Annals of Operations Research manuscript No.
(will be inserted by the editor)

The Multi-Depot Heterogeneous VRP with Backhauls: Formulation and a Hybrid VNS with GRAMPS Meta-heuristic Approach

Fatih Kocatürk · G. Yazgı Tütüncü · Said Salhi

Received: date / Accepted: date

Abstract In this paper, we investigate the Multi-Depot Heterogeneous VRP with Backhauls. Though the problem is a generalisation of three existing routing problems, this is the first time this combined routing problem is investigated. A mathematical formulation is first presented followed by some tightening. A powerful and novel hybridisation of Variable Neighbourhood Search (VNS) with the Greedy Randomized Adaptive Memory Programming Search is proposed. As there are no problem instances available for bench-marking and evaluation purposes, we generated data sets by combining those from existing vehicle routing problems. The proposed meta-heuristic obtains a number of optimal solutions for small instances and yields about 13% gap from the lower bounds compared to nearly 40% and 20% average gap values for our CPLEX implementation and the VNS without hybridisation, respectively.

Keywords Routing · Heterogeneous vehicle fleet · Backhauling · Multiple depots · GRAMPS and VNS hybridisation

1 Introduction & Literature Review

In this paper, we study a logistical problem that is commonly faced in real-life logistic systems and which integrates three complex but related routing problems. These include the multi-depot vehicle routing problem, the heterogeneous vehicle

Fatih Kocatürk
Department of Mathematics, İzmir University of Economics, Sakarya Cad. 156, Balçova, İzmir, Turkey
E-mail: fatih.kocaturk@std.ieu.edu.tr

G. Yazgı Tütüncü
Department of Mathematics, İzmir University of Economics, Sakarya Cad. 156, Balçova, İzmir, Turkey
IESEG School of Management, 3 Rue de la Digue,59000, Lille, France

Said Salhi
Kent Business School, University of Kent, Canterbury, Kent, CT2 7FS, UK

fleet problem and the routing problem with backhauls. We refer to this integrated routing problem as the (MDHFVRPB) for short.

We briefly introduce these three individual but related routing problems and provide recent reviews for further reading. The classical vehicle routing problem (VRP) consists of a set of customers with their respective demand, a depot as the supply center and a fleet of vehicles having the same capacity. In real life, VRP and its variants contain additional constraints and challenges beyond the classical VRP such as multiple depots, heterogeneous fleet and pickup customers. For example, within reverse logistics, efficient solution methods for the VRP with Deliveries and Pickups (VRPDP) contributes considerably to reducing waste in terms of time and energy consumption resulting in a reduction of CO_2 emission and consequently health benefits.

VRPDP is an extension of the classical VRP. Here, a vehicle picks up a predefined amount of products from customers besides delivering some other products and transports these delivered products to the depot. Most of the researchers assumed that vehicles can visit pickup (backhaul) customers after visiting all delivery (linehaul) customers (Nagy and Salhi (2005)). They showed the difficulty of arranging the picked up and delivery goods in the vehicle while visiting. There are two different cases when this assumption is relaxed: Simultaneous Pickups and Deliveries (SPD), Mixed Pickups and Deliveries (MPD). In the former, customers can receive and dispatch goods at the same time (in one visit) whereas in the latter, customers are either delivery or pickup locations but not both. The VRPDP can be divided into three categories, namely, (i) SPD, (ii) MPD and (iii) Deliver First Pickup Second (DFPS) (Salhi and Nagy (1999)). When the MPD and DFPS are combined, the problem is called the VRP with Backhauls (VRPB).

Heterogeneous Fixed Fleet VRP (HFFVRP) which is initiated by Taillard (1999) can be defined as a special case of the Heterogeneous Vehicle Fleet VRP (HVFVRP) with the addition that the number of vehicles in each type is fixed. In other words, the HFFVRP aims to find the best routes for the given vehicles, while HVFVRP aims to find the best vehicle fleet combination.

There are few studies about Multi-Depot VRP with Backhauls (MDVRPB). Salhi and Nagy (1999) developed an insertion based heuristic that uses cluster-insertion method for VRPB and adapted this heuristic to the multi-depot problem. They also analyzed SPD and MPD versions of VRPDP. Nagy and Salhi (2005) developed an effective compound heuristic approach for the VRPDP with SPD and MPD versions and applied this heuristic to the multi-depot problem. Li et al. (2015) proposed an iterated local search method for the Multi-Depot VRPDP (MDVRPDP) with simultaneous pickup and delivery approach. Irnich (2000) introduced the multi-depot pickup and delivery problem with a single-hub and heterogeneous vehicles which is a special case of the MDVRPDP. This problem differs from MDVRPDP as the pickup requests are first collected to the hub location, then delivery requests are then dispatched from the hub by a vehicle, of a given heterogeneous fleet, departed from one of the request locations. In addition, every request location served as depots of vehicles and all vehicles starting at a location have to return to the same location at the end of the planning period. It is also worth noting that in their study their primary concern is the assignment of requests to vehicles rather than the routing itself as the trips are short due to narrow time windows and large quantities to deliver as they base their experiments on a real life case study. That is why they opted for a set covering type formulation.

In brief, their problem does not have the same structure of MDVRPDP in terms of depot definition.

The reader will find the review paper on the VRPDP by Berbeglia et al. (2007), and the recent reviews on the VRPB by Koç and Laporte (2018) and the SPD by Koç et al. (2020) to be useful, informative and complementary.

The only work that is closer to ours is the recent study by Penna et al. (2019). In their study, they addressed a family of rich VRPs including the use of heterogeneous fleet with other attributes such as backhauls, multiple depots, among others. Although, they proposed a unified algorithm that is capable of solving VRPs having some extensions, they did not introduce a mathematical model of these MDHFVRPB extensions and also did not provide any data set to test the proposed algorithm on MDHFVRPB. They used a two phase approach where in phase one a pool of promising routes are constructed using an Iterated Local Search with a Randomized Variable Neighbourhood Decent. Phase two uses this set of routes to solve a corresponding set partitioning problem with a commercial solver. Our study differs from theirs in producing a formal mathematical formulation and also in the construction of the initial solution, the use of adaptive learning and the VNS as will be shown in the subsequent sections.

To the best of our knowledge, this is the first study that integrates the MDVRPB and heterogeneous vehicle fleet which we refer to as the MDHFVRPB. Moreover, this is the first time where VNS and GRAMPS meta-heuristics are efficiently hybridised.

The contribution of the study is four-folds;

(i) The multi-depot routing problem with backhauls and heterogeneous vehicles is studied,
(ii) A new formulation is proposed which is then enhanced by introducing tightening,
(iii) A novel hybridisation of VNS and GRAMPS adopting a two stage approach is developed,
(iv) New data sets, based on the commonly used instances from related routing problems, are generated and interesting results obtained for comparison and benchmarking purposes.

The rest of the paper is organized as follows: In Section 2, a mathematical model with its enhanced version are given. The VNS-GRAMPS algorithm is presented in Section 3 and the explanation of some of the steps are given 4. The generation of the problem instances and corresponding computational results are reported and analysed in Section 5. Finally, we conclude the paper and highlight some research avenues in Section 6.

2 Mathematical Model

We first provide an overview of the problem and the necessary notation. The corresponding mathematical formulation is presented followed by some tightening.

2.1 Overview and Notation

In this section, the Multi-Depot Heterogeneous VRP with Backhauls (MDHFVRPB) is modelled by combining earlier formulations for the Fleet Size and Mix VRP with Backhauls proposed by Salhi et al. (2013) and the MDHFVRP also presented by Salhi and Sari (1997). The properties of the MDHFVRPB are summarized as follows: Customers are divided into two groups, namely, delivery (linehaul) and pickup (backhaul) customers. There are more than one depot in the system and there is a heterogeneous vehicle fleet (unlimited number of vehicles in each type) with fixed and variable costs varying according to the vehicle type. Backhaul customers cannot be visited unless all linehaul customers are visited. While a route consisting of only backhaul customers is not allowed, a route may include linehaul customers only if necessary. The vehicle capacity constraint is imposed.

