaluation can be divided into two phases that are subsequently performed: a greedy phase and a local search phase.

In the greedy phase, for each vehicle, a pickup route (respectively, a delivery route) is constructed choosing at each time, among the items at the bottom (top) of each stack, the item that has the least impact (least distance) on the pickup (delivery) route. In other words, at each moment an item is chosen, the pickup route (delivery route) is constructed, so that the item collected (delivered) is the closest item to the partially constructed route.

Figure 5 illustrates the greedy phase of the evaluation heuristic when applied to the pickup region (a) and other in the delivery region (b), considering a vehicle with a container of dimensions (2×4). The dashed positions in the container indicate that the items already have been inserted in the routes by previous iterations. In the case shown in Fig. 5(a), the last vertex inserted in the pickup

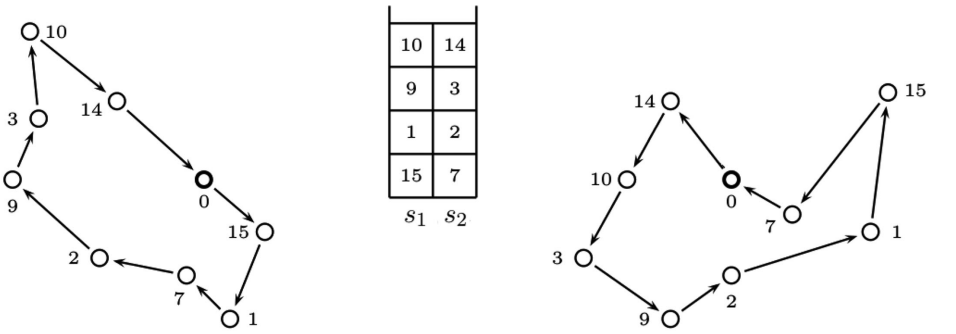

Fig. 6. Representation of the routes through the operations carried out in the stacks.

route is vertex 9^P, which refers to request 9. From this vertex, only vertices 3^P and 10^P, associated with requests 3 and 10, respectively, are accessible in the pickup region, since the constraints of the loading plan must be satisfied. Amid vertices 3^P and 10^P, vertex 3^P will be chosen, since vertex 3^P is the closest to the vertex 9^P. Similarly, in Fig. 5(b) the vertex 2^D will be chosen, since, among the candidates 1^D and 2^D, 2^D is the closest to the last vertex (9^D) inserted in the delivery route.

Note that the routes assigned to a vehicle can be represented by the order that the operations are performed on the container stacks. Figure 6 illustrates this statement, where the stacks of a container of size (2×4) were named s_1 and s_2 to facilitate the appropriate referencing of each stack. The sequences $\mathcal{S}^P$ and $\mathcal{S}^D$ represent, respectively, the route assigned to the vehicle in the pickup and delivery regions.

The second phase of the evaluation heuristic consists of applying a refinement algorithm to the routes determined by the greedy phase. In this phase, a local search is applied separately in each one of the sequences $\mathcal{S}^P$ and $\mathcal{S}^D$ in order to find better routes. The neighborhood structure defined to perform this local search consists of exchanging two operations s_i and $s_j \mid i \neq j$ of their positions. Figure 7 shows a neighbor $\mathcal{S}^{P'}$ from $\mathcal{S}^P$, as well as the changes caused in the pickup route.

In the local search procedure, the inspection of neighbors is done casually (a neighbor is chosen randomly from the neighbors of a neighborhood structure) and the neighborhood of the current sequence is explored until a sequence that represents a shorter route is found than the route already known, that is, we use a first improvement strategy.

In possession of a solution representation and a defined strategy to evaluate these solutions, the next section defines and details the neighborhood structures to be applied to DVRPMS solutions.

3.3. Neighborhood structures

In order to explore the solution space of the DVRPMS, we define four neighborhood structures: item swap (IS), item ejection chain (IC), stack permutation (SP), and stack swap (SS). All these structures are used in our VNS algorithm and are described and detailed as follows.

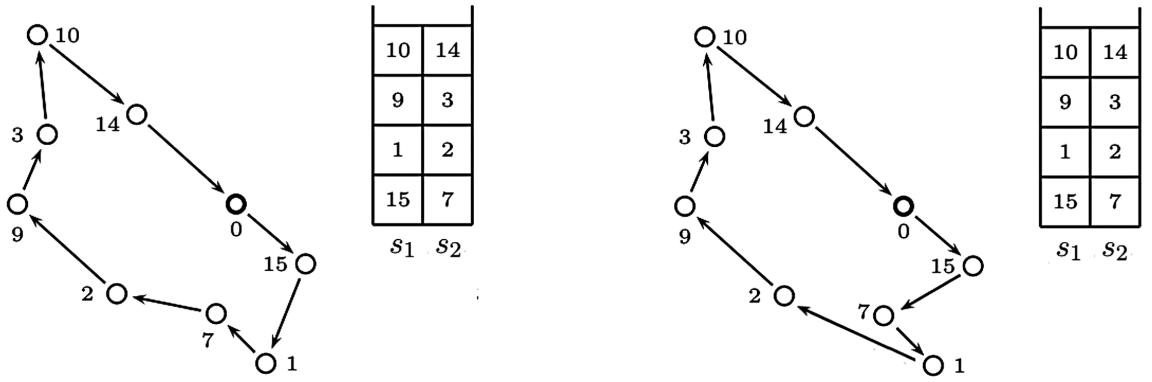

Fig. 7. Local search phase of the evaluation heuristic.

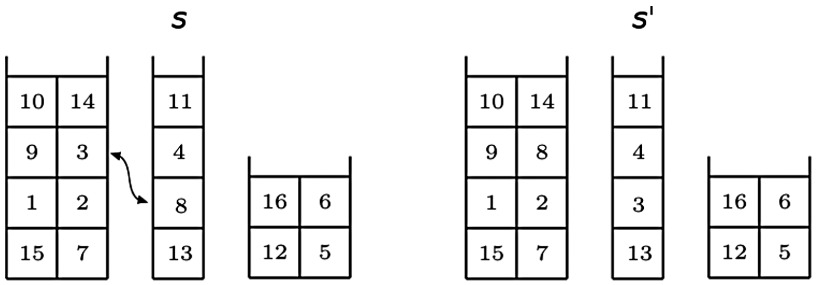

Fig. 8. An item swap (IS) example (Chagas et al., 2016).

3.3.1. Item swap

The IS neighborhood of a solution of the DVRPMS, defined by Chagas et al. (2016), contains the solutions that can be obtained by swapping two items of the containers. The items to be relocated may belong to the same container or to different container, then the size of IS neighborhood is $O(\binom{|I|}{2}) = O(|I|^2)$. Figure 8 shows an example of a solution s and one of its neighbors s'.

