A VNS-Based Algorithm with Adaptive Local Search for Solving the Multi-Depot Vehicle Routing Problem

Sinaide Nunes Bezerra[1], Marcone Jamilson Freitas Souza[2], Sérgio Ricardo de Souza[1(✉)], and Vitor Nazário Coelho[3]

[1] Federal Center of Technological Education of Minas Gerais (CEFET-MG), Av. Amazonas 7675, Nova Gameleira, Belo Horizonte, MG 30510-000, Brazil
sinaide@hotmail.com, sergio@dppg.cefetmg.br
[2] Federal University of Ouro Preto (UFOP), Campus Universitário, Morro do Cruzeiro, Ouro Preto, MG 35400-000, Brazil
marcone@ufop.edu.br
[3] Fluminense Federal University (UFF), Institute of Computing, Niterói, RJ, Brazil
vncoelho@gmail.com

Abstract. The Multi-Depot Vehicle Routing Problem (MDVRP) is a variant of the Vehicle Routing Problem (VRP) that consists in designing a set of vehicle routes to serve all customers, such that the maximum number of vehicle per depot, the vehicle capacity and the maximum time for each route are respected. The objective is to minimize the total cost of transportation. This paper presents an algorithm, named VNSALS, based on the Variable Neighborhood Search (VNS) with Adaptive Local Search (ALS) for solving it. The main procedures of VNSALS are perturbation, ALS and cluster refinement. The perturbation procedure of VNS is important to diversify the solutions and avoid getting stuck in local optima. The ALS procedure consists in memorizing the results found after applying a local search and in using this memory to select the most promising neighborhood for the next local search application. The choice of the neighborhood is very important to improve the solution in heuristic methods because the complexity of the local search is high and expensive. On the other hand, customer's reallocation keeps the clusters more balanced. VNSALS is tested in classical instances of MDVRP for evaluating its efficiency and the results are presented and discussed.

Keywords: Multi-Depot Vehicle Routing Problem · Adaptive Local Search · Variable Neighborhood Search

1 Introduction

Vehicle routing problem (VRP) is a classical optimization problem with several variants. Researchers are dedicated to studying it all over the world, applying the most diverse techniques for its solution, as the literature points out. In a

A. Sifaleras et al. (Eds.): ICVNS 2018, LNCS 11328, pp. 167–181, 2019.
https://doi.org/10.1007/978-3-030-15843-9_14

solution of VRP, each vehicle leaves the depot and executes a route over a certain number of customers and returns to the depot, insuring that the total demand on the route does not exceed the vehicle capacity. In some cases, a maximum route duration or distance constraint is enforced and the problem may involve a homogeneous or heterogeneous fleet [6,11,16,17].

This work has its focus on the Multi-Depot Vehicle Routing Problem (MDVRP) with homogeneous fleet. MDVRP is a variant of the classical VRP in which there is more than one depot. MDVRP is solved in [6] with Tabu Search. In [12], the authors apply an algorithm based on the Adaptive Large Neighborhood Search (ALNS). Genetic Algorithm (GA) is used in [17] and a hybrid algorithm based on Iterated Local Search (ILS) is applied in [16]. An exact method is proposed in [5]. Other hybrid metaheuristic algorithms combining Greedy Randomized Adaptive Search Procedure (GRASP), ILS and Simulated Annealing (SA) are proposed in [1]. In [11], the authors present a parallel coevolutionary algorithm based in evolution strategy and in [4] an algorithm based on the General VNS [8] is proposed. A recent survey of exact and heuristic methods for solving MDVRP can be found in [10].

In this current paper, we proposes a new algorithm, inspired on the Adaptive Guided Variable Neighborhood Search algorithm from [2] and similar to the General VNS algorithm of [4], for solving MDVRP. The new algorithm uses an adaptive local search method rather than a local search based on the Randomized Variable Neighborhood Descent (RVND) method, as used in [4]. The idea behind VNS is that switching the neighborhood structure after the current neighborhood structure trapped in local optima may help VNS to escape from the local optima. Thus, applying different neighborhood structures can generate different search trajectories, which help in escaping from the current point as well as dealing with problems related to landscape changes that usually occur during the solving process. However, the sequence of the neighborhood structures in VNS has a critical impact on the algorithm performance, which is usually dependent on the problem and/or its instances. This implies that not only different instances require different sequences of the neighborhood structures but also different stages of the solving process [2]. In this scenario, we create a simply ranking to classify the neighborhoods that will be chosen for refining the current solution.

The remaining of this paper is organized as follows. Section 2 describes MDVRP. Section 3 presents the neighborhoods used for exploring the solution space of MDVRP. Section 4 introduces the proposed algorithm, named Variable Neighborhood Search with Adaptive Local Search (VNSALS). The calibration of the VNSALS parameters are described in Sect. 5, followed by the experimental results that are shown and discussed in Sect. 6. Finally, Sect. 7 concludes this work.

2 Multi-Depot Vehicle Routing Problem

The Multi-Depot Vehicle Routing Problem (MDVRP) consists in determining a set of vehicle routes such that [10]:

(i) each vehicle route starts and ends at the same depot;
(ii) each customer is serviced exactly once by a vehicle;
(iii) the total demand of each route does not exceed the vehicle capacity; and
(iv) the total cost of the distribution is minimized.

Figure 1 represents a MDVRP problem with thirteen customers, in the form $\mathcal{V}^{CST} = \{1, 2, \ldots, 13\}$ and two depots, named $\mathcal{V}^{DEP} = \{14, 15\}$. In this representation, depot 14 has two routes that serves customers 2, 3, 4, 5, 9 and 12; depot 15 has three routes, serving customers 1, 6, 7, 8, 10, 11 and 13.