Parameters:

n : Number of customers, $(1, \ldots, n)$,
m : Number of depots, $(n+1, \ldots, n+m)$,
l : Number of linehaul customers, $(1, \ldots, l)$,
b : Number of backhaul customers, $(l+1, \ldots, n)$,

All customers and depots are considered as node $(1, \ldots, n+m)$, where m depots are represented as $(n+1, \ldots, n+m)$, l linehaul customers are represented as $(1, \ldots, l)$, and $b = n - l$ backhaul customers are represented as $(l+1, \ldots, n)$.
q_i : Demand of customer i $(i = 1, \ldots, l)$ and $q_i = 0$ for $i = l+1, \ldots, n+m$,
p_i : Supply of customer i $(i = l+1, \ldots, n)$ and $p_i = 0$ for $i = 1, \ldots, l$ and $i = n+1, \ldots, n+m$,
K : Number of vehicle types,
Q_k : Capacity of vehicle type k $(k = 1, \ldots, K)$,
f_k : Fixed cost of vehicle type k $(k = 1, \ldots, K)$,
α_k : Variable cost of vehicle type k $(k = 1, \ldots, K)$,
D_{ij} : Distance between customers i and j $(i, j = 1, \ldots, n+m)$.

Decision Variables:

$$x_{ijdk} = \begin{cases} 1, \text{if the vehicle } k \text{ originating from depot } d \text{ and travelling along} \\ \text{arc } (i,j) \text{ is chosen;} \\ 0, \text{Otherwise.} \end{cases}$$

where $i, j = 1, \ldots, n+m;\ k = 1, \ldots, K;\ d = n+1, \ldots, n+m.$

y_{ij} = The total remaining load on the vehicle travelling along arc (i, j) before reaching customer j.

2.2 The Initial Mathematical Formulation

$$Min\ Z = \sum_{d=n+1}^{n+m} \sum_{k=1}^{K} f_k \sum_{i=n+1}^{n+m} \sum_{j=1}^{l} x_{ijkd} + \sum_{d=n+1}^{n+m} \sum_{k=1}^{K} \sum_{i=1}^{n+m} \sum_{j=1}^{n+m} \alpha_k D_{ij} x_{ijkd} \tag{1}$$

Subject to

$$\sum_{d=n+1}^{n+m}\sum_{k=1}^{K}\sum_{i=1}^{n+m} x_{ijkd} = 1, \qquad j = 1,\ldots,n, \tag{2}$$

$$\sum_{d=n+1}^{n+m}\sum_{k=1}^{K}\sum_{j=1}^{n+m} x_{ijkd} = 1, \qquad i = 1,\ldots,n, \tag{3}$$

$$\sum_{i=1}^{n+m} x_{ijkd} = \sum_{i=1}^{n+m} x_{jikd}, \qquad k = 1,\ldots,K;\ j = 1,\ldots,n+m;\ d = n+1,\ldots,n+m, \tag{4}$$

$$\sum_{i=n+1}^{n+m}\sum_{j=1}^{l} y_{ij} = \sum_{j=1}^{l} q_j, \tag{5}$$

$$\sum_{i=l+1}^{n}\sum_{j=n+1}^{n+m} y_{ij} = \sum_{i=l+1}^{n} p_i, \tag{6}$$

$$\sum_{i=1}^{l} y_{ij} + \sum_{i=n+1}^{n+m} y_{ij} = \sum_{i=1}^{n+m} y_{ji} + q_j, \quad j = 1,\ldots,l, \tag{7}$$

$$\sum_{i=l+1}^{n} y_{ji} + \sum_{i=n+1}^{n+m} y_{ji} = \sum_{i=1}^{n} y_{ij} + p_j, \ j = l+1,\ldots,n, \tag{8}$$

$$y_{ij} \le \sum_{d=n+1}^{n+m}\sum_{k=1}^{K} Q_k x_{ijkd}, \qquad i \ne j = 1,\ldots,n+m, \tag{9}$$

$$y_{ij} = 0, \qquad i = 1,\ldots,l,\ \textit{and}\ j = l+1,\ldots,n+m, \tag{10}$$

$$y_{ij} = 0, \qquad i = n+1,\ldots,n+m,\ \textit{and}\ j = n+1,\ldots,n+m, \tag{11}$$

$$y_{ii} = 0, \qquad i = 1,\ldots,n, \tag{12}$$

$$x_{d_1ikd_2} = 0, \qquad i = 1,\ldots,n;\ k = 1,\ldots,K;\ d_1 \ne d_2 = n+1,\ldots,n+m, \tag{13}$$

$$x_{id_1kd_2} = 0, \qquad i = 1,\ldots,n;\ k = 1,\ldots,K;\ d_1 \ne d_2 = n+1,\ldots,n+m, \tag{14}$$

$$x_{djkd} = 0, \qquad j = l+1,\ldots,n;\ k = 1,\ldots,K;\ d = n+1,\ldots,n+m, \tag{15}$$

$$x_{ijkd} = 0, \qquad i = l+1,\ldots,n;\ j = 1,\ldots,l;\ k = 1,\ldots,K;\ d = n+1,\ldots,n+m, \tag{16}$$

$$x_{ijkd} \in \{0,1\}, \qquad i,j = 1,\ldots,n+m;\ k = 1,\ldots,K;\ d = n+1,\ldots,n+m, \tag{17}$$

$$y_{ij} \ge 0, \qquad i,j = 1,\ldots,n+m, \tag{18}$$

The objective function (1) aims to minimize the total cost. In the first part of the objective function, the fixed cost of each used vehicle is added and the multiplication of variable cost and travelled distance of each used vehicle is also added to the total cost in the second part. While constraint sets (2) and (3) ensure that each customer must be visited by only one vehicle and only once, constraint set (4) ensures the continuity of each route and completion by one vehicle. Constraint (5) equates the total load send from depots to linehaul customers with the total sum of the demands of all linehaul customers. The total load coming from backhaul customers to depots and the total supplies of backhaul customers are equated in constraint (6). Constraints (7) and (8) control the entering and leaving load flow for linehaul and backhaul customers, respectively. The upper bound of the load carried along each arc is equated to the capacity of the vehicle travels along that arc in constraint (9). Constraints (10) guarantee that there is no carried load from linehaul customers to backhaul customers and depots. Also, the carried load amount between depots is not allowed in constraint (11) and the carried load from customer to itself is also not permitted in constraint (12). Constraint (13)

and (14) impose that a vehicle departs and returns to the same depot. While constraint (15) avoids travelling of the vehicles from depots to backhaul customers, constraint (16) avoids travelling of the vehicles from backhaul customers to linehaul customers. Binary decision variables are defined in constraint (17) and continuous, non-negative decision variables are given in constraint (18). It is worth noting that as the type and originating depot of the vehicle travelling along arc (i, j) are determined by the binary decision variable x_{ijdk}, it is not necessary to include indices k and d for the continuous, non-negative decision variable y_{ij}.

2.3 Some tightening of the formulation

Similar modifications to those given by Salhi et al. (2013) are adopted here. In other words, we removed the parts in which the decision variables take zero value from the constraints (2)-(4), and redefined the carried load on each arc separately for linehaul and backhaul customers in constraint (9).