3.3.2. Item ejection chain

The IC neighborhood of a solution of the DVRPMS contains the solutions that can be obtained by exchanging three items of the containers by way of ejection chain, that is, the first item is relocated in position of the second one, the second item is relocated in position of the third one, and the third item is relocated in position of the first one. As in the neighborhood structure IS, the items to be relocated may belong to the same container or to a different container, so the size of IC neighborhood is $O(|I|^3)$. Figure 9 shows a solution s and one of its neighbors s'.

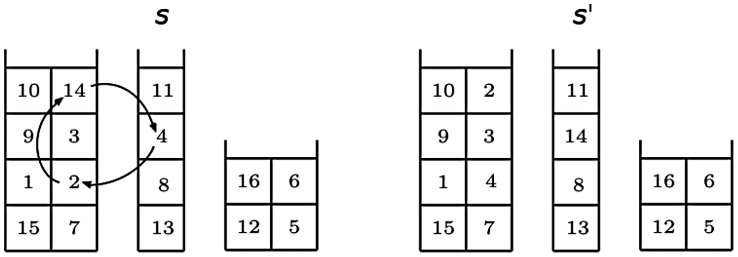

Fig. 9. An item ejection chain example.

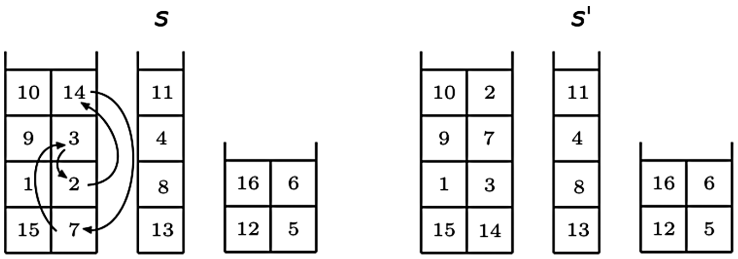

Fig. 10. A stack permutation example.

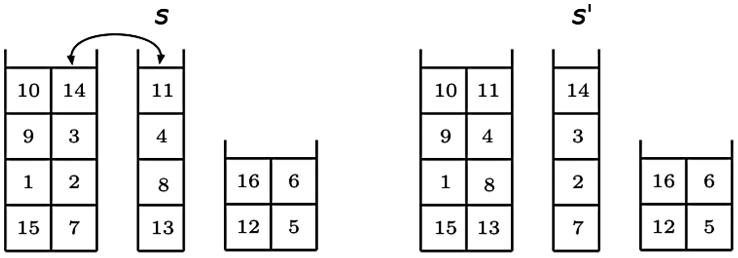

Fig. 11. A stack swap example.

3.3.3. Stack permutation

The SP neighborhood of a solution of the DVRPMS contains the solutions that can be obtained by rearranging items from the same row (horizontal stack) using a permutation of these items. As the total number of stacks of a solution is $\sum_{k \in K} R_k$ and each stack of size L_k has $L_k!$ permutations, the size of SP neighborhood is $O(\sum_{k \in K} L_k!\ R_k)$. Figure 10 shows an example of a solution s and one of its neighbors s'.

3.3.4. Stacks swap

The SS neighborhood of a solution of the DVRPMS contains the solutions that can be obtained by swapping two stacks of the different containers. As the stacks to be relocated must belong to different containers, the size of SS neighborhood is given by a combination of the total number of stacks grouped in pairs, disregarding the pairs of stacks that belong to the same vehicle, that is, $O(\binom{\sum_{k \in K} R_k}{2} - \sum_{k \in K} \binom{R_k}{2})$. Figure 11 shows an example of a solution s and one of its neighbors s'.

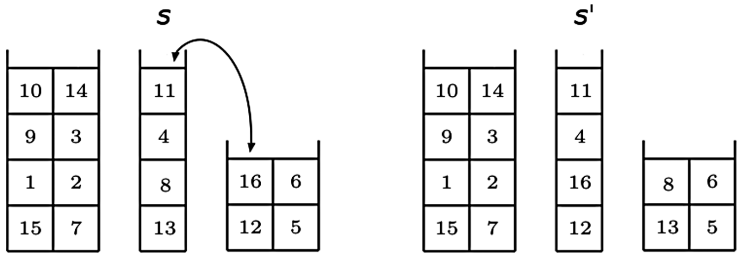

Fig. 12. A stack swap example, when the heights of the stacks are different.

For two stacks with different heights, the swap is made only between the items of the smallest stack and the items loaded in the first (bottom-up) positions of the largest stack. Figure 12 shows an example for this situation.

3.4. Initial solution

The initial solution of our VNS algorithm is created randomly. A random subset of customer requests is assigned to each vehicle in such a way that the number of requests does not exceed the capacity of each vehicle and that each request is to be served by a single vehicle.

3.5. Variable neighborhood search

The VNS metaheuristic, proposed by Mladenović and Hansen (1997) (for a recent description, see, e.g., Hansen and Mladenović, 2014), is a higher level procedure widely used to solve a large variety of practical and complex problems (Felipe et al., 2009; Tricoire et al., 2011; Wei et al., 2014, 2015; Pinto et al., 2020; Smiti et al., 2020). In its simplest form, known as basic VNS (Hansen and Mladenović, 2001), the VNS requires a local search procedure and a set of neighborhood structures $\mathcal{N}_k$ ($k = 1, 2, \ldots, k_{\max}$) which are used to perform shaking moves in order to escape from local optima solutions found throughout the algorithm.

Algorithm 1 describes our proposed VNS, where $f_{heur}(\cdot)$ and $f_{opt}(\cdot)$ are the functions that run the evaluation heuristics described in Section 3.2 for each container and return the total distance traveled by the vehicles. Initially, we define the set $\mathcal{N}$ that consists of the four neighborhood structures previously described. These neighborhood structures are arranged hierarchically according to their perturbation strength, that is, the first neighborhood structure is the `IS`, followed by the `IC`, `SP`, and `SS`. The algorithm's initial solution (line 3) is generated randomly according to Section 3.4. While the number of iterations without improvement has not reached the established maximum (*iter_max*), the algorithm chooses a random neighbor s' using the kth neighborhood structure (initially $k = 1$) from the current solution s and then applies a local search in s', which produces a new solution s''. The local search (see Algorithm 2) consists of applying a descent method using the `IS` neighborhood structure with a first improvement strategy. If the solution s'' is better than s, s'' becomes the current solution and the procedure is repeated using the first neighborhood structure (k is reset to 1), otherwise the procedure is repeated using the next neighborhood structure ($k + 1$). The method stops when the current number of iterations reaches the stopping criteria, noted by

iter_max. Then, the best solution found is evaluated by the function $f_{opt}(\cdot)$ and then returned. Note that during the internal part of the algorithm (lines 5–19), all the solutions are evaluated by function $f_{heur}(\cdot)$. This means that throughout the algorithm, each solution is evaluated heuristically, favoring efficiency, while the last solution is evaluated optimally, favoring quality, so that the functioning of the algorithm does not become costly (time-consuming).