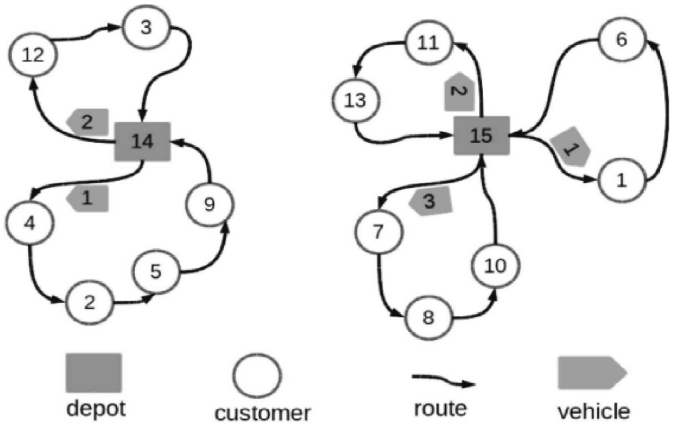


Fig. 1. Example of MDVRP with thirteen customers and two depots.

MDVRP is defined as follows [10]. Let $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ be a complete graph, where $\mathcal{V}$ is a set of nodes and $\mathcal{A}$ is a set of arcs. The set of nodes are partitioned into two subsets: the set of customers to be served, given by $\mathcal{V}^{CST} = \{1, 2, \ldots, N\}$, and the set of depots $\mathcal{V}^{DEP} = \{N + 1, N + 2, \ldots, N + M\}$, with $\mathcal{V} = \mathcal{V}^{DEP} \cup \mathcal{V}^{CST}$ and $\mathcal{V}^{DEP} \cap V^{CST} = \oslash$. There is a non-negative cost c_{ij} associated with each arc $(i, j) \in \mathcal{A}$. For each customer, there is a non-negative demand d_i and there is no demand at the depot nodes. In each depot, there are a fleet of K identical vehicles, each with capacity Q. The service time at each customer i is t_i, while the maximum route duration time is set to T. A conversion factor w_{ij} might be needed to transform the cost c_{ij} into time units. In this work, however, the cost is the same as the time and distance units, so $w_{ij} = 1$.

MDVRP consists in designing a set of vehicle routes serving all customers, such that the maximum number of vehicle per depot, vehicle-capacity and maximum duration time in the route are respected, and the total cost of transportation is minimized. The MDVRP mathematical formulation requires the definition of the binary decision variable x_{ijk}, which is equal to 1 when vehicle k visits node j immediately after node i, and 0 otherwise. Auxiliary binary variables y_i are also used in the subtour elimination constraints [10,11]. The mathematical model is given by:

$$\min \sum_{i=1}^{N+M} \sum_{j=1}^{N+M} \sum_{k=1}^{K} c_{ij} x_{ijk} \tag{1}$$

Subject to

$$\sum_{i=1}^{N+M} \sum_{k=1}^{K} x_{ijk} = 1 \ \ (j = 1, \ldots, N) \tag{2}$$

$$\sum_{j=1}^{N+M} \sum_{k=1}^{K} x_{ijk} = 1 \ \ (i = 1, \ldots, N) \tag{3}$$

$$\sum_{i=1}^{N+M} x_{ihk} - \sum_{j=1}^{N+M} x_{hjk} = 0 \ \ (k = 1, \ldots, K; \ h = 1, \ldots, N+M) \tag{4}$$

$$\sum_{i=1}^{N+M} \sum_{j=1}^{N+M} d_i x_{ijk} \leq Q \ \ (k = 1, \ldots, K) \tag{5}$$

$$\sum_{i=1}^{N+M} \sum_{j=1}^{N+M} (c_{ij} w_{ij} + t_i) x_{ijk} \leq T \ \ (k = 1, \ldots, K) \tag{6}$$

$$\sum_{i=N+1}^{N+M} \sum_{j=1}^{N} x_{ijk} \leq 1 \ \ (k = 1, \ldots, K) \tag{7}$$

$$\sum_{j=N+1}^{N+M} \sum_{i=1}^{N} x_{ijk} \leq 1 \ \ (k = 1, \ldots, K) \tag{8}$$

$$\begin{gathered} y_i - y_j + (N+M) x_{ijk} \leq N + M - 1 \\ \text{for } 1 \leq i \neq j \leq N \text{ and } 1 \leq k \leq K \end{gathered} \tag{9}$$

$$x_{ijk} \in \{0, 1\} \ \forall \ i, j, k \tag{10}$$

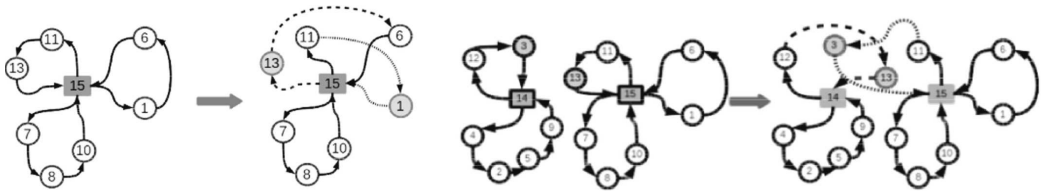
$$y_i \in \{0, 1\} \quad \forall \ i \tag{11}$$

In this formulation, the objective, shown in Expression 1, is to minimize the total cost. Constraints (2) and (3) guarantee that each customer is served by exactly one vehicle. Flow conservation is guaranteed through constraints (4). Constraints (5) and (6) refer to the vehicle capacity and the total route cost, respectively. Vehicle availability is verified by constraints (7) and (8). Subtour elimination is provided by constraints (9). Finally, constraints (10) and (11) define x and y as binary variables.

3 Neighborhoods for MDVRP

We used seven neighborhoods widely applied in the literature to explore the solution space of this problem: Swap(1, 1), Swap(2, 1), Shift(1, 0), Shift(2, 0), 2-Opt, 2-Opt* and Reverse [15,17,19]. These neighborhoods are described in the sequel.