1- Redefining constraint (2)

Constraint (2) is redefined as two constraints as follows, the first part stands for linehaul customers and the second part for backhaul customers. This formulation has the same number of constraints as the previous one, but it has a fewer number of decision variables.

$$\sum_{d=n+1}^{n+m}\left(\sum_{k=1}^{K}\sum_{i=1}^{l} x_{ijkd} + \sum_{k=1}^{K}\sum_{i=n+1}^{n+m} x_{ijkd}\right) = 1,\ j = 1,\ldots,l, \qquad (2a)$$

$$\sum_{d=n+1}^{n+m}\sum_{k=1}^{K}\sum_{i=1}^{n} x_{ijkd} = 1, \qquad j = l+1,\ldots,n. \qquad (2b)$$

2- Redefining constraint (3)

Constraint (3) is redefined similarly as two constraints. This new formulation has also a fewer decision variables.

$$\sum_{d=n+1}^{n+m}\sum_{k=1}^{K}\sum_{j=1}^{n+m} x_{ijkd} = 1,\ \ i = 1,\ldots,l, \qquad (3a)$$

$$\sum_{d=n+1}^{n+m}\sum_{k=1}^{K}\sum_{j=l+1}^{n+m} x_{ijkd} = 1,\ i = l+1,\ldots,n. \qquad (3b)$$

3- Redefining constraint (4)

Constraint (4) can be divided into three constraints for linehaul customers, backhaul customers and depots. This formulation has less number of constraints and less number of decision variables.

$$\sum_{i=1}^{l} x_{ijkd} + \sum_{i=n+1}^{n+m} x_{ijkd} = 1, \qquad k = 1,\dots,K,\ j = 1,\dots,l,\ d = n+1,\dots,n+m, \tag{4a}$$

$$\sum_{i=1}^{n} x_{ijkd} = \sum_{i=l+1}^{n+m} x_{jikd}, \qquad k = 1,\dots,K,\ j = l+1,\dots,n,\ d = n+1,\dots,n+m, \tag{4b}$$

$$\sum_{j=n+1}^{n+m}\sum_{i=1}^{l} x_{jikd} = \sum_{i=1}^{n+m}\sum_{j=n+1}^{n+m} x_{ijkd}, \ k = 1,\dots,K,\ d = n+1,\dots,n+m. \tag{4c}$$

4- Redefining constraint (9)

Constraint (9) can be replaced with four constraints as shown below:

$$y_{dj} \le \sum_{k=1}^{K} Q_k x_{djkd}, \qquad j = 1,\dots,l,\ d = n+1,\dots,n+m, \tag{9a}$$

$$y_{id} \le \sum_{k=1}^{K} Q_k x_{idkd}, \qquad i = l+1,\dots,n,\ d = n+1,\dots,n+m, \tag{9b}$$

$$y_{ij} \le \sum_{k=1}^{K} (Q_k - q_i) x_{ijkd}, \ i \ne j = 1,\dots,l,\ d = n+1,\dots,n+m, \tag{9c}$$

$$y_{ij} \le \sum_{k=1}^{K} Q_k x_{ijkd}, \qquad i \ne j = l+1,\dots,n,\ d = n+1,\dots,n+m. \tag{9d}$$

The original model has $(n+m)(Km+n+m)+n-m$ constraints, $O(Km^2 + Kmn + n^2)$, whereas the restricted model, using the substitution of $n = l + b$ for simplicity, has $2n + Km(n+1) + m(n^2 - 2l(n-l))$ constraints, $O(mn^2 + Kmn)$. A similar calculation is performed for the decision variables. Our modifications have therefore resulted in a reduction of $2mn - n + 2lm(n-l) + (m-1)(Km - n^2 + m)$ constraints, $O(Km^2 + n^2 - mn^2)$, and $2m^2 - 5l^2 + 5mn - 2lm + 6ln - n - m$ decision variables, $O(m^2 + n^2 + mn)$. The restricted model generally obtained better LB and UB values while requiring less or the same amount of CPU time (3 hours) as shown by the interesting and convincing results in the computational experiments section.

3 Hybrid Variable Neighbourhood Search with GRAMPS Algorithm (VNS-GRAMPS)

In this section we first provide an overview, then the algorithm itself followed by some explanation of each of the main steps.

3.1 Overview

The GRAMPS algorithm consists of two stages both using the GRASP algorithm. In the first stage, the Reactive GRASP (RGRASP) algorithm, in which the size of the Restricted Candidate List (RCL) and the neighbourhood size parameter used in the local searches are automatically set. In this stage, the best solution and the appropriate parameter values that yield the best solution are recorded in the memory. In other words, the first stage acts as a training stage whose chosen parameters will be used in stage two. Here, the best solution found so far is used as the initial solution for the GRASP algorithm to search for new solutions based on the parameters identified in stage one. Within the search we adapted techniques to produce initial solutions, a learning process to identify the most appropriate parameters values for the RCL and neighbourhood sizes as well as some guidance on how to implement the local searches based on their respective performances. In this study, we only considered increasing the RCL and neighbourhood size parameters. However, providing flexibility for these parameters to both increase and decrease could be worth examining in the future. In stage 1, the RCL length and the neighbourhood size are re-actively increased by one in every 10 iterations and both are initialized to 5. The parameter values that result in the best solution are then used in the second stage.

3.2 The Algorithm VNS-GRAMPS

The GRAMPS algorithm was proposed by Ahmadi and Osman (2005) for the capacitated clustering problem, and then successfully applied by Tütüncü et al. (2009) and Tütüncü (2010) for the visual interactive decision support system to solve the VRP with Backhauls (VRPB), and the VRP with Heterogeneous Vehicle Fleet (VRPHVF), respectively.

In this study, we solve the (MDHFVRPB) by developing a new hybrid GRAMPS meta-heuristic which applies Variable Neighbourhood Search (VNS) procedure in the local search step. We refer to this hybrid GRAMPS meta-heuristic as VNS with GRAMPS, or VNS-GRAMPS for short.

The first phase VNS-GRAMPS is a RGRASP algorithm and it consists of a solution construction and a local search. The former step starts with the selection of the initial seed solution, i.e., determination of the vehicle combination in each depot. This is constructed using the newly proposed Initial Seed Solution Construction Algorithm (ISSCA) which is given in Section 4.1. Then, the modified Relative Distance Search Algorithm (REDSA) (Tütüncü et al. (2009), Tütüncü (2010)) which is described in Section 4.2 constructs the initial solution by inserting unassigned customers to the routes of the generated initial seed solution. In the local search step, a simple implementation of the VNS algorithm is applied to find an improved solution if possible. The details of the applied VNS algorithm is presented in Section 4.3.

At the beginning of the GRASP phase, all routes of the best solution, recorded in the first RGRASP stage, are sorted in decreasing order with respect to the cost. Then the GRASP phase starts with the first route having the maximum cost and iterates for all routes. At the beginning of each iteration, the Seed Improvement Algorithm (SIA) is applied to obtain the initial seed solution by marking the

customers of the related route and one of its adjacent route as unrouted. Only two customers, one from each route, remain as routed on these two adjacent routes by using a method defined in Section 4.4. The other routes except these two adjacent routes remain unchanged. After obtaining the initial seed solution, an initial solution is constructed with REDSA by assigning the unrouted customers. The aim of the SIA given in Section 4.4 is to construct better initial solutions by considering past information. The local search step of the GRASP phase also uses the same VNS algorithm defined in Section 4.3 with the best neighbourhood size parameter saved in the memory during the RGRASP phase. The GRASP phase continues until a feasible solution that costs less than the average feasible cost saved in memory of the RGRASP phase is found. However, if no solution can be obtained after a certain number of infeasible solutions, the search then terminates.

For completeness, a flow chart describing the overall algorithm of the VNS-GRAMPS meta-heuristic is provided in Figure 1.