Algorithm 1. Variable neighborhood search

1: Let $\mathcal{N} = \{\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \mathcal{N}_4\} = \{$`IS, IC, SP, SS`$\}$ be the set of neighborhood structures
2: $k_{\max} \leftarrow |\mathcal{N}|$
3: $s \leftarrow$ generate a random solution
4: $iter \leftarrow 0$
5: **while** $iter < iter\_max$ **do**
6: $k \leftarrow 1$
7: **repeat**
8: $s' \leftarrow$ pick a random neighbor in the k-th neighborhood structure of s ($s' \in \mathcal{N}_k(s)$)
9: $s'' \leftarrow$ apply local search in s' using the `IS` neighborhood structure ▷ Algorithm 2
10: **if** $f_{heur}(s'') < f_{heur}(s)$ **then**
11: $s \leftarrow s''$
12: $k \leftarrow 1$
13: $iter \leftarrow 0$
14: **else**
15: $k \leftarrow k + 1$
16: $iter \leftarrow iter + 1$
17: **end if**
18: **until** $k > k_{\max}$
19: **end while**
20: $f_{opt}(s)$
21: **return** `s`

Algorithm 2. Local search

1: Let s be the solution in which the local search will be applied
2: Let `IS`(s) be the set of solutions in the item swap (IS) neighborhood of solution s
3: $improv \leftarrow$ **true**
4: **while** $improv =$ **true do**
5: $improv \leftarrow$ **false**
6: $neighbors \leftarrow$ `IS`(s)
7: **while** $neighbors \neq \{\}$ **and** $improv =$ **false do**
8: $s' \leftarrow$ pick a random neighbor $s' \in neighbors$
9: **if** $f_{heur}(s') < f_{heur}(s)$ **then**
10: $s \leftarrow s'$
11: $improv \leftarrow$ **true**
12: **else**
13: $neighbors \leftarrow neighbors \setminus \{s'\}$
14: **end if**
15: **end while**
16: **end while**
17: **return** `s`

4. Computational results

The VNS algorithm was implemented in C++ and was sequentially (nonparallel) performed on a computer with the same settings as those used in the experiments reported by Chagas et al. (2016), that is, on an Intel Core i5-3570 @ 3.40 GHz × 4 computer with 16 GB RAM running the operating system Ubuntu 14.04 LTS 64 bits. Since for some instances, the ILP formulation requires a large amount of memory, we ran it on an Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20 GHz × 40 computer with 384 GB of RAM, running the operating system CentOS Release 6.8 (Final) Kernel Linux 2.6.32-642.1.1.el6.x86_64. Our ILP formulation was implemented in C++ using the Concert Technology Library of CPLEX 12.5 under an academic license, with all CPLEX default settings, except for the runtime that was limited to three hours. It is worth mentioning that unlike the VNS algorithm, the ILP formulation was executed on multiple threads, as defined by the CPLEX default settings.

4.1. Benchmark instances

The set of benchmark instances used to validate the proposed approaches to solve the DVRPMS was the one described by Iori and Riera-Ledesma (2015). The authors defined 24 different types of instances, where each type contains the number of customer requests, the number of vehicles available, and the configurations of the vehicle's loading compartments. These 24 types of instances are divided into two sets. The first one, denoted by C, contains 15 types of instances for which the total capacity of the vehicles is equal to the total number of customer requests (number of items), that is, for all these types of instances the containers must be fully loaded in order to serve all customers. The second set, denoted by $\neg C$, contains nine types of instances for which the total number of customer requests (number of items) is less than the total capacity of the vehicles, so the containers do not need to be completely filled in order to serve customers. Tables 1 and 2 show, respectively, the specifications of sets C and $\neg C$. Each type is described by the number of customer requests $|I|$, the number of vehicles $|K|$, and the configuration of the containers of the vehicle fleet (column $(R \times L)'s$). The last column of each table shows the total capacity of each type of fleet.

Regarding customer locations, for each type of instances previously defined, Iori and Riera-Ledesma (2015) used the data from five DTSPMS benchmark instances (R05, R06, R07, R08, and R09), giving in total 120 instances. Each of these instances reports the customers' locations in the pickup region and in the delivery region. These locations were chosen at random within two different regions of dimensions 100 × 100, and the depot of each region was fixed in coordinates (50, 50). The distance between any two points of the same region was calculated using the rounded Euclidean distance.

4.2. Parameter tuning

The proposed VNS algorithm has only one parameter (number of iterations without improvement) that is referenced as *iter_max* and it is responsible for stopping the execution of the algorithm

Table 1
Instance specifications of group C

T	$\|I\|$	$\|K\|$	$(R \times L)'s$	$\sum_{k \in K} R_k L_k$
(a)	12	2	(2×3) (2×3)	12
(b)	12	2	(2×2) (2×4)	12
(c)	12	3	(2×2) (2×2) (2×2)	12
(d)	16	2	(2×4) (2×4)	16
(e)	16	3	(2×2) (2×3) (2×3)	16
(f)	16	4	(2×2) (2×2) (2×2) (2×2)	16
(g)	18	2	(2×3) (3×4)	18
(h)	18	3	(2×3) (2×3) (2×3)	18
(i)	18	4	(2×2) (2×2) (2×2) (2×3)	18
(j)	20	2	(2×4) (3×4)	20
(k)	20	3	(2×3) (2×3) (2×4)	20
(l)	20	4	(2×3) (2×3) (2×2) (2×2)	20
(m)	24	2	(3×4) (3×4)	24
(n)	24	3	(2×4) (2×4) (2×4)	24
(o)	24	4	(2×3) (2×3) (2×3) (2×3)	24

Table 2
Instance specifications of group $\neg C$

T	$\|I\|$	$\|K\|$	$(R \times L)'s$	$\sum_{k \in K} R_k L_k$
(p)	18	2	(4×4) (4×4)	32
(q)	18	3	(3×4) (3×4) (3×4)	36
(r)	18	4	(2×4) (2×4) (2×4) (2×4)	32
(s)	20	2	(4×4) (4×4)	32
(t)	20	3	(3×4) (3×4) (3×4)	36
(u)	20	4	(2×4) (2×4) (2×4) (2×4)	32
(v)	24	2	(4×4) (4×4)	32
(w)	24	3	(3×4) (3×4) (3×4)	36
(x)	24	4	(2×4) (2×4) (2×4) (2×4)	32

(stopping criteria). Naturally, the higher the *iter_max* value, the wider the search and, consequently, the algorithm finds better solutions but spends more time. In order to balance the quality of the solutions and the execution time, for each instance, we ran 30 times the VNS algorithm with different values of *iter_max* and constructed the chart shown in Fig. 13. The values tested for *iter_max* are arranged on the horizontal axis of the chart, such values have been defined in function of the number of customer requests $|I|$. For each *iter_max* value, we plot the average value of the objective function and also the average execution time.