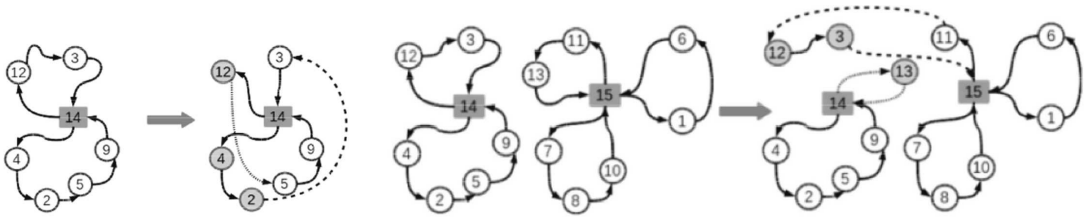
Swap(1, 1) - permutation between a customer v_j from a route r_k and a customer v_t from a route r_l. In Fig. 2(a), the clients 1 and 13 were swapped in the same depot. In Fig. 2(b), the client 13 from depot 14 and client 13 from depot 15 are swapped.



(a) Swap(1,1) in one depot. (b) Swap(1,1) in two different depots.

Fig. 2. Examples of neighborhood Swap(1, 1).

Swap(2, 1) - permutation of two adjacent customers v_j and v_{j+1} from a route r_k by a customer v_t from a route r_l. In Fig. 3(a), the adjacent clients 4 and 2 were exchanged with client 12. In Fig. 3(b), clients 12 and 3 from depot 15 were exchanged with client 13 belongs to the depot 14.

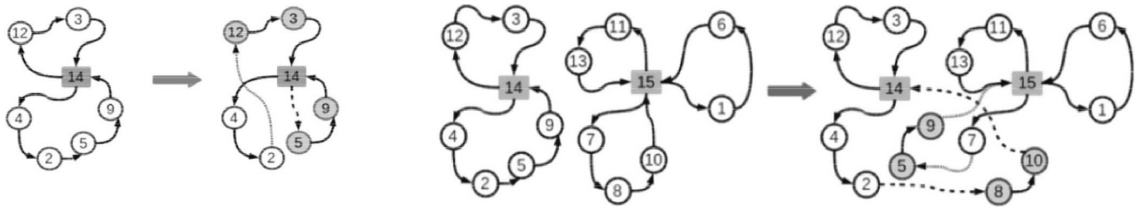


(a) Swap(2,1) in one depot. (b) Swap(2,1) in two different depots.

Fig. 3. Examples of Swap(2, 1) neighborhood.

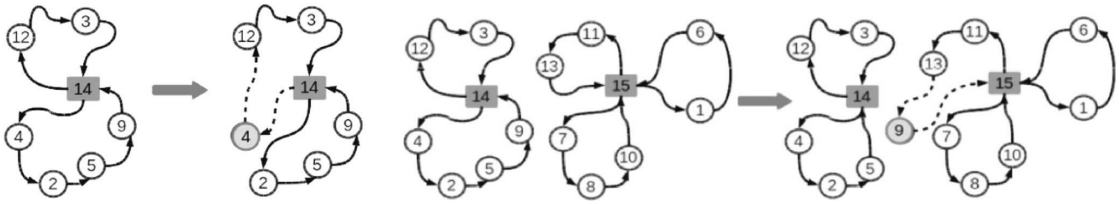
Swap(2, 2) - permutation between two adjacent customers v_j and v_{j+1} from a route r_k by another two adjacent customers v_t and v_{t+1}, $\forall v_t, v_{t+1} \in \mathcal{V}^{CST}$, belonging to a route r_l. In Fig. 4(a), the adjacent clients 5 and 9 were exchanged with client 12 and 3. In Fig. 4(b), clients 5 and 9 from depot 14 were exchanged with client 8 and 10 belonging to the depot 15.

Shift(1, 0) - transference of a customer v_j from a route r_k to a route r_l. In Fig. 5(a), the client 4 was moved from one route to the other one. The same situation occurs in Fig. 5(b) where client 9 was moved to the depot 15.



(a) Swap(2,2) in one depot. (b) Swap(2,2) in two different depots.

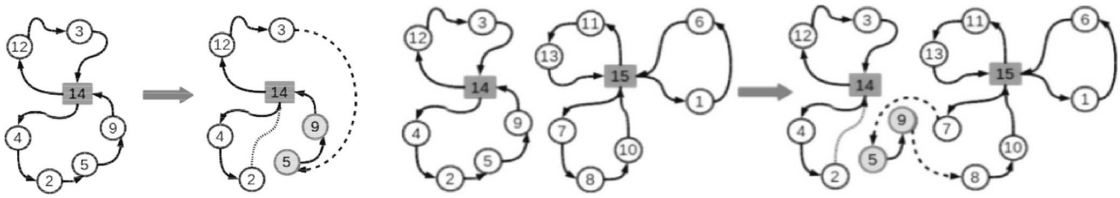
Fig. 4. Examples of Swap(2, 2) neighborhood.



(a) Shift(1,0) in one depot. (b) Shift(1,0) in two different depots.

Fig. 5. Examples of Shift(1, 0) neighborhood.

Shift(2, 0) - transference of two adjacent customers v_j and v_{j+1} from a route r_k to a route r_l. In Fig. 6(a), the adjacent clients 5 and 9 were moved from one route to the other one. The same occurs in Fig. 6(b), where clients 5 and 9 from depot 14 were moved to the depot 15.



(a) Shift(2,0) in one depot. (b) Shift(2,0) in two different depots.

Fig. 6. Examples of Shift(2, 0) neighborhood.

2-Opt - Two non-adjacent arcs are deleted and another two ones are added so that a new route is generated. In Fig. 7, the arcs $(4, 2)$ and $(9, 12)$ are deleted, while the arcs $(4, 9)$ and $(2, 12)$ are inserted, changing the sub-route to $(14, 4, 9, 5, 2, 12, 3, 14)$.

The 2-Opt* neighborhood is based on the deletion and reinsertion of two arc pairs from two different routes. This neighborhood is sometimes called crossover neighborhood [18]. For 2-Opt* in two distinct routes r_1 and r_2, let u and v be arcs from r_1 and v and y be arcs from r_2. There are two alternatives, as follows:

– *Alternative 1*: replace (u, x) and (v, y) by (u, v) and (x, y);
– *Alternative 2*: replace (u, x) and (v, y) by (u, y) and (x, v).