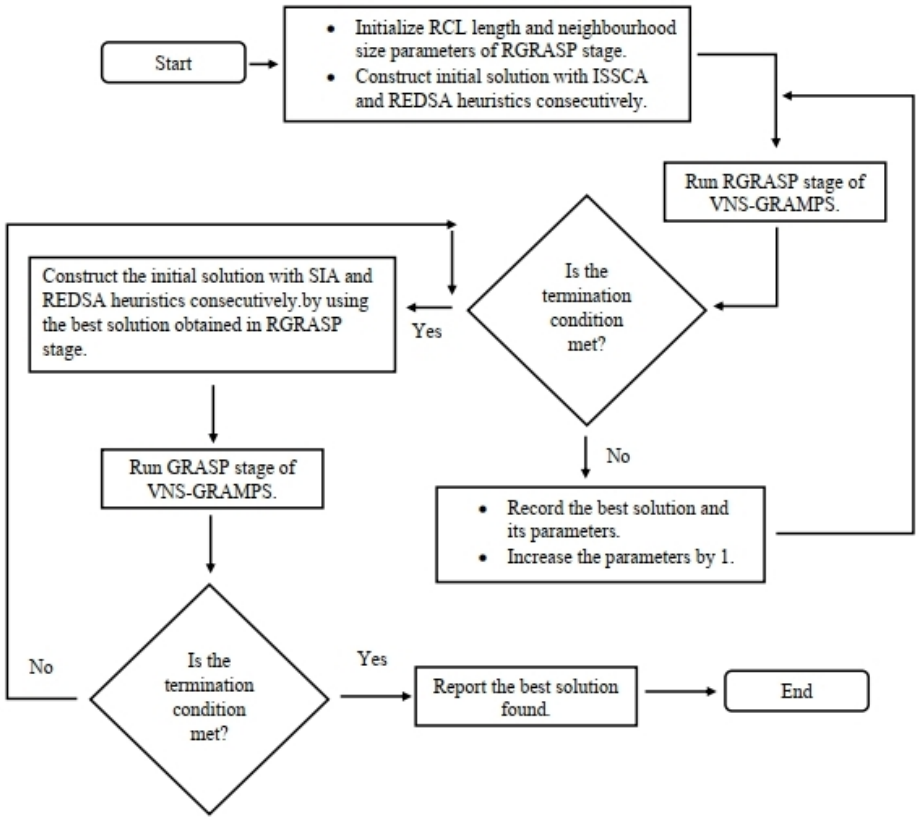

Fig. 1 The flow chart of the overall VNS-GRAMPS meta-heuristic

4 Explanation of the Main Steps

In this section we describe the main ingredients of the algorithms 1 and 2 given in the supplement material (see Kocatürk et al. (2020)). This includes the way the initial solutions are found and the VNS platform with its neighbourhood structures and the various local searches used.

4.1 Initial Seed Solution Construction Algorithm (ISSCA)

GRAMPS algorithm requires a fast procedure to obtain an initial seed solution for the vehicle fleet. Therefore, we have developed ISSCA, which is not only fast, but it can also provide a good initial seed solution with a good vehicle combination. This procedure consists of three steps.

Step 1- ISSCA starts by clustering customers around the depots by using the clustering method given by de Oliveira et al. (2016). Two rules are defined to cluster customers around the depots: (a) The closest depot of customer j and (b) The closest depot of customer k which is the closest to customer j. If the results of these two rules result in having the same depot, then j is assigned to such a depot. However, If the closest depot of j is not the one assigned to customer k, then j can be assigned to both. In other words, customer j is known as a borderline customer of these two depots.

Step 2- The total demand of the customers assigned at each depot/cluster is calculated.

Step 3- In each depot, an initial seed solution is generated by constructing routes with associated vehicle types. As the composition of the vehicle fleet is dependent on the order the depots are examined, we adopt three initial seed solutions. These include (i) sorting the depots in increasing order of their total demand, (ii) same as (i) but in a decreasing order and (iii) using a random depot order. ISSCA then selects the solution having the minimum cost among these three solutions as the initial seed solution. In each depot, the routes are obtained following a cluster first route second strategy.

In each depot, the routes are constructed as follows:

1. The first route r is generated by determining the vehicle type k randomly among the possible vehicle types.
2. Customer i closest to the depot is assigned as the first customer of the first route r.
3. The unrouted customer j closest to customer i is assigned to route r.
4. We continue to assign the unrouted customers closest to the last assigned customer of route r until the capacity of the vehicle type k is exceeded.
5. A complete potential route is formed by joining the last assigned customer to the depot.
6. At this stage the information of the cost of the complete route, the configuration of the route, the last assigned customer and the vehicle type are recorded.
7. The next larger vehicle if any, is then considered and the search continues in the same way to assign the customer closest to the last saved assigned customer. It is worth noting that in the case the vehicle type k has a greater capacity than a pre-determined threshold which we will define later, we assign customer

j that has the minimum distance per unit of demand instead, (i.e., the ratio $\frac{D_{ij}}{d_j}$). We introduced this rule to favour the assignment of larger customers to those vehicles with a larger capacity.

8. If the total load of route r violates the largest vehicle capacity, the search stops assigning customers to route r, and determines the vehicle type k^*, where the last added customer j_{last} based on the vehicle type of the route using the stored information. In other words, route r is constructed by using vehicle k^* and containing all customers from i to j_{last}.
9. The next route is then constructed using a random vehicle type, and the customers are assigned following the same procedure. These steps are repeated until all customers allocated to the depot are assigned. The search starts again with the next chosen depot and its corresponding customers. This process is repeated until all depots are considered.

Note that the customers included in two clusters (borderline customers) are assigned to the routes originated from their two corresponding depots. This provides flexibility for the local search operators to find better routes and more appropriate vehicle types. Since ISSCA will find a vehicle fleet combination whose total capacity is larger than the total demand of the customers, empty routes are incorporated into the search but are obviously deleted from the final solution.

4.2 RElative Distance Search Algorithm (REDSA)

GRAMPS algorithm is an iterative search procedure and it needs an initial solution construction algorithm to start the search in each iteration. We used REDSA which is based on a combination of greediness and randomness. This specification constitutes the greedy randomized nature of GRAMPS. To obtain different initial solutions by REDSA in each iteration of GRAMPS, we therefore opted to leave only one linehaul customer located in the middle position on each route generated by ISSCA.

The aim of the REDSA heuristic is to construct an initial solution by assigning the unrouted customers in the initial seed solution generated by ISSCA. Here, each customer is assigned to one route only. The construction of the routes is based on the following three steps.

Step 1- The insertion cost and the insertion position are evaluated according to the approach proposed by Baker (1992) for each unassigned customer. This insertion cost is calculated as follows: c_{rj} represents the insertion cost of customer j to route r, and this is calculated only for the customers included in the cluster of the depot that the route r is originated from. This is performed for each route r.

$$c_{rj} = \alpha_r \min_{0 \le l \le |I_r|+1} \{D_{i_l j} + D_{j i_{l+1}} - D_{i_l i_{l+1}}\}, \tag{19}$$

where I_r is the set of customers assigned to route r, i_0 and $i_{|I_r|+1}$ represent the depot, and α_r represents the variable cost of route r. The problem constraints are checked during the calculation, and the position, p_{rj}, of customer j in route r that gives the minimum insertion cost is saved.

Step 2- The unassigned customers are individually assigned to the routes by prioritizing first those customers with a single route assignment and the least insertion cost. Then the first and the second least insertion costs are calculated for the customers who can be assigned to multiple routes. This is commonly known as the regret cost or opportunity cost. If there are more than one customer that can only be assigned to a single route, $R_j = 1$, where R_j is the number of routes that customer j can be assigned to, the one with the highest demand is chosen. Each iteration includes a recalculation of the number of routes, R_j. If there is at least one customer which cannot be assigned to any route with $R_j = 0$, there is no feasible solution. In the case of getting an infeasible solution, the heuristic continues to assign customers until all possible customer assignments are completed.

The minimal first and second insertion costs of customer j are represented as c_j^{1st} and c_j^{2nd}, respectively. The insertion priority coefficient of customer j is represented as $IP_j = c_j^{2nd} - c_j^{1st}$ The customer with the highest insertion priority coefficient is given priority during the assignment.

If there are no customers with $R_j = 1$, a customer is selected randomly from the RCL including the customers that can be assigned to multiple routes. RCL is constructed by selecting a certain number of customers with the highest insertion priority coefficients. The pseudo-code of the REDSA algorithm is given in Algorithm 3 in the supplement material (see Kocatürk et al. (2020)) and R^1 represents the number of customers that can only be assigned to a single route.

Here in this step, all insertion costs c_{rj}, respective positions p_{rj} and insertion priority coefficients IP_j are recalculated for each unassigned customer when a customer is assigned to a route. This step is repeated until all possible customer assignments are completed.

Step 3- The 3-Opt heuristic with the best-improvement strategy is applied to all constructed routes.

It is worth stressing that although REDSA aims to satisfy the problem constraints, we may still obtain an infeasible solution because of the existence of some unassigned customers.

In stage one of VNS-GRAMPS, we set the initial length of the RCL to 5 and we increase it by one in every 10 iterations. These two parameters which are set to 5 and 10, are empirically found to be reliable after preliminary experiments. The dynamic updating of the length of the RCL during the first stage led to the final length of the RCL which is then used throughout the second stage of VNS-GRAMPS.