After analyzing the chart, it is inferred that from $item\_max = 7\ |I|$, the improvement is small in relation to the processing time. Therefore, the final experiments were performed with $item\_max = 7\ |I|$.

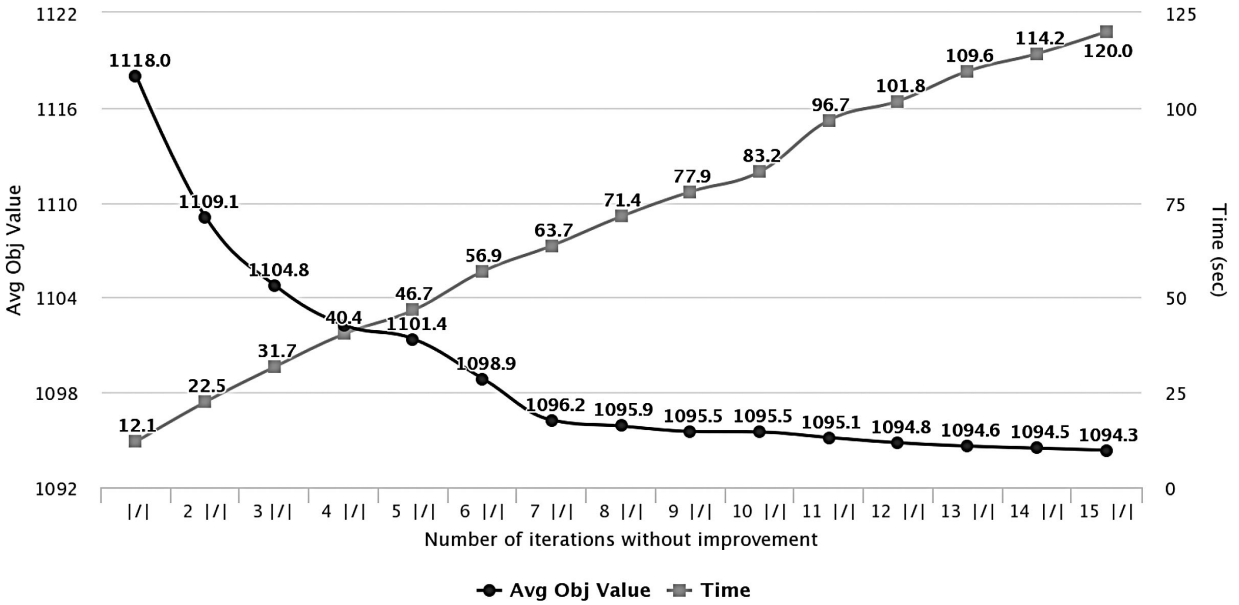

Fig. 13. Analysis of the stopping criteria of the VNS algorithm.

4.3. Results on test instances

The results obtained by our approaches to solve the DVPRMS is reported in this section. As pointed before, the execution time of the ILP formulation was limited to three hours. All exact algorithms proposed by Iori and Riera-Ledesma (2015) were limited to one hour. Our VNS algorithm was ran 10 independent times and the average and best value of the objective function obtained in these 10 runs were used in our analysis.

Tables 3 and 4 report the results on instances of group C, that is, those in which the total capacity of the fleet of vehicles and the number of customer requests are exactly the same. Thus, the containers must be completely filled so that all customers are served. Table 3 shows the results for the 30 instances (ID 001–ID 030), considered by Iori and Riera-Ledesma (2015) as small instances, and Table 4 shows the results for the 45 large instances (ID 031–ID 075). Table 5 reports the results for the 45 instances of group $\neg C$ (ID 076–ID 120), that is, those in which the total fleet capacity of vehicles is larger than the number of customers and, therefore, the containers will not necessarily be completely filled.

The first three columns of each table describe the instances, being that each instance is identified by a number (column ID), a name indicating the pickup and delivery regions (column R), and a type of instance (column T). The best results obtained by Iori and Riera-Ledesma (2015) are presented in the columns UB and t (seconds) that indicate, respectively, the upper bound and the execution time in seconds. The results obtained by our ILP formulation are described in columns LB, UB, $Gap\%$, t (seconds), and Opt. Columns LB and UB, respectively, indicate the lower bound and upper bound obtained at the end of the execution. Column $Gap\%$ shows the relative gap between UB and LB, which can be calculated as $100 \times (UB - LB)/UB$. Column t (seconds)

Table 3
Comparative analysis on smaller-size instances of group *C*

Instance			Iori and Riera-Ledesma (2015)		ILP formulation					Silveira et al. (2015)		Chagas et al. (2016)			VNS		
ID	*R*	*T*	*UB*	*t* (seconds)	*UB*	*LB*	*Gap%*	*t* (seconds)	*Opt*	*Best*	*t* (seconds)	*Avg*	*Best*	*t* (seconds)	*Avg*	*Best*	*t* (seconds)
001	R05	(*a*)	738	2	**738**	738.0	0.00	46	*	746	31	738.0	**738**	13	738.0	**738**	2
002	R06		895	0	**895**	895.0	0.00	18	*	897	30	895.0	**895**	13	895.0	**895**	2
003	R07		761	5	**761**	761.0	0.00	486	*	**761**	31	761.0	**761**	13	761.0	**761**	2
004	R08		848†	0	**851**	851.0	0.00	27	*	**851**	31	851.0	**851**	13	851.0	**851**	2
005	R09		771†	0	**776**	776.0	0.00	20	*	781	31	776.0	**776**	14	776.0	**776**	2
006	R05	(*b*)	716	1	**716**	716.0	0.00	23	*	728	31	716.9	**716**	3	716.0	**716**	1
007	R06		859†	2	**866**	866.0	0.00	18	*	873	31	866.0	**866**	3	877.4	**866**	1
008	R07		733	3	**733**	733.0	0.00	95	*	**733**	31	733.0	**733**	3	733.0	**733**	1
009	R08		858	3	**858**	858.0	0.00	31	*	875	31	858.0	**858**	3	865.5	**858**	1
010	R09		738†	1	**741**	741.0	0.00	16	*	760	31	741.5	**741**	3	752.8	**741**	1
011	R05	(*c*)	855	0	**855**	855.0	0.00	72	*	859	35	855.0	**855**	5	855.0	**855**	1
012	R06		1011	0	**1011**	1011.0	0.00	78	*	**1011**	35	1011.0	**1011**	5	1011.0	**1011**	1
013	R07		894	1	**894**	894.0	0.00	367	*	**894**	35	894.0	**894**	5	894.0	**894**	1
014	R08		990	1	**990**	990.0	0.00	146	*	**990**	35	990.0	**990**	5	990.0	**990**	1
015	R09		852	0	**852**	852.0	0.00	54	*	853	35	852.0	**852**	5	852.0	**852**	1
016	R05	(*d*)	947	329	**947**	919.0	2.96	3 h		976	32	948.1	948	1	950.2	**947**	5
017	R06		1036	43	**1036**	1036.0	0.00	1625	*	1093	32	1040.2	**1036**	1	1045.1	**1036**	6
018	R07		925	46	**925**	925.0	0.00	3303	*	971	32	925.0	**925**	1	926.6	**925**	4
019	R08		1006†	52	**1010**	1010.0	0.00	1632	*	1060	33	1021.5	1020	1	1014.6	**1010**	6
020	R09		907†	97	**921**	921.0	0.00	3195	*	3195	10	925.1	925	1	924.1	**921**	5
021	R05	(*e*)	1050	21	**1050**	1050.0	0.00	4225	*	1087	37	1050.0	**1050**	21	1051.6	**1050**	5
022	R06		1102†	6	**1114**	1114.0	0.00	1778	*	**1114**	37	1114.0	**1114**	20	1114.0	**1114**	4
023	R07		1063	29	1072	958.1	10.63	3 h		1089	37	1063.0	**1063**	20	1066.9	**1063**	4
024	R08		1117†	10	**1126**	1126.0	0.00	2785	*	1142	36	1130.8	**1126**	21	1136.3	**1126**	4
025	R09		1021	9	**1021**	1021.0	0.00	2187	*	1046	36	1021.6	**1021**	21	1026.1	**1021**	5
026	R05	(*f*)	1217	37	**1217**	1054.5	13.35	3 h		1223	42	1217.0	**1217**	10	1217.0	**1217**	2
027	R06		1290	10	**1290**	1155.5	10.43	3 h		**1290**	41	1290.0	**1290**	10	1290.0	**1290**	2
028	R07		1230	44	**1230**	1046.8	14.90	3 h		**1230**	41	1230.0	**1230**	10	1230.0	**1230**	2
029	R08		1261	18	**1261**	1148.3	8.94	3 h		1262	44	1261.0	**1261**	10	1261.0	**1261**	2
030	R09		1127	7	**1134**	996.9	12.09	3 h		**1134**	43	1134.0	**1134**	10	1134.0	**1134**	2
Average			960.6	25.9	963.0	934.0	2.44	3260.9	$\frac{23}{30}$	976.0	33.9	963.7	963.2	8.8	965.2	962.7	2.5