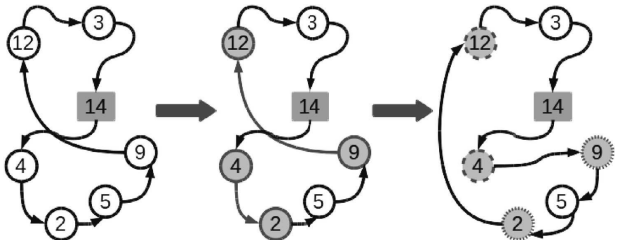
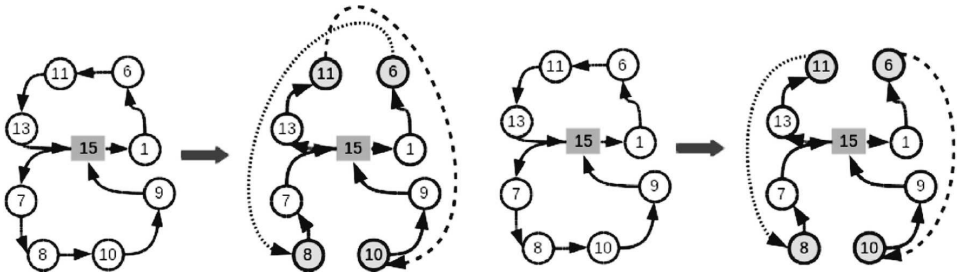


Fig. 7. Example of neighborhood 2-Opt.



(a) 2-Opt* (Alternative 1). (b) 2-Opt* (Alternative 2).

Fig. 8. Examples of 2-Opt* neighborhoods.

Figure 8 represents 2-Opt* cases.

Reverse - This move reverses the route direction. It is represented in Fig. 9.

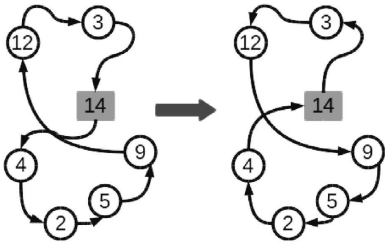


Fig. 9. Example of Reverse neighborhood.

The Swap(1, 1), Swap(2, 1), Shift(1, 0) and Shift(2, 0) neighborhoods may be in route (intra-route), between different routes (inter-route) and occur in the same depot (intra-depot) or between different depots (inter-depots); 2-Opt and Reverse may occur in intra-depot/intra-route; and 2-Opt* may occur in intra-depot/inter-route.

4 Description of the Proposed Algorithm

In this work we proposed an algorithm, named Variable Neighborhood Search with Adaptive Local Search (VNSALS), for solving MDVRP. VNSALS is

inspired in the Adaptive Guided Variable Neighborhood Search algorithm from [2] and in the GVNS algorithm from [4].

VNSALS is a VNS-based algorithm [8] that utilizes some problem-specific knowledge and uses an adaptive learning mechanism to find the most suitable neighborhood structure during the searching process. In the adaptive learning part, the algorithm memorizes the neighborhood structure that made an improvement in the solution quality [3]. The pseudo-code of VNSALS is described in Algorithm 1.

Algorithm 1. VNSALS (*iterMax, maxTime, maxLevel, alsTraining, alsReset, Nshake*)

```
1: Let s be an initial solution;
2: p ← 1;                                   // Initial value of perturbation level
3: k ← 1;                                   // Initial neighborhood
4: iter ← 0; itAdaptative ← 0; s' ← s
5: // Stopping criterion
6: while iter < IterMax or t < maxTime do
7:    if itAdaptive < (alsTraining * IterMax) then
8:       s'' ← RandomNeighborhood(s', success, N);
9:    else
10:      s'' ← ALS(s', success, N);
11:   end if
12:   if f(s'') < f(s) then
13:      s ← s'';
14:      k ← 1;                              // Return to the first neighborhood
15:      p ← 1;                              // Return to the first level
16:      iter ← 0;
17:   else
18:      p ← p + 1;                          // Change perturbation level
19:      iter ← iter + 1;
20:   end if
21:   if p > maxLevel then
22:      k ← k + 1;                          // Change neighborhood
23:      p ← 1;                              // Return to the first level
24:   end if
25:   if k > Nshake then
26:      k ← 1; p ← 1;          // Reset neighborhood and level
27:   end if
28:   itAdaptive ← itAdaptive + 1
29:   if itAdaptive >= (alsReset * IterMax) then
30:      resetRankNeighborhood();
31:      itAdaptive ← 0;
32:   end if
33:   s' ← Perturbation(s, k, p, N);
34:   s' ← Split(s');
35:   s' ← ClusterRefinement(s');
36:   s' ← Split(s');
37: end while
38: return s;
```

A solution s of MDRVP is represented by a list vector, inspired and adapted from [14]. Each position of this vector indicates a depot and each list indicates the visit routes to be performed by vehicles from that depot.

As an example, let $\mathcal{V}^{DEP} = \{14, 15\}$ and $\mathcal{V}^{CST} = \{1, 2, \ldots, 13\}$ be a set of two depots and thirteen customers, respectively, and $\mathcal{R}_{DEP}$ be the set of routes per depot. The solution represented in Fig. 1 is $\mathcal{R}_{14} = \{r_1, r_2\}$, where $r_1 = [\mathbf{14}\ 4\ 2\ 5\ 9\ \mathbf{14}\ 12\ 3\ \mathbf{14}]$ and $r_2 = [\mathbf{15}\ 2\ 6\ \mathbf{15}\ 11\ 13\ \mathbf{15}\ 7\ 8\ 10\ \mathbf{15}]$.

In this solution, there are two routes in the depot 14. At the first one, the vehicle leaves the depot and visits the customers 4, 2, 5 and 9, in this order, and then returns to the depot. In the second route, a vehicle leaves the depot, visits the customers 12 and 3 and returns to the depot. In the depot 15 there are 3 routes, which are represented in the vector r_2.