4.3 Variable Neighbourhood Search Algorithm

As part of the local search used in GRASP in both stages, we adopt a commonly used meta-heuristic, namely, the Variable Neighbourhood Search (VNS), originally developed by Mladenović and Hansen (1997). This simple but powerful technique avoids entrapment in a local optimum by adopting a systematic change of neighbourhoods. VNS searches in increasing distant neighbourhoods of the current incumbent solution and moves to a new solution if and only if an improvement

is obtained instead of following a normal trajectory as tabu search or simulated annealing. For more information on these issues and heuristic search in general, see Salhi (2017).

A basic VNS starts by generating an initial solution, x, a set of defining neighbourhood structures $N_k,\ k \in \{1, \ldots, k_{max}\}$, adopting a local search and a stopping criterion. The main steps of the VNS include the use of the shaking, the local search and whether or not to move to the next neighbourhood. In the shaking step, the algorithm randomly generates a new solution, x', in the k^{th} neighbourhood of the solution x ($x' \in N_k(x)$), and then it applies a local search to find the corresponding local optimum solution, x'', around x'. In the movement step, if the solution is improved (i.e., $f(x'') < f(x)$), the new solution becomes the current incumbent solution ($x = x''$) and the search returns to the first neighbourhood (i.e., N_1), otherwise, the next larger neighbourhood is explored. The pseudo code of the VNS is given in Algorithm 4 in the supplement material (see Kocatürk et al. (2020)).

The initial solution constructed by using ISSCA and REDSA heuristics is used as the initial solution of VNS in the local search step of VNS-GRAMPS. In the shaking step, a random solution is created around the respective neighbourhood of the initial solution with respect to some procedures defined in Section 4.3.2. Then, the local optimum of the randomly generated solution is found using the respective local search operators. Finally, the best solution obtained with VNS is updated in the improvement step. These steps are repeated until the maximum number of iterations is reached, and the best solution obtained is compared against the global best solution within the GRASP framework.

4.3.1 Neighbourhood Structures

In this study, we used five neighbourhood structures which are described in the next subsection as part of the local searches. As the order in which these will be used in the VNS is critical, at this stage we do not explicitly denote the k^{th} neighbourhood $N_k(x), k = 1, \ldots, K_{max} = 5$. This will be defined at the end of the next section where an empirical experiment is conducted. The five neighbourhood structures include: One-node interchange, Two-node interchange, Two-shift type 1, Two-shift type 2 and Two-one node interchange.

4.3.2 Local Search Operators

In this section, we explained the five local search operators which are based on the five neighbourhood structures mentioned earlier.

One-node Interchange: In this local search, a customer is selected from a route, and then it is checked for insertion to another route.

In the shaking step of VNS, the random solution x' is generated from the current incumbent solution x as follows: First, a random route is selected with respect to the controlled randomized function given in Equation (20), and a random customer is chosen from the selected route with respect to the controlled randomized function given in Equation (21). In equation (20), we give importance to the route having the highest cost, and we focus on the customer that will result in the highest saving when it is moved to another route. Then, the selected customer, say j, is checked for insertion in the routes that are in the ρ neighbourhoods of j.

These are the routes of the closest ρ customers or depots of j. This is an important neighbourhood reduction scheme that cuts the unnecessary computations of the non-promising moves. This aspect is strongly demonstrated in Salhi and Sari (1997) and recently in Sze et al. (2016). If one of the closest nodes of j is a depot, then all routes originating from that depot are also included into the neighbouring routes of j. While checking customer j for insertion to a neighbouring route r, we adopt the best improvement strategy (i.e., we move customer j to the best position in the route r, if there exists a cost saving). We apply the customer interchanges if it is a feasible movement. If the algorithm could not find a feasible position that provides a cost saving in route r, the search continues with the next neighbouring route.

In the local search step, we check all routes by starting from customer j that is in the first position of the first route. In other words, this is an exhaustive application of the shaking in $N_1(.)$ as explained above. As before, we check customer j for insertion to the ρ neighbour routes of j. If it finds a feasible move having a cost saving, it moves the customer j, and continues to search with the next customer.

$$r\_rand_r = (f_r + \alpha_r * Distance_r) * U(0,1), \forall r \in \{1, \ldots, r_{max}\} \tag{20}$$

where f_r is the fixed cost, α_r is the variable cost, and $Distance_r$ is the total travel distance of route r, $U(0,1)$ is a uniform random number in the interval $[0,1]$, and r_{max} is the total number of routes.

$$pos\_rand_i = (D_{i-1,i} + D_{i,i+1} - D_{i-1,i+1}) * U(0,1), \forall i \in I_r = \{1, \ldots, |I_r|\} \tag{21}$$

where $D_{i,i+1}$ is the distance between customers i and $i+1$, and I_r is the set of customers in the route r. Additionally, the customers with indices 0 and $|I_r|+1$ represent the originated depot of route r.

Two-Node Interchange: Here, a random customer is selected from a route, and then the selected customer is swapped with another customer from a different route. This application is applied to all customers and to all routes and the interchange that results in the overall best saving is selected.

In the shaking step which represents one move of the local search, is defined as follows. A random solution x' is generated from the current incumbent solution x as follows: First, a random route is selected with respect to Equation (20), and then a customer is chosen randomly from the selected route with respect to Equation (21). Then, the selected customer, say j_1, is checked for insertion to its ρ neighbour routes. The first possible neighbour route that is not violating the problem constraints is selected as the second route to which the customer change is applied. A customer, say j_2, is picked randomly from the selected route with Equation (21), and these customers are swapped.

Two-Shift Type 1: In this local search, two adjacent customers are selected from a route, and then the selected customers are checked for insertion to another route in the same order. This is applied to all customers and all routes and the one that yields the best saving is selected. The random solution x' is generated from the current incumbent solution x as follows in the shaking step: First, a random route is selected with respect to Equation (20), and then two adjacent customers are chosen randomly from the selected route with respect to Equation (22). In this equation, we focus on the customers having the minimum distance between

them and having the maximum saving when they are moved from the route by minimizing the obtained ratio. Then, the selected customers, say j_1 and j_2, are checked for insertion in the ρ neighbouring routes.

$$p\_rand2_i = \frac{D_{i,i+1} + D_{i-1,i+1}}{(D_{i-1,i} + D_{i,i+1} + D_{i+1,i+2}) * U(0,1)}, \forall i \in I_r = \{1, \ldots, |I_r|\} \quad (22)$$

Two-Shift Type 2: In this local search, two adjacent customers are selected from a random route, and these customers are inserted into two different routes. This is applied for all customers and all routes and the best one is chosen. For the shaking step, the random solution x' is generated from the current incumbent solution x as follows. First, a random route is selected, and two adjacent customers are chosen randomly by using Equation (22) from the selected route. Then, the selected customers, say j_1 and j_2, are checked for insertion to the routes that are in the ρ neighbourhoods. The insertion order of the customers is determined randomly.

Two-One Node Interchange: Here, two adjacent customers are selected from a route, and then these customers are moved to a different route without changing the order of the customers. Moreover, a customer from the target route is moved into the origin route of the adjacent customers. This application is performed for all customers and all routes and the one producing the overall best saving is chosen. The random solution x' is generated from the current incumbent solution x as follows in the shaking step: First, a random route is selected with respect to Equation (20), and then two adjacent customers are chosen randomly from the selected route by using Equation (22). Then, the selected customers, say j_1 and j_2, are checked for insertion to the ρ neighbouring routes. The customers are moved into a route that is not violating the problem constraints, and a customer from the target route is moved into the originated route of customers j_1 and j_2.