Table 4
Comparative analysis on larger-size instances of group *C*

Instance			Iori and Riera-Ledesma (2015)		ILP formulation					Silveira et al. (2015)		Chagas et al. (2016)			VNS		
ID	*R*	*T*	*UB*	*t* (seconds)	*UB*	*LB*	*Gap%*	*t* (seconds)	*Opt*	*Best*	*t* (seconds)	*Avg*	*Best*	*t* (seconds)	*Avg*	*Best*	*t* (seconds)
031	R05	(*g*)	950	9	**950**	950.0	0.00	778	*	1083	31	952.5	**950**	19	964.8	**950**	16
032	R06		1012†	27	**1024**	1024.0	0.00	7224	*	1087	31	1038.8	**1024**	19	1035.6	**1024**	15
033	R07		932	67	**932**	932.0	0.00	8592	*	1054	32	936.8	**932**	21	943.4	**932**	13
034	R08		1011†	50	**1018**	1018.0	0.00	3635	*	1128	31	1040.7	1031	20	1037.8	1024	16
035	R09		909†	48	**919**	919.0	0.00	2860	*	985	31	937.4	**919**	21	931.5	**919**	18
036	R05	(*h*)	1147	60	**1147**	1021.9	10.91	3 h		1198	41	1147.0	**1147**	33	1153.9	**1147**	8
037	R06		1165	8	**1177**	1100.0	6.54	3 h		1190	42	1177.0	**1177**	33	1177.0	**1177**	10
038	R07		1123	32	1140	930.1	18.41	3 h		1136	41	1123.0	**1123**	33	1123.0	**1123**	7
039	R08		1184	40	**1184**	1096.7	7.37	3 h		1214	41	1184.0	**1184**	33	1188.6	**1184**	9
040	R09		1080	50	1097	954.0	13.03	3 h		1111	41	1081.4	**1080**	33	1080.6	**1080**	9
041	R05	(*i*)	1269	319	1291	999.4	22.59	3 h		1292	43	1272.0	**1269**	18	1274.6	**1269**	5
042	R06		1264	38	**1275**	1084.8	14.92	3 h		1282	43	1275.1	**1275**	18	1275.1	**1275**	6
043	R07		1261	193	1281	936.8	26.87	3 h		1284	43	1261.0	**1261**	18	1273.7	**1261**	4
044	R08		1310	504	1312	1109.3	15.45	3 h		1338	44	1310.0	**1310**	18	1317.1	**1310**	6
045	R09		1157	62	**1157**	954.0	17.54	3 h		1196	44	1158.0	**1157**	18	1162.8	**1157**	6
046	R05	(*j*)	1012	115	1013	953.0	5.92	3 h		1126	34	1019.2	**1012**	6	1024.3	**1012**	21
047	R06		1018	24	**1018**	1018.0	0.00	1453	*	1177	33	1043.5	**1018**	6	1050.3	**1018**	21
048	R07		1047	290	1056	941.6	10.83	3 h		1162	34	1065.8	1054	6	1075.5	**1047**	18
049	R08		1040	751	1058	981.7	7.22	3 h		1221	10	1068.8	**1050**	8	1073.4	**1050**	18
050	R09		959	55	**973**	930.4	4.38	3 h		1126	34	985.2	977	7	987.6	974	22
051	R05	(*k*)	1174	860	1231	981.0	20.31	3 h		1212	45	1180.4	**1174**	28	1180.4	**1174**	13
052	R06		1200	66	1234	1064.4	13.74	3 h		1253	44	1209.5	**1200**	27	1227.0	**1200**	11
053	R07		1243	1471	1375	942.6	31.45	3 h		1289	44	1247.4	**1243**	28	1258.0	**1243**	11
054	R08		1196	915	1266	1050.3	17.04	3 h		1275	44	1210.9	**1206**	28	1211.3	**1206**	14
055	R09		1133	268	1178	959.2	18.57	3 h		1196	46	1151.0	**1144**	27	1154.4	**1144**	12