The strategy used for generating an initial solution (line 1 of Algorithm 1) is "*cluster first and then route*". In this way, the customers are assigned to the depots in a balanced way using the Gillet and Johnson algorithm [7]. Initially, the heuristic determines the distance between each consumer $j \in \mathcal{V}^{CST}$ not yet assigned to the two nearest depots a_1 and $a_2 \in \mathcal{V}^{DEP}$, with rate $vr_j = a_1/a_2$, $\forall\ j = N+1, N+2, \cdots, N+M$. Then, this rate is sorted ascending according to the values vr_j, assigning the consumer to the nearest feasible depot. This process repeats until all customers are allocated in only one depot. In order to generate a feasible solution, the number of vehicles can be greater than K in this phase. After this construction phase, the VNSALS algorithm tries to refine this constructed solution.

The main procedures of the VNSALS algorithm are *Perturbation*, *Cluster-Refinement* and the Adaptive Local Search (ALS), introduced at lines 33, 35, 8 and 10 of Algorithm 1, respectively. These procedures are necessary because only search in neighborhoods are insufficient to lead to an optimal solution [19].

Perturbation: In the perturbation phase (showed in Algorithm 2), the solution s undergoes a shake move in a given neighborhood $\mathcal{N}_k(s)$, generating a new solution s'. The neighborhood moves are applied in different depots (inter-depot), where the routes, for example, r_1 belong to d_1 and r_2 to d_2 and the number of moves is represented by *Nshake*. The moves are applied in this order: Shift(1, 0), Swap(2, 1), Shift(2, 0), Swap(2, 2), Swap(1, 1).

In order to generate a shake move in a solution s belonging to the neighborhood $\mathcal{N}_k$, each move in the *Perturbation* phase is applied p times, at most, where the value of p is a random integer between 1 and *maxLevel* (see line 33 of Algorithm 1).

Algorithm 2. Perturbation (s, k, p, $\mathcal{N}$)

1: **for** $(i = 1; i \leq p; i++)$ **do**
2: Generate the neighbor $s' \in \mathcal{N}_k(s)$;
3: **end for**
4: **return** s';

ClusterRefinement: The aim is to reduce the total distance between customers and depot assigned. After criteria are established, the idea is select the more expensive customer in depot $\mathcal{A}$ and move it to another generic depot $\mathcal{F}$. We create three criteria to select the customer, where $dist_{\mathcal{A},i}$ is the distance from depot $\mathcal{A}$ to customer i; d_i and t_i are the demand and service time of the customer i, respectively; $\mathcal{S}_1$ the set of customers assigned to depot $\mathcal{A}$; and $\mathcal{S}_\mathcal{F}$ the set of customers assigned to depot $\mathcal{F}$:

(i) Let θ_1 the cost of customer i given by $\theta_1 = \max\{\alpha \times dist_{\mathcal{A},i} + \beta \times d_i \,|\, i \in S_1\}$ and π_1 given by $\pi_1 = \min\{\alpha \times dist_{\mathcal{F},i} + \beta \times d_i \,|\, i \in S_\mathcal{F}, \mathcal{F} \in \mathcal{V}^{DEP}\}$. Then select the depot $\mathcal{F}$ given by π_1 and move the customer i to depot $\mathcal{F}$.

(ii) Let θ_2 be the cost given by $\theta_2 = (\sum_{i=1}^{|\mathcal{S}_1|}(\alpha \times dist_{\mathcal{A},i} + \beta \times d_i))/|\mathcal{S}_1|$. Select all customers $\in \mathcal{S}_1$ with cost $\theta_1 > \theta_2$ and save them in $\mathcal{S}_3$. For any customer in $\mathcal{S}_3$, find a depot $\mathcal{F} \in \mathcal{V}^{\mathcal{DEP}}$, with cost π_1 and move the selected customer to depot $\mathcal{F}$;

(iii) Find the most expensive customer $i \in \mathcal{S}_1$ given by $\theta_3 = \max\{\alpha \times dist_{\mathcal{A},i} + \beta \times d_i\}/c_\mathcal{A}$, where $c_\mathcal{A}$ is the cost of depot $\mathcal{A}$, according to Eq. (1). Let $c_\mathcal{F}$ be the cost of depot $\mathcal{F}$ and $\pi_2 = (\pi_1/c_\mathcal{F})$. Then select the depot $\mathcal{F}$ given by π_2 and move the customer i to depot $\mathcal{F}$.

For any execution (line 35 in Algorithm 1) only one of these criteria is randomly selected and applied in solution. In this work, we considered $\alpha = 1$ and $\beta = 1$.

Local Search: The local search is used to refine the solution by neighborhoods described in Sect. 3. Only one neighborhood is applied in s'. During the training, each neighborhood is selected randomly (Algorithm 3) and always that its local search improves, the value associated with the i-th neighborhood is updated (accumulated). In vector *success* is accumulated the improvement of any neighborhood. For example, let $\mathcal{N}_1$, $\mathcal{N}_2$ and $\mathcal{N}_3$ be neighborhood structures. Consider that during the training $\mathcal{N}_1$ improved the solution three times, $\mathcal{N}_2$ five times and $\mathcal{N}_3$ two times. After the training (Algorithm 1, line 10), the probabilities of neighborhoods are $\{3/10, 5/10, 2/10\} = \{0.3, 0.5, 0.2\}$, respectively. In this phase, roulette wheel selection is used to select the neighborhood which will be applied in the current solution (Algorithm 4). After any times (*alsReset*), the vector *success* is reset at line 30 of Algorithm 1 (*resetRankNeighborhood*()) and the training process is restarted.