4.3.3 Performance measures of the local searches

We determined the application order of the local search operators in the local search step of VNS with respect to the following two performance criteria. We used the success ratio and the average improvement to assess the performance of the operators.

i) $Success\_Ratio = \frac{\#\ iterations\ the\ operator\ l\ improved\ the\ solution}{\#\ iterations\ the\ operator\ l\ entered} \times 100,$

ii) $Average\_Improvement = \frac{Total\ cost\ improvement\ of\ the\ operator\ l\ (\%)}{\#\ iterations\ the\ operator\ l\ improved\ the\ solution}.$

We calculated the effective improvement ratio of a local search operator given in Equation (23) in order to calculate the improvement performance of the operator in an iteration in which the operator improved the solution. We reported the effective improvement ratios of the local search operators for 12 problem instances in Table 1, and we selected one or two instances in which VNS-GRAMPS performed better among the instances having the same number of customers. According to the average effective improvement ratios, the Two-shift Type 2 obtained the best value as 1.52%, the Two-shift Type 1 obtained 0.63%, the Two-one Node Interchange got 0.23%, the One-node Interchange a 0.12% and finally the Two-node Interchange

obtained 0.10% only. We then re-order the neighbourhood structures as follows: Two-shift Type 2 as N_1, Two-shift Type 1 as N_2, Two-one Node Interchange as N_3, One- node interchange as N_4, and Two-node Interchange as N_5.

$$Effective\_Improvement\_Ratio = \frac{Average\_Improvement}{Success\_Ratio} \tag{23}$$

Table 1 Effective improvement ratios of local search operators

Instance No	One-node Interchange	Two-node Interchange	Two-shift Type 1	Two-shift Type 2	Two-one Node Interchange
1	0.50	0.10	3.38	5.26	1.50
3	0.04	0.05	0.27	1.69	0.10
5	0.15	0.12	1.04	3.42	0.08
6	0.09	0.12	1.14	1.76	0.22
8	0.06	0.24	0.24	0.42	0.14
9	0.05	0.07	0.25	1.34	0.09
11	0.03	0.02	0.10	1.78	0.06
12	0.02	0.02	0.06	0.15	0.03
15	0.22	0.18	0.32	0.83	0.31
18	0.10	0.15	0.47	0.38	0.11
23	0.11	0.09	0.25	1.08	0.11
24	0.03	0.04	0.04	0.09	0.02
Average:	0.12	0.10	0.63	1.52	0.23

4.4 Seed Improvement Algorithm (SIA)

In the second stage of the VNS-GRAMPS meta-heuristic, the initial seed solution is generated based on the best solution found in the first stage. This is performed as follows:

- All routes of the best solution are ordered in decreasing value of the cost,
- The GRASP stage starts with the first route in the list (i.e., the one having the largest cost) and then iterates for all routes,
- An adjacent route of the one having the next greater route index and originating from the same depot is selected,
- Two customers, one from each route of the two selected routes, that maximize the function g_{ij} in Equation (24), are chosen as the remaining customers,
- The other customers from these two routes are deleted,
- The other routes of the solution remain unchanged.

Selection Criterion

We develop the following criterion to select the two customers one from each of the two routes.

$$g_{ij} = \frac{\theta_{ij}}{\max \theta_{ij}} + \frac{(D_{0i} + D_{0j})}{2 * \max D_{ij}} \tag{24}$$

This function is used to define new initial seed solutions in order to search for different regions of the solution space. Here, θ_{ij} represents the angle between

customers i and j, and D_{ij} refers to the distance between these two customers. In Equation (24), D_{0i} and D_{0j} represent the distances between the customers i, j and the depot of origin. The first part of the function given in Equation (24) is used to decrease the route overlaps. Our aim here is to select two customers forming the largest angle with the depot as the new seeds in the area spanned by the selected routes. The second part of the function prevents the selection of any new seeds that happen to be too close to the depot.

5 Computational Experiments

In this section, three data sets are introduced for bench-marking purposes, then the performance of the two mathematical models are compared. This is followed by the results and the analysis of the hybrid VNS-GRAMPS and basic VNS algorithms. Hybrid VNS-GRAMPS and VNS algorithms are coded using C# programming language and run on a PC having Intel(R) Core(TM) i3-3110M CPU @ 2.40 GHz CPU, 8 GB RAM and Windows 10 Prof. 64 bit operating system.

5.1 Problem Instances

As the the problem instances of MDHFVRPB are not available in the literature we generated three scenarios based on the data sets of related routing problems that are widely used in the literature. The data sets can be accessed via the url https://github.com/fatihkocaturk/MDHFVRPB-Data-Sets under "MDHFVRPB-Instances.rar" file. The solution files can also be collected from the corresponding author.

(i) Scenario 1- In this scenario, we produce a new MDHFVRPB data set by combining the depot information of the Multi-Depot Heterogeneous VRP (MDHFVRP) used in Salhi et al. (2014) with the problem instances of the Fleet Size and Mix VRP with Backhauls (Salhi et al. (2013)). The derived problem instances are originally based on the problem set of heterogeneous vehicle fleet VRP of Golden et al. (1984) and the problem set of VRP with backhauls of Toth and Vigo (1997).

This new MDHFVRPB data set is given in Table 9 of the supplement material (see Kocatürk et al. (2020)).

(ii) Scenario 2- Here, the data set is generated by defining linehaul and backhaul customers for each of the 26 MDHFVRP problem instances used in Salhi and Sari (1997). Three linehaul/backhaul percentages as 50/50, 67/33 and 80/20 are generated for each problem instance. The customer locations, demands, heterogeneous vehicle fleet are all kept unchanged. As in the literature, these new MDHFVRPB instances follow also the same pattern by using the first customer of every two, three and five customers as backhaul customer for the 50/50, 67/33 and 80/20 linehaul/backhaul percentages, respectively. These new data set is given in Table 10 and Table 11 of the supplement material (see Kocatürk et al. (2020)).

(iii) Scenario 3- This third data set is derived by defining linehaul and backhaul customers for the 10 MDHFVRP problem instances given in Vidal et al. (2014) which are based on the MDVRP problem instances of Cordeau et al. (1997). Similarly to the earlier two scenarios, three linehaul/backhaul percentages as 50/50,

67/33 and 80/20 are generated here for each problem instance while the customer locations, the demands and the heterogeneous vehicle fleet are kept the same. This MDHFVRPB new data set is reported in Table 12 of the supplement material (see Kocatürk et al. (2020)).

5.2 Performance of the two mathematical models

In this section CPLEX results for MDHFVRPB are reported. The proposed basic and restricted models are modelled by using GAMS 23.9.4 and solved with IBM ILOG CPLEX 12.4.0.1 solver. The experiments were run on a computer having Intel (R) Core (TM) i5-2310 CPU @ 2.90 GHz CPU, 4 GB RAM and Windows 7 Prof. 64 bit operating system. We conducted 3 hours CPU time limit to solve the basic and restricted models.
(i) Case of Scenario 1- The CPLEX results for MDHFVRPB problem instances of scenario 1 are reported in Table 1 of the supplement material (see Kocatürk et al. (2020)). We found the optimal solutions of 10 out of 12 instances with 20 customers with both models. In general, we can note that of those 10 instances the reduced model requires relatively shorter CPU times and in the remaining 26 instances, where optimality cannot be guaranteed, it yields relatively tighter Lower Bounds (LB) except for two cases, namely, instances #12 and #31.
(ii) Case of Scenario 2- The CPLEX results for MDHFVRPB problem instances of scenario 2 are reported in Tables 2 and 3 of the supplement material (see Kocatürk et al. (2020)). The basic model found tighter LB values for 12 out of 78 problem instances only, and lower UB values for 27 problems. However, the basic model found lower UB values for 5 out of 9 problems with 360 customers, while the restricted model obtained a lower UB value for the problem #77 only. On the other hand, the restricted model found tighter LB values for 8 of the problems with 360 customers, while the basic model found one only, namely, problem #71. While the basic model found lower UB values for only 17 out of 48 problem instances with 100 and fewer customers, it only found tighter LB values for 6 instances. As a result, the restricted model was able to find tighter LB values for both small and large sized problems. The basic model performed better at obtaining lower UB values for large sized problems while lower UB values for small sized problems are found by the reduced model.
(iii) Case of Scenario 3- The CPLEX results for MDHFVRPB problem instances of scenario 3 are reported in Table 4 of the supplement material (see Kocatürk et al. (2020)). The basic model obtained lower UB values for only 8 out of 30 problem instances, and 6 of them are small sized problems with 144 or fewer customers. In addition, the basic model was able to find tighter LB values for only 5 out of 30 problems, 3 of which are small sized problems. The restricted model behaves much better here where it obtained lower UB and tighter LB values for large and small sized problem instances.