Continued

Table 4
Continued

Instance			Iori and Riera-Ledesma (2015)		ILP formulation					Silveira et al. (2015)		Chagas et al. (2016)			VNS		
ID	*R*	*T*	*UB*	*t* (seconds)	*UB*	*LB*	*Gap%*	*t* (seconds)	*Opt*	*Best*	*t* (seconds)	*Avg*	*Best*	*t* (seconds)	*Avg*	*Best*	*t* (seconds)
056	R05	(*l*)	1293	1306	1298	961.1	25.96	3 h		1347	49	1294.6	**1293**	30	1293.0	**1293**	11
057	R06		1340	202	1341	1043.6	22.18	3 h		1374	49	1340.4	**1340**	29	1342.4	**1340**	9
058	R07		1373	2843	1405	914.5	34.91	3 h		1418	49	1373.0	**1373**	30	1380.0	**1373**	9
059	R08		1305	460	1345	1049.4	21.98	3 h		1345	49	1311.5	**1305**	29	1323.6	**1305**	9
060	R09		1252	832	1261	896.6	28.90	3 h		1310	49	1259.2	**1258**	30	1262.3	**1258**	9
061	R05	(*m*)	1060	2326	**1069**	924.0	13.56	3 h		1220	35	1089.1	1073	14	1092.4	1071	61
062	R06		1093	70	**1093**	976.9	10.62	3 h		1324	35	1111.1	**1093**	14	1120.1	**1093**	52
063	R07		1096	263	1213	901.4	25.69	3 h		1325	35	1108.8	1103	13	1111.8	**1098**	51
064	R08		1120	1149	1230	941.5	23.46	3 h		1300	34	1156.6	**1133**	18	1159.0	**1133**	67
065	R09		1034	735	1148	863.4	24.79	3 h		1266	34	1074.6	**1047**	20	1074.6	**1047**	60
066	R05	(*n*)	1318	1 h	1407	914.2	35.03	3 h		1387	47	1276.2	**1262**	3	1275.7	**1262**	20
067	R06		1289	1 h	1550	1016.3	34.44	3 h		1438	48	1315.6	**1304**	3	1308.6	**1304**	21
068	R07		1350	1 h	1501	889.7	40.73	3 h		1429	46	1345.5	**1333**	3	1347.3	1338	22
069	R08		1297	1 h	1426	975.9	31.56	3 h		1426	47	1320.5	**1297**	3	1324.2	**1297**	22
070	R09		1239	1 h	1577	817.0	48.19	3 h		1346	47	1264.5	1243	3	1242.5	**1240**	24
071	R05	(*o*)	1518	1 h	1552	912.8	41.19	3 h		1422	53	1373.5	**1367**	60	1368.6	**1367**	25
072	R06		1449	1 h	1749	992.3	43.27	3 h		1516	53	1448.2	**1448**	60	1454.4	**1448**	25
073	R07		1677	1 h	1716	900.8	47.50	3 h		1509	48	1469.2	**1465**	61	1473.2	**1465**	28
074	R08		1463	1 h	1555	975.2	37.29	3 h		1488	52	1448.8	**1444**	61	1446.0	**1444**	25
075	R09		1570	1 h	1488	871.1	41.46	3 h		1413	52	1363.5	**1362**	60	1362.0	**1362**	23
Average			1198.5	1166.8	1249.6	968.7	20.35	9905.4	$\frac{6}{45}$	1271.5	41.3	1196.0	1188.0	23.8	1198.7	1187.5	19.6

Table 5
Comparative analysis on instances of group ¬*C*

Instance			Iori and Riera-Ledesma (2015)		ILP formulation					Chagas et al. (2016)			VNS		
ID	*R*	*T*	*UB*	*t* (seconds)	*UB*	*LB*	*Gap*%	*t* (seconds)	*Opt*	*Avg*	*Best*	*t* (seconds)	*Avg*	*Best*	*t* (seconds)
076	R05	(*p*)	912†	24	**913**	913.0	0.00	2607	*	938.9	921	34	942.8	921	26
077	R06		945†	138	**948**	948.0	0.00	7141	*	1006.5	989	53	999.2	989	24
078	R07		793†	43	**801**	801.0	0.00	1181	*	900.6	804	124	871.4	804	44
079	R08		902	5	**902**	902.0	0.00	714	*	988.5	909	96	975.7	934	29
080	R09		874	11	**874**	874.0	0.00	2790	*	893.1	889	17	882.7	880	22
081	R05	(*q*)	1031♭	31	**942**	894.6	5.03	3 h		995.0	965	9	960.4	**942**	25
082	R06		1061♭	46	**1010**	952.9	5.65	3 h		1025.2	**1010**	9	1032.4	**1010**	21
083	R07		978♭	40	**931**	816.8	12.27	3 h		933.1	**931**	9	938.9	**931**	17
084	R08		1060♭	74	**995**	964.5	3.07	3 h		1013.4	998	9	1006.1	998	20
085	R09		982♭	36	**889**	867.3	2.44	3 h		908.0	902	10	895.5	**889**	21
086	R05	(*r*)	1200♭	2303	1159	887.8	23.40	3 h		1124.5	1116	5	1116.6	**1113**	12
087	R06		1191♭	302	**1121**	944.2	15.78	3 h		1135.9	1125	5	1131.4	**1121**	9
088	R07		1105♭	191	1141	792.5	30.54	3 h		1103.4	1051	5	1093.1	**1042**	8
089	R08		1191♭	776	1198	971.5	18.91	3 h		1167.4	**1147**	5	1177.2	**1147**	10
090	R09		1100♭	2342	1050	861.1	17.99	3 h		1055.6	1048	5	1046.0	**1033**	10
091	R05	(*s*)	947	4	**950**	914.5	3.74	3 h		1001.3	991	36	989.0	**950**	46
092	R06		1010♭	8	**1005**	1005.0	0.00	4512	*	1021.2	**1005**	55	1021.5	**1005**	38
093	R07		939	8	**952**	881.2	7.44	3 h		1006.9	963	156	1005.3	963	42
094	R08		953	7	**958**	937.8	2.11	3 h		1028.2	1009	98	1009.4	963	41
095	R09		945♭	5	**940**	909.3	3.26	3 h		950.1	942	35	942.7	941	35
096	R05	(*t*)	1101♭	298	1006	891.8	11.36	3 h		1049.7	1000	12	1018.2	**998**	36
097	R06		1059♭	17	**1017**	992.6	2.40	3 h		1105.0	1074	15	1057.3	**1017**	28
098	R07		1101♭	1911	1053	840.6	20.17	3 h		1028.3	**1021**	12	1050.5	**1021**	26
099	R08		1115♭	337	1061	906.5	14.57	3 h		1051.7	1039	16	1050.6	**1025**	31
100	R09		1002♭	15	957	861.0	10.03	3 h		972.9	957	15	970.6	**953**	33