Algorithm 3. RandomNeighborhood(s, *success*, $\mathcal{N}$)

1: $i \leftarrow rand(|\mathcal{N}|)$; {Select a neighborhood $\mathcal{N}_i \in \mathcal{N}$}
2: $s' \leftarrow LocalSearch(\mathcal{N}_i, s)$; {Apply local search to neighborhood $\mathcal{N}_i$}
3: **if** $f(s') < f(s)$ **then**
4: $\quad s \leftarrow s'$;
5: $\quad success(i) \leftarrow success(i) + 1$; {Increment success vector associated with $\mathcal{N}_i$}
6: **end if**
7: **return** $s, success$;

Algorithm 4. ALS(s, *success*, $\mathcal{N}$)

1: $i \leftarrow rouletteWheel(success)$; {Select a index of neighborhood}
2: $s' \leftarrow LocalSearch(\mathcal{N}_i, s)$; {Apply local search to the neighborhood selected}
3: **if** $f(s') < f(s)$ **then**
4: $\quad s \leftarrow s'$;
5: **end if**
6: **return** s;

Split Algorithm: After perturbation, the solution is represented by a giant tour from each depot, without route delimiters. It is basically a single sequence made of all customers assigned to a depot. For example, the sequence for depot 14 in Fig. 1 is $\{4, 2, 5, 9, 12, 3\}$. Individual routes are created from this giant tour with the Split algorithm [13], which can optimally extract feasible routes from a single sequence. After Split algorithm, we apply Reverse neighborhood, because it can improve the solution [19].

If the solution s'' returned by the local search is better than the current solution s, then s is updated (line 13 of Algorithm 1) and the perturbation level returns to its minimum value (line15 of Algorithm 1). Otherwise, the perturbation level is increased (line 18 of Algorithm 1). If the perturbation level exceeds the level value (*maxLevel*), then the search moves to the next neighborhood (line 22 of Algorithm 1) and the perturbation level returns to the minimum level (line 23 of Algorithm 1).

5 Parameter Tuning

The VNSALS algorithm has six parameters to be tuned: (i) maximum number of iterations (*iterMax*); (ii) maximum time of processing (*maxTime*); (iii) number of neighborhoods for shaking (*Nshake*); (iv) maximum level of perturbations (*maxLevel*); (v) number for training (*alsTraining*) and (vi) reset adaptive (*alsReset*). To make a fare calibration of the parameters we used an automated algorithm, called IRACE (Iterated Racing for Automatic Algorithm Configuration) [9]. This algorithm was designed to provide the most appropriate parameters for an optimization algorithm and a set of instances. IRACE runs as a package of the `R` software, that is a free environment for statistical computing and graphics.

Table 1. Grouping of instances.

Group	1													2								
Instance	p01	p02	p03	p04	p05	p06	p07	p12	p13	p14	p15	p16	p17	p08	p09	p10	p11	p18	p19	p20	p21	p22	p23
n	50	50	75	100	100	100	100	80	80	80	160	160	160	249	249	249	249	240	240	240	360	360	360
m	4	2	3	8	5	6	4	5	5	5	5	5	5	14	12	8	6	5	5	5	5	5	5
d	4	4	5	2	2	3	4	2	2	2	4	4	4	2	3	4	5	6	6	6	9	9	9
β	800	400	1125	1600	1000	1800	1600	800	800	800	3200	3200	3200	6972	8964	7968	7470	7200	7200	7200	16200	16200	16200

In order to calibrate the parameters of the proposed algorithm, the instances were grouped in sets according to theirs sizes, determined by the value

$\gamma = n \times m \times d$. Table 1 shows two groups of instances with $\gamma \leq 3200$ and $\gamma > 3200$. A set of instances of each group was chosen for testing by IRACE software. The following instances were chosen: Group 1 (p02, p04, p06, p14, p17); Group 2 (p09, p19 and p21). We set the following values for the parameters: $iterMax = \{400, 450, 500\}$; $Nshake = \{3, 4, 5\}$; $maxLevel = \{3, 4, 5\}$; $alsTraining = \{0.1, 0.15\}$; $alsReset = \{0.2, 0.3, 0.4\}$ and $maxTime = \{30, 60\}$. The best values returned by IRACE were: $iterMax = 500$, $Nshake = 5$, $maxLevel = 3$, $alsTraining = 0.15$, $alsReset = 0.3$ and $maxTime = 60$ min.

6 Computational Experiments

The VNSALS algorithm was coded in C++ and tested in a computer with Intel Core i5-2310M, 2.90 GHz, 4 GB RAM, under operational system Ubuntu 16.04 64 bits and compiler `G++` version 5.4.

The Courdeau's instances of MDVRP from [6] were used to verify the performance of the VNSALS algorithm. These instances have $n = 50$ to $n = 360$ customers; $d = 2$ to $d = 9$ depots; $m = 2$ to $m = 14$ vehicles; and load $q = 60$ to $q = 500$.

Six algorithms from literature were used for comparing the results of the VNSALS algorithm. These algorithms are: ALNS [12], HGSADC [17], ILS-RVND-SA [16], HGSADC+ [19], CoES [11], and GVNS [4]. The computer configurations used to test these algorithms, as well as the number of runs of each algorithm in each instance, are shown in Table 2.

Table 2. Computer configurations used to test the algorithms.

Algorithm	Runs	Computer
ALNS	10	Pentium IV 3.0 GHz
HGSADC	10	Opteron 2.4 GHz scaled for a Pentium IV 3.0 GHz
ILS+RVND+SA	10	Intel CoreTM i7 with 2.93 GHZ and 8 GB of RAM
GRASPxILS	5	Intel CoreTM 2 Quad CPU Q8400 @ 2.66 GHz
HGSADC+	10	Opteron 2.4 GHz scaled for a Pentium IV 3.0 GHz
GVNS	30	Intel Core i3-2370M, 2.40 GHz, 4GB RAM
VNSALS (our algorithm)	30	Intel Core i5-2310M, 2.90 GHz, 4GB RAM

Table 3 presents the best results found by the algorithms ALNS, HGSADC, ILS-RVND-SA, HGSADC+, CoES, GVNS, shown in their respective original articles, and the results concerning the application of the proposed VNSALS algorithm. For VNSALS, the average values, the best solutions, as well as the execution times, are presented. Each instance was run 30 times, using the parameter values chosen by IRACE (which are described in Sect. 5) and the parameter $maxTime = 60$ min. The average results and the best results of VNSALS are 1.88% and 0.99%, respectively, higher in relation to the values of the Best Known Solutions (BKS). Compared to the HGSADC+ algorithm, which has the best

results for MDVRP, VNSALS presents a total cost 1.85% and 0.96% higher in relation to the average and best values, respectively. These results show that VNSALS is a competitive algorithm against the best algorithms for the MDVRP solution in the literature.