5.3 Performance of the VNS-GRAMPS vs a basic VNS

The problem instances were also solved by a basic VNS meta-heuristic implementation. This is performed in order to compare the performance of our VNS-GRAMPS

meta-heuristic. In the initialization step of the VNS algorithm, the initial solution was constructed by using ISSCA and REDSA heuristics. In the shaking step, the random solution around the respective neighbourhood was obtained by using the same procedure given in Section 4.3. The five neighbourhoods defined in Section 4.3.2 were also used in the local search step of the VNS and applied in the same order, namely, two-shift type 2, two-shift type 1, two-one node interchange, one-node interchange and two-node interchange. The maximum number of iterations was used as the stopping criterion and set to 100. The VNS algorithm was run 10 times for each problem instance and the result of the best solution was reported. The CPU times reported for the VNS algorithm are the average over the 10 runs. Note that, the implementation was designed and aimed to set the total number of iterations to approximately the same value between VNS-GRAMPS and VNS algorithms in order to make a fair comparison. For this purpose, the outer iteration number of the RGRASP phase of VNS-GRAMPS was set to 100 and the outer iteration number of the VNS algorithm used in the local search step was restricted to 10. Another strategy would be to set the same overall CPU time for both implementations even though one will use more iterations than the other.

The performances of the algorithms are assessed using the average percent gap values against the LB and UB values obtained by CPLEX. These gaps (in %) are given in the following Equations (25) and (26.)

$$Gap_{LB}(\%) = \frac{Z_{alg} - LB}{LB} \times 100 \tag{25}$$

$$Gap_{UB}(\%) = \frac{Z_{alg} - UB}{UB} \times 100 \tag{26}$$

where Z_{alg} is the cost of the solution found by VNS-GRAMPS or VNS algorithms whereas LB and UB are the lower and upper bound values obtained by CPLEX, respectively.

5.3.1 Results for Scenario 1

The detailed results for the MDHFVRPB problem instances of scenario 1 can be found in Table 5 of the supplement material (see Kocatürk et al. (2020)).

We obtain the optimal solutions for the problems #2, #3, #7, #8, #9 and #11 by VNS-GRAMPS. The best solutions were also found for 14 out of 24 problem instances having 50 and more customers by VNS-GRAMPS, and the other 10 out of 24 best solutions were obtained by CPLEX. The lowest average gap against LB is 13.01% for VNS-GRAMPS, and the average gaps against LB for UB and VNS were reported as 39.05% and 18.44%, respectively. The performance of VNS-GRAMPS and VNS was also compared with respect to the average gaps against UB values recorded by CPLEX. The lowest average gap against UB was −9.39% for VNS-GRAMPS and −5.08% for VNS. With regard to CPU times, VNS-GRAMPS obtained the solutions in relatively shorter CPU times for the problem instances with 20 customers, but it took longer for the problems with 50 and more customers. This is because VNS-GRAMPS applies VNS in the local search step at each iteration. Finally, VNS-GRAMPS required 4.34 minutes CPU time on average, whereas VNS, as expected, needed a relatively shorter CPU time of 1.54 minutes only.

The summary results of UB, VNS and VNS-GRAMPS are reported in Table 2. In the case of using the average gap values against LB, the average gap value of UB for the problem instances with 75 customers was relatively very large, 151.35%. The largest average gap values against LB were also obtained by VNS and VNS-GRAMPS for the instances with 75 customers. It is worth mentioning that for the smaller instances, namely, those with 20 and 50 customers, the best average gap values against LB were found by CPLEX, whereas for the larger instances, namely, those with 75 and 100 customers, the best gaps against LB were obtained by VNS-GRAMPS. In the case of using the average gap values against UB, the best gap values were obtained by VNS-GRAMPS for both smaller and larger instances.

Table 2 Summary of VNS and VNS-GRAMPS Test Results for Scenario 1

N	# Best Solutions			Average Gap wrt LB (%)			Average Gap wrt UB (%)		CPU (min)	
	UB	VNS	VNS-GRAMPS	UB	VNS	VNS-GRAMPS	VNS	VNS-GRAMPS	VNS	VNS-GRAMPS
20	12/12	1/12	6/12	0.33	5.51	2.37	5.14	2.01	0.89	0.39
50	8/12	0/12	4/12	12.16	20.44	14.58	7.67	2.37	1.16	2.39
75	0/6	0/6	6/6	151.35	32.45	25.91	-44.69	-47.40	1.74	7.37
100	2/6	0/6	4/6	57.99	26.30	18.26	-11.40	-17.72	3.40	13.12
			Average:	39.05	18.44	**13.01**	-5.08	**-9.39**	**1.54**	4.34

5.3.2 Results for Scenario 2

The detailed results for Scenario 2 can be found in Table 6 and Table 7 of the supplement material (see Kocatürk et al. (2020)).

The VNS algorithm was able to find the same or better results than VNS-GRAMPS for only 9 out of 78 problem instances. VNS and VNS-GRAMPS found the same result for 3 of these 9 problems (#13, #46, #49), while CPLEX found the same UB values for problems #46 and #49. When the performances of the VNS and VNS-GRAMPS algorithms were compared according to the gap from the LB values found with CPLEX, an average of 41.90% and 44.58% gap values were obtained for VNS-GRAMPS and VNS, respectively. On the other hand, an average of -57.88% and -57.02% gap values were obtained for VNS-GRAMPS and VNS, respectively, when the performances of the VNS and VNS-GRAMPS algorithms were compared according to the gap from the UB values found with CPLEX. The reason for the average gap values against UB being so close is that there was no UB values for 18 out of 78 problem instances. In terms of CPU times, VNS always found the solutions in less time due to the same reason noted in the earlier section. For instance, VNS-GRAMPS found a solution in 35.11 minutes on average while VNS in 10.26 minutes.

In Table 3, summary performance values of VNS and VNS-GRAMPS algorithms are reported. The results are presented under two categories, namely, the small-sized problems, with 100 or less customers, and the large-sized problems, with 160 or more customers. In the first category, VNS-GRAMPS found better solutions than VNS. VNS-GRAMPS obtained the largest average gap value against LB for instances with 75 customers and the smallest average gap value against LB for those with 55 customers. In brief, VNS-GRAMPS and VNS algorithms achieved an average gap of 25.61% and 28.31% respectively, with respect to LB.

When the average gap values from the UB values obtained with CPLEX, VNS-GRAMPS obtained lower gap values than VNS for all cases. While, the lowest gap value against UB was obtained for instances with 75 customers, the largest gap value was obtained for instances with 80 customers. To sum up, VNS-GRAMPS and VNS algorithms achieved an average gap of -57.01% and -56.21% respectively, with respect to UB. VNS is much faster than VNS-GRAMPS where an average of 2.04 minutes was required by the former and 5.07 minutes by the latter. In the second category, VNS-GRAMPS obtained the largest average gap value against LB for problems with 360 customers but the smallest value for instances with 160 customers. VNS-GRAMPS and VNS algorithms obtained an average of 58.18% and 60.85% gap values respectively. When the performances of the algorithms were compared with respect to the average gap values against UB, VNS-GRAMPS always obtained smaller gap values than VNS. VNS-GRAMPS obtained the smallest gap value for instances with 249 customers and the largest gap for instances with 160 customers. VNS-GRAMPS and VNS algorithms obtained an average of -58.87% and -57.95% gap values, respectively, with respect to UB. With respect to CPU times, VNS is faster than VNS-GRAMPS with the former using an average of 12.54 minutes while the latter requiring 46.81 minutes.