Continued

Table 5
Continued

Instance			Iori and Riera-Ledesma (2015)		ILP formulation					Chagas et al. (2016)			VNS		
ID	*R*	*T*	*UB*	*t* (seconds)	*UB*	*LB*	*Gap*%	*t* (seconds)	*Opt*	*Avg*	*Best*	*t* (seconds)	*Avg*	*Best*	*t* (seconds)
101	R05	(*u*)	1259	1 h	1268	870.6	31.34	3 h		1195.9	1159	6	1165.2	**1149**	15
102	R06		1225b	258	1197	977.3	18.36	3 h		1185.1	1179	6	1165.0	**1161**	14
103	R07		1325	1 h	1329	825.4	37.90	3 h		1241.2	**1232**	6	1235.2	**1232**	13
104	R08		1245	1 h	1336	951.0	28.82	3 h		1215.6	1207	6	1194.4	**1181**	14
105	R09		1169b	909	1200	863.0	28.08	3 h		1153.1	1136	6	1132.8	**1118**	15
106	R05	(*v*)	1033	26	1079	904.8	16.15	3 h		1075.3	1062	112	1051.9	**1033**	93
107	R06		1051	34	1084	944.9	12.83	3 h		1085.3	1056	274	1074.4	**1051**	95
108	R07		1058	52	1106	861.5	22.11	3 h		1111.4	1095	75	1086.7	**1080**	93
109	R08		1076	26	1089	923.5	15.20	3 h		1093.4	**1069**	717	1102.9	1080	90
110	R09		1019	6	1030	863.9	16.13	3 h		1039.6	1023	420	1038.5	**1022**	95
111	R05	(*w*)	1142b	2826	1289	844.7	34.47	3 h		1216.0	1194	14	1153.2	**1073**	59
112	R06		1188b	331	1271	948.1	25.40	3 h		1212.8	1204	16	1166.3	**1093**	65
113	R07		1186b	2603	1200	862.5	28.13	3 h		1210.1	1134	19	1189.4	**1098**	52
114	R08		1174b	666	1198	907.6	24.24	3 h		1249.8	1159	19	1209.0	**1157**	82
115	R09		1098b	2250	1283	814.0	36.56	3 h		1174.3	1087	24	1142.8	**1058**	64
116	R05	(*x*)	1333	1 h	1392	850.6	38.89	3 h		1348.9	1300	9	1293.3	**1262**	29
117	R06		1377	1 h	1700	937.7	44.84	3 h		1359.7	1320	9	1310.7	**1304**	29
118	R07		1412	1 h	1596	858.8	46.19	3 h		1388.9	**1338**	9	1359.7	**1338**	22
119	R08		1367	1 h	1520	909.3	40.18	3 h		1399.9	1361	10	1357.5	**1297**	27
120	R09		1578	1 h	1448	831.9	42.55	3 h		1314.8	1252	9	1263.1	**1240**	32
Average			1106.9	1061.3	1119.7	895.2	17.74	9781.1	$\frac{6}{45}$	1103.9	1075.0	58.0	1086.1	1058.2	36.6

informs the total processing time and column *Opt* indicates by an asterisk the instances solved to optimality. The next columns show the best results and the processing time (columns *Best* and *t* (seconds)) obtained by Silveira et al. (2015). The results obtained by Chagas et al. (2016) and the proposed VNS algorithm are presented in three columns, *Avg*, *Best,* and *t* (seconds) that inform, respectively, the average solution value, the best solution value, and the total processing time consumed by the 10 runs. Note that Silveira et al. (2015) did not consider instances of group $\neg C$ in their experiments, so Table 5 does not show these results. Although Chagas et al. (2016) do not show the results of instances of group $\neg C$, we executed their algorithm and reported the results in Table 5.

Tables 3 and 4 show the superiority of the results reported by Iori and Riera-Ledesma (2015) when compared to the results reached by the ILP formulation, since for every instance of group *C* the algorithms developed by Iori and Riera-Ledesma (2015) run faster than our ILP formulation and find better solutions for most instances. However, although the ILP formulation is not as efficient as the exact methods of Iori and Riera-Ledesma (2015), as it was solved with no tentative of strengthening or specialized techniques, it is of fundamental importance for this paper and for future works that will address the DVRPMS, since through the results of our ILP formulation, it was possible to note inconsistencies in the results published by Iori and Riera-Ledesma (2015).

All results proven to be inconsistent are highlighted in Tables 3 and 4 by the character †, which is inserted in column *UB* that relates to the results of Iori and Riera-Ledesma (2015). Those results were considered inconsistencies due to the lower bound (column *LB*) of the ILP formulation being larger than the upper bound (column *UB*) reported by Iori and Riera-Ledesma (2015). After noticing these inconsistencies, in June 2016 the authors were notified and after a careful examination of the details of the solution, we noticed that for some instances the solutions reported by their algorithms were unfeasible (the routes did not respect the stacks' LIFO policy). The authors shared with us their code, but we could not find the reason of the inconsistencies.

Since the results of Iori and Riera-Ledesma (2015) have not been corrected yet, we will focus on the comparative analysis of results of group *C* considering only the other methods of resolution. In addition, in order to highlight the solution quality of each method, best results reached for each instance are shown in bold.

For the small instances of group *C*, the ILP formulation was able to find the optimal solution for 23 of 30 instances, whereas for the 45 large instances, only six of them were proved to have found the optimal solution. The high relative gap values (column *Gap*%) indicate the difficulty of the ILP formulation on finding optimal solutions, mainly, for the larger instances.

It is possible to note that the VNS algorithm overcomes all results obtained by Silveira et al. (2015), finding solutions of the same or higher quality with less computational time. The VNS was also efficient in comparison to the ILP formulation, because for only three instances (ID 034, 050, and 061) the ILP formulation obtained better solutions.

We also noticed that the average solution (column *Avg*) obtained by the algorithm proposed by Chagas et al. (2016) presents slightly better results than the VNS for most instances of group *C*. However, considering the best solutions obtained by each method (column *Best*), it is possible to note that in nine instances (ID 016, 019, 020, 034, 048, 050, 061, 063, and 070) the VNS algorithm was able to find solutions of higher quality than those found by the algorithm proposed by Chagas

et al. (2016). In the other 65 instances, both methods found the same solution and only in one instance (ID 068) the VNS was worse.

Regarding the computation time, the VNS was more efficient than the algorithm proposed by Chagas et al. (2016) for most instances. On average (last row of the tables), the VNS was 3.5 times faster than their algorithm in the smaller instances of group C and 1.2 faster for the larger instances also of group C.

Again, for instances of group $\neg C$, some results (instances with ID 076, 077, and 078) obtained by Iori and Riera-Ledesma (2015) were also found to be unfeasible. In addition, we noted that several results that are stated by Iori and Riera-Ledesma (2015) as optimal results are overcome by the ILP formulation and/or by our VNS and/or by the method proposed by Chagas et al. (2016). These results are highlighted by the character ♭, which was inserted in column *UB* that relates to the results of Iori and Riera-Ledesma (2015).

For most instances of group $\neg C$, when comparing the best solution values, the method based on VNS outperformed the method described in Chagas et al. (2016), with two exceptions (ID 079 and 109). Regarding computational time, comparing the average execution time of all methods (last row of Table 5), the heuristic based on the VNS metaheuristic was faster, proving to be a reliable heuristic.

In the electronic supplementary material, which can be found at http://www.goal.ufop.br/software/dvrpms, we present the best solutions (routes and loading plan for each vehicle) found so far for the DVRPMS.