Table 3. Results of the algorithms.

Inst	*n*	*m*	*d*	*q*	*T*	**BKS**	ALNS	HGSADC	ILS-RVND-SA	HGSADC+	CoES	GVNS	VNSALS		T(min)
													Average	Best	Average
p01	50	4	4	80	∞	**576.87**	576.87	576.87	576.87	576.87	576.87	591.09	579.12	576.87	0.37
p02	50	2	4	160	∞	**473.53**	473.53	473.53	473.53	473.53	475.06	476.66	476.72	473.53	0.40
p03	75	3	5	140	∞	**640.65**	641.19	641.19	641.19	640.65	643.57	641.19	642.81	641.19	1.01
p04	100	8	2	100	∞	**999.21**	1006.09	1001.04	1001.04	1000.66	1011.42	1025.44	1017.53	1003.72	10.69
p05	100	5	2	200	∞	**750.03**	752.34	750.03	750.21	750.03	752.39	757.46	756.55	751.15	6.85
p06	100	6	3	100	∞	**876.50**	883.01	876.5	876.5	876.5	877.86	889.79	888.23	880.42	5.17
p07	100	4	4	100	∞	**881.97**	889.36	884.43	881.97	881.97	893.36	898.31	895.08	884.04	4.88
p08	249	14	2	500	310	**4372.78**	4421.03	4397.42	4393.7	4383.63	4474.23		4577.46	4503.76	60.29
p09	249	12	3	500	310	**3858.66**	3892.5	3868.59	3864.22	3860.77	3904.92		3987.11	3928.40	60.13
p10	249	8	4	500	310	**3631.11**	3666.85	3636.09	3634.72	3631.71	3680.02		3774.47	3702.20	59.17
p11	249	6	5	500	310	**3546.06**	3573.23	3548.25	3546.15	3547.37	3593.37		3627.77	3567.71	55.67
p12	80	5	2	60	∞	**1318.95**	1319.13	1318.95	1318.95	1318.95	1318.95	1326.85	1321.46	1318.95	1.96
p13	80	5	2	60	200	**1318.95**	1318.95	1318.95	1318.95	1318.95	1318.95		1323.61	1318.95	0.75
p14	80	5	2	60	180	**1360.12**	1360.12	1360.12	1360.12	1360.12	1360.12		1363.09	1360.12	0.58
p15	160	5	4	60	∞	**2505.42**	2519.64	2505.42	2505.42	2505.42	2549.65	2567.62	2551.49	2505.42	29.31
p16	160	5	4	60	200	**2572.23**	2573.95	2572.23	2572.23	2572.23	2572.23		2590.57	2572.23	4.12
p17	160	5	4	60	180	**2709.09**	2709.09	2709.09	2710.21	2709.09	2733.8		2720.61	2709.09	2.28
p18	240	5	6	60	∞	**3702.85**	3736.53	3702.85	3702.85	3702.85	3781.66	3796.04	3797.59	3762.64	60.16
p19	240	5	6	60	200	**3827.06**	3838.76	3827.06	3827.55	3827.06	3827.06		3851.20	3839.36	14.62
p20	240	5	6	60	180	**4058.07**	4064.76	4058.07	4058.07	4058.07	4094.86		4080.18	4069.21	6.81
p21	360	5	9	60	∞	**5474.84**	5501.58	5476.41	5474.84	5474.84	5668.97		5711.17	5669.61	60.61
p22	360	5	9	60	200	**5702.16**	5722.19	5702.16	5705.84	5702.16	5708.78		5736.91	5714.45	46.53
p23	360	5	9	60	180	**6078.75**	6092.66	6078.75	6078.75	6080.43	6159.9		6116.5	6089.9	20.6
					Total cost	**61235.86**	61533.36	61284.00	61273.88	61253.86	61978.00		62387.20	61842.91	

Table 4 compares the average gaps of the ALNS, HGSADC, ILS-RVND-SA, HGSADC+, CoES, GVNS and VNSALS algorithms. A blank line means that the respective algorithm was not tested in the respective instance. In this table, the average gap is calculated by Eq. (12):

$$gap_i^{\text{avg}} = \frac{\overline{f}_i^{\text{VNSALS}} - f_i^{\star}}{f_i^{\star}} \tag{12}$$

where $f_i^{\star}$ represents the value of the best known solution (BKS) from literature relative to instance i and $\overline{f}_i^{\text{VNSALS}}$ represents the average value produced by the VNSALS algorithm in this instance. From Table 4, we may observed that the best results achieved by the VNSALS algorithm are close to the best known values. In terms of variability of the final solutions, the gap varies from 0.19% to 4.68%, with average gap of 1.49%. The VNSALS results are better than those of the GVNS algorithm, which is a VNS-based algorithm without the ALS procedure and the cluster refinement.

Table 4. Comparison of the algorithms with respect to the average gaps

Inst	ALNS	HGSADC	ILS-RVND-SA	HGSADC+	CoES	GVNS	VNSALS
p01	0.00	0.00	0.00	0.00	0.00	2.47	0.39
p02	0.00	0.00	0.00	0.00	0.32	0.66	0.67
p03	0.08	0.08	0.08	0.00	0.46	0.08	0.34
p04	0.69	0.18	0.18	0.15	1.22	2.63	1.83
p05	0.31	0.00	0.02	0.00	0.31	0.99	0.87
p06	0.74	0.00	0.00	0.00	0.16	1.52	1.34
p07	0.84	0.28	0.00	0.00	1.29	1.85	1.49
p08	1.10	0.56	0.48	0.25	2.32		4.68
p09	0.88	0.26	0.14	0.05	1.20		3.33
p10	0.98	0.14	0.10	0.02	1.35		3.95
p11	0.77	0.06	0.00	0.04	1.33		2.30
p12	0.01	0.00	0.00	0.00	0.00	0.60	0.19
p13	0.00	0.00	0.00	0.00	0.00		0.35
p14	0.00	0.00	0.00	0.00	0.00		0.22
p15	0.57	0.00	0.00	0.00	1.77	2.48	1.84
p16	0.07	0.00	0.00	0.00	0.00		0.71
p17	0.00	0.00	0.04	0.00	0.91		0.43
p18	0.91	0.00	0.00	0.00	2.13	2.52	2.56
p19	0.31	0.00	0.01	0.00	0.00		0.63
p20	0.16	0.00	0.00	0.00	0.91		0.54
p21	0.49	0.03	0.00	0.00	3.55		4.32
p22	0.35	0.00	0.06	0.00	0.12		0.61
p23	0.23	0.00	0.00	0.03	1.33		0.62
Average	0.41	0.07	0.05	0.02	0.90		1.49

7 Conclusions

In this paper, we presented a VNS-based algorithm, named VNSALS (Variable Neighborhood Search with Adaptive Local Search), for solving the Multi-Depot Vehicle Routing Problem (MDVRP). The main characteristic of VNSALS is the existence of an Adaptive Local Search (ALS) algorithm that classifies the neighborhoods and performs cluster refinement, in order to minimize the cost associated with depots. In the ALS procedure, the neighborhood order is not a parameter, because it is chosen adaptively by roulette wheel.

VNSALS was tested in classical instances of MDVRP and its results were compared with those of the best known values and other six algorithms of the literature.

The computational experiments showed that the VNSALS algorithm is better than the other VNS-based algorithm of [4]. However, the proposed algorithm was not better than the other algorithms with which it was compared. New improvements need to be made to improve its performance.

Acknowledgements. The authors would like to thank the CAPES Foundation, the Brazilian Council of Technological and Scientific Development (CNPq), the Minas Gerais State Research Foundation (FAPEMIG), the Federal Center of Technological Education of Minas Gerais (CEFET-MG), and the Federal University of Ouro Preto (UFOP) for supporting this research.

References

1. Allahyari, S., Salari, M., Vigo, D.: A hybrid metaheuristic algorithm for the multi-depot covering tour vehicle routing problem. Eur. J. Oper. Res. **242**(3), 756–768 (2015)
2. Aziz, R.A., Ayob, M., Othman, Z., Ahmad, Z., Sabar, N.R.: An adaptive guided variable neighborhood search based on honey-bee mating optimization algorithm for the course timetabling problem. Soft Comput. **21**(22), 6755–6765 (2017)
3. Aziz, R., Ayob, M., Othman, Z., Sarim, H.: Adaptive guided variable neighborhood search. J. Appl. Sci. **13**(6), 883–888 (2013)
4. Bezerra, S.N., de Souza, S.R., Souza, M.J.F.: A GVNS algorithm for solving the multi-depot vehicle routing problem. Electron. Notes Discrete Math. **66**, 167–174 (2018). 5th International Conference on Variable Neighborhood Search
5. Contardo, C., Martinelli, R.: A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. Discrete Optim. **12**, 129–146 (2014)
6. Cordeau, J.F., Gendreau, M., Laporte, G.: A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks **30**(2), 105–119 (1997)
7. Gillett, B.E., Johnson, J.G.: Multi-terminal vehicle-dispatch algorithm. Omega **4**(6), 711–718 (1976)
8. Hansen, P., Mladenović, N., Moreno Pérez, J.A.: Variable neighbourhood search: methods and applications. 4OR **6**(4), 319–360 (2008)
9. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The IRACE package: iterated racing for automatic algorithm configuration. Oper. Res. Perspect. **3**, 43–58 (2016)
10. Montoya-Torres, J.R., Franco, J.L., Isaza, S.N., Jiménez, H.F., Herazo-Padilla, N.: A literature review on the vehicle routing problem with multiple depots. Comput. Ind. Eng. **79**, 115–129 (2015)
11. de Oliveira, F.B., Enayatifar, R., Sadaei, H.J., Guimarães, F.G., Potvin, J.Y.: A cooperative coevolutionary algorithm for the multi-depot vehicle routing problem. Expert Syst. Appl. **43**, 117–130 (2016)
12. Pisinger, D., Ropke, S.: A general heuristic for vehicle routing problems. Comput. Oper. Res. **34**(8), 2403–2435 (2007)
13. Prins, C.: A simple and effective evolutionary algorithm for the vehicle routing problem. Comput. Oper. Res. **31**(12), 1985–2002 (2004)
14. Salhi, S., Imran, A., Wassan, N.A.: The multi-depot vehicle routing problem with heterogeneous vehicle fleet: formulation and a variable neighborhood search implementation. Comput. Oper. Res. **52**(PB), 315–325 (2014)
15. Subramanian, A., Drummond, L., Bentes, C., Ochi, L., Farias, R.: A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. Comput. Oper. Res. **37**(11), 1899–1911 (2010)
16. Subramanian, A., Uchoa, E., Ochi, L.S.: A hybrid algorithm for a class of vehicle routing problems. Comput. Oper. Res. **40**(10), 2519–2531 (2013)
17. Vidal, T., Crainic, T.G., Gendreau, M., Lahrichi, N., Rei, W.: A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. Oper. Res. **60**(3), 611–624 (2012)
18. Vidal, T., Crainic, T.G., Gendreau, M., Prins, C.: Heuristics for multi-attribute vehicle routing problems: a survey and synthesis. Eur. J. Oper. Res. **231**(1), 1–21 (2013)
19. Vidal, T., Crainic, T.G., Gendreau, M., Prins, C.: Implicit depot assignments and rotations in vehicle routing heuristics. Eur. J. Oper. Res. **237**(1), 15–28 (2014)