5. Conclusions

In this paper, we addressed the DVRPMS. For solving it, initially, we introduced a new ILP formulation. As the DVRPMS is a complex combinatorial problem, and finding an exact solution is time consuming, a VNS-based algorithm was also proposed to solve it.

In order to evaluate the quality of our solution approaches, we have performed tests with two sets of instances used in previous works that address the DVRPMS. The first set (denoted by C) contains instances in which the total capacity of the vehicles is equal to the total number of customer requests, while the second set (denoted by $\neg C$) contains instances in which the total number of customer requests is less than the total capacity of the vehicles.

Although the ILP formulation did not solve most larger instances of group C and most instances of group $\neg C$, this exact method was of fundamental importance for the DVRPMS because it highlighted inconsistencies in the values encountered in the literature.

Since these inconsistencies were raised in this work and our ILP formulation did not solve most instances to the optimality, it is hard to do a statistical analysis with all solutions. Though when considering the average values of the best solutions, the VNS algorithm outperformed all methods and spent less computational time, especially instances of the group $\neg C$.

As a future work, complex methods such as branch-and-price and branch-and-cut-and-price could be applied to our new ILP formulation. So, they can perform better for larger instances, consequently, find the global optima values for those instances in which the first ILP formulation was not able to solve. Developing new neighborhood structures can also vary the results and thus the VNS algorithm could even obtain better values.

Acknowledgments

The authors would like to thank Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG), Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Universidade Federal de Viçosa (UFV), and Universidade Federal de Ouro Preto (UFOP) for supporting this research.

References

Alba Martínez, M.A., Cordeau, J.F., Dell'Amico, M., Iori, M., 2013. A branch-and-cut algorithm for the double traveling salesman problem with multiple stacks. *INFORMS Journal on Computing* 25, 1, 41–55.

Barbato, M., Grappe, R., Lacroix, M., Calvo, R.W., 2016. Polyhedral results and a branch-and-cut algorithm for the double traveling salesman problem with multiple stacks. *Discrete Optimization* 21, 25–41.

Carrabs, F., Cerulli, R., Speranza, M.G., 2010. A branch-and-bound algorithm for the double TSP with two stacks. Technical Report, Dipartimento di Matematica e Informatica, Università di Salerno, Fisciano, Italy.

Casazza, M., Ceselli, A., Nunkesser, M., 2012. Efficient algorithms for the double traveling salesman problem with multiple stacks. *Computers & Operations Research* 39, 5, 1044–1053.

Chagas, J.B.C., Santos, A.G., 2016. A branch-and-price algorithm for the double vehicle routing problem with multiple stacks and heterogeneous demand. 16th International Conference on Intelligent Systems Design and Applications (ISDA), Springer, Porto, Portugal, pp. 921–934.

Chagas, J.B.C., Santos, A.G., 2017. An effective heuristic algorithm for the double vehicle routing problem with multiple stack and heterogeneous demand. 17th International Conference on Intelligent Systems Design and Applications (ISDA), Springer, Delhi, India, pp. 785–796.

Chagas, J.B.C., Silveira, U.E.F., Benedito, M.P.L., Santos, A.G., 2016. Simulated annealing metaheuristic for the double vehicle routing problem with multiple stacks. 19th International Conference on Intelligent Transportation Systems (ITSC), IEEE, Rio de Janeiro, Brazil, pp. 1311–1316.

Cordeau, J.F., Iori, M., Laporte, G., Salazar González, J.J., 2010. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading. *Networks* 55, 1, 46–59.

Côté, J.F., Archetti, C., Speranza, M.G., Gendreau, M., Potvin, J.Y., 2012. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *Networks* 60, 4, 212–226.

Felipe, Á., Ortuño, M.T., Tirado, G., 2009. The double traveling salesman problem with multiple stacks: a variable neighborhood search approach. *Computers & Operations Research* 36, 11, 2983–2993.

Hansen, P., Mladenović, N., 2001. Variable neighborhood search: principles and applications. *European Journal of Operational Research* 130, 3, 449–467.

Hansen, P., Mladenović, N., 2014. Variable neighborhood search. In Burke, E.K. (ed.), *Search Methodologies*. Springer, New York, pp. 313–337.

Iori, M., Riera-Ledesma, J., 2015. Exact algorithms for the double vehicle routing problem with multiple stacks. *Computers & Operations Research* 63, 83–101.

Lusby, R.M., Larsen, J., Ehrgott, M., Ryan, D., 2010. An exact method for the double TSP with multiple stacks. *International Transactions in Operational Research* 17, 5, 637–652.

Mladenović, N., Hansen, P., 1997. Variable neighborhood search. *Computers & Operations Research* 24, 11, 1097–1100.

Moon, C., Kim, J., Choi, G., Seo, Y., 2002. An efficient genetic algorithm for the traveling salesman problem with precedence constraints. *European Journal of Operational Research* 140, 3, 606–617.

Petersen, H.L., Archetti, C., Speranza, M.G., 2010. Exact solutions to the double travelling salesman problem with multiple stacks. *Networks* 56, 4, 229–243.

Petersen, H.L., Madsen, O.B., 2009. The double travelling salesman problem with multiple stacks—formulation and heuristic solution approaches. *European Journal of Operational Research* 198, 1, 139–147.

Pinto, T., Alves, C., Valério de Carvalho, J., 2020. Variable neighborhood search algorithms for the vehicle routing problem with two-dimensional loading constraints and mixed linehauls and backhauls. *International Transactions in Operational Research* 27, 1, 549–572.

Sampaio, A.H., Urrutia, S., 2016. New formulation and branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *International Transactions in Operational Research* 24, 77–98.

Savelsbergh, M.W., Sol, M., 1995. The general pickup and delivery problem. *Transportation Science* 29, 1, 17–29.

Silveira, U.E.F., Benedito, M.P.L., Santos, A.G., 2015. Heuristic approaches to double vehicle routing problem with multiple stacks. 15th International Conference on Intelligent Systems Design and Applications (ISDA), IEEE, Marrakesh, Marocco, pp. 231–236.

Smiti, N., Dhiaf, M.M., Jarboui, B., Hanafi, S., 2020. Skewed general variable neighborhood search for the cumulative capacitated vehicle routing problem. *International Transactions in Operational Research* 27, 1, 651–664.

Tricoire, F., Doerner, K.F., Hartl, R.F., Iori, M., 2011. Heuristic and exact algorithms for the multi-pile vehicle routing problem. *OR Spectrum* 33, 4, 931–959.

Wei, L., Zhang, Z., Lim, A., 2014. An adaptive variable neighborhood search for a heterogeneous fleet vehicle routing problem with three-dimensional loading constraints. *IEEE Computational Intelligence Magazine* 9, 4, 18–30.

Wei, L., Zhang, Z., Zhang, D., Lim, A., 2015. A variable neighborhood search for the capacitated vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research* 243, 3, 798–814.