Table 3 Summary of VNS and VNS-GRAMPS Test Results for Scenario 2

Small-Sized

N	# Best Solutions		Average Gap wrt LB (%)		Average Gap wrt UB (%)		CPU (min)	
	VNS	VNS-GRAMPS	VNS	VNS-GRAMPS	VNS	VNS-GRAMPS	VNS	VNS-GRAMPS
50	1/6	6/6	23.38	20.86	-68.78	-69.45	1.19	2.02
55	0/3	3/3	18.33	15.10	-70.76	-71.60	1.14	2.31
75	0/3	3/3	41.34	40.12	-79.95	-80.22	1.71	4.50
80	2/9	9/9	25.43	22.15	-19.55	-20.86	2.01	7.64
85	0/6	6/6	19.27	17.37	-72.60	-73.03	1.84	4.92
100	0/12	12/12	36.68	33.69	-68.42	-69.06	2.89	5.59
		Average:	28.31	**25.61**	-56.21	**-57.01**	**2.04**	5.07

Large-Sized

N	# Best Solutions		Average Gap wrt LB (%)		Average Gap wrt UB (%)		CPU (min)	
	VNS	VNS-GRAMPS	VNS	VNS-GRAMPS	VNS	VNS-GRAMPS	VNS	VNS-GRAMPS
160	0/9	9/9	43.73	39.72	-33.53	-35.97	4.87	17.25
240	4/9	5/9	70.29	69.59	-48.06	-48.31	10.02	36.24
249	1/12	11/12	50.10	47.57	-84.78	-85.02	12.02	59.97
360	1/9	8/9	82.86	79.38	-64.63	-65.11	23.41	69.37
		Average:	60.85	**58.18**	-57.95	**-58.87**	**12.54**	46.81

Presence of maximum distance/time constraint-

In Table 4, a summary performance for the two algorithms is reported according to the maximum travel distance, TD, constraint. The application of TD constraint to the problems decreased the average gap values obtained against LB. Also, as the TD constraint value increased, the average gap against LB increased, i.e. the solution quality decreased. In other words, we can conclude that the addition of TD constraints eases the problem and improves the performance of the meta-heuristics. One of the reason for the increase in performance of the meta-heuristics is due to the fact that the TD constraint results in the generation of solutions consisting of many routes with fewer customers. This feature provides extra flexibility to the local search to improve the solution. On the contrary, the

average gap values against UB decreased as the TD constraint value increased, i.e. the UB values found by CPLEX increased. In terms of CPU time, the TD constraint increased the problem solving times. Furthermore, it is noted that as TD constraint value increased, the CPU times also increased.

Table 4 VNS and VNS-GRAMPS Summary Test Results with respect to Travel Distance Constraint for Scenario 2

TD	# Best Solutions		Average Gap wrt LB (%)		Average Gap wrt UB (%)		CPU (min)	
	VNS	VNS-GRAMPS	VNS	VNS-GRAMPS	VNS	VNS-GRAMPS	VNS	VNS-GRAMPS
180	3/12	10/12	22.63	20.31	-21.01	-22.33	5.07	30.63
200	2/12	11/12	21.52	19.20	-32.25	-33.72	6.99	32.72
310	1/12	11/12	50.10	47.57	-84.78	-85.02	12.02	59.97
∞	3/42	40/42	55.86	52.92	-71.91	-72.53	6.65	12.94
		Average:	44.58	**41.90**	-57.02	**-57.88**	**7.29**	25.94

5.3.3 Results for Scenario 3

The detailed results of the VNS-GRAMPS and VNS meta-heuristics for scenario 3 data set are reported in Table 8 of the supplement material (see Kocatürk et al. (2020)). The VNS algorithm only found better results than VNS-GRAMPS for two problem instances, #3 and #28. VNS-GRAMPS and VNS algorithms have found solutions with average gap values of 91.08% and 96.07%, respectively, from the LB values found with CPLEX. When the solutions are compared with respect to the average gaps from the UB values of CPLEX, VNS-GRAMPS and VNS obtained solutions with average gap values of -72.38% and -71.82%, respectively. While the VNS algorithm found a solution in an average of 11.46 minutes, VNS-GRAMPS required an average of 29.88 minutes.

In Table 5, the performance of VNS-GRAMPS and VNS algorithms is summarized under two categories similarly to scenario 2. Here, the small-sized problems have 96 and fewer customers whereas the large ones contain 144 and more customers. In the first group, VNS-GRAMPS achieved lower average gap values against LB for each problem size, and as the number of customers increased, the average gap values increased. Here, VNS-GRAMPS achieved an average gap of 36.81% against LB while VNS obtained 39.98%. VNS-GRAMPS also obtained lower average gap values against UB than VNS for each problem size. The average gap values against UB were decreased as the number of customers increased, since the UB values found with CPLEX increased. VNS-GRAMPS achieved an average gap of -47.70% against UB while VNS obtained -46.91%. However, VNS solved these small-sized problems in an average of 2.30 minutes while VNS-GRAMPS needed an average of 4.08 minutes. In the second group, VNS-GRAMPS obtained lower average gap values against LB for each problem size, and the average gap values generally increased as the number of customers increased. VNS-GRAMPS and VNS obtained the largest average gap against LB for problem size with 216 customers and the smallest average gap against LB for problem size with 144 customers. For large-sized problems, VNS-GRAMPS and VNS achieved an average gap of 114.34% and 120.11%, respectively, with respect to LB. When the results were compared with respect to the average gap values from the UB values found

with CPLEX, VNS-GRAMPS always found smaller gap values, but the obtained gap results were similar and there was a slight difference for each class. The reason for this similarity was that the UB values found by CPLEX for larger problem instances were too far from the solutons found by VNS-GRAMPS and VNS. VNS found solutions in shorter average CPU time for each problem size. For instance, VNS solved large problems in an average of 15.38 minutes, while VNS-GRAMPS required an average of 40.93 minutes.

Table 5 VNS and VNS-GRAMPS Summary Test Results for Scenario 3

Small-Sized

N	# Best Solutions		Average Gap wrt LB (%)		Average Gap wrt UB (%)		CPU (min)	
	VNS	VNS-GRAMPS	VNS	VNS-GRAMPS	VNS	VNS-GRAMPS	VNS	VNS-GRAMPS
48	1/3	2/3	13.56	12.91	-6.72	-7.38	1.32	1.41
72	0/3	3/3	50.94	45.77	-80.09	-80.72	1.99	3.68
96	0/3	3/3	55.45	51.76	-74.01	-75.15	3.59	7.16
		Average:	39.98	**36.81**	-46.91	**-47.70**	**2.30**	4.08

Large-Sized

N	# Best Solutions		Average Gap wrt LB (%)		Average Gap wrt UB (%)		CPU (min)	
	VNS	VNS-GRAMPS	VNS	VNS-GRAMPS	VNS	VNS-GRAMPS	VNS	VNS-GRAMPS
144	0/6	6/6	90.07	83.62	-83.08	-83.81	6.97	17.79
192	0/3	3/3	110.46	105.11	-83.14	-83.49	12.10	32.18
216	0/3	3/3	156.75	149.79	-81.14	-81.53	14.85	40.97
240	0/3	3/3	124.53	121.08	-83.87	-84.02	18.09	52.26
288	1/6	5/6	134.45	128.56	-87.79	-88.18	24.35	62.76
		Average:	120.11	**114.34**	-84.27	**-84.72**	**15.38**	40.93

6 Conclusion and Suggestions

In this paper, we introduced a new logistical problem that is commonly faced in practice. In real life, the companies generally have a heterogeneous vehicle fleet to serve customers and multiple depots to supply the demands of the customers. We called this practical problem as the Multi-Depot Heterogeneous VRP with Backhauls (MDHFVRPB). We first defined two mathematical models where the first one is a basic one whereas the second one contains some new added neighbourhood reductions. We also generated new problem instances as these do not exist in the literature. To solve larger instance, we then developed an interesting hybridisation of VNS and GRASP which we refer to as the VNS-GRAMPS meta-heuristic. For comparison purposes we also present a basic VNS meta-heuristic. We conducted 3 hours of CPU time limit to solve the basic and restricted mathematical models and found the optimal solutions of 10 out of 12 problems with 20 customers of scenario 1 data set with both models. The restricted model obtained the optimal solutions in shorter CPU times and generally found tighter lower bound for the other 26 instances where optimality cannot be guaranteed. The restricted model also found tighter lower bound in general for the generated data sets for scenarios 2 and 3.

When using VNS-GRAMPS, we obtained the optimal solutions for the 6 instances of scenario 1, namely, #2, #3, #7, #8, #9 and #11. The best solutions

were also obtained for 14 out of 24 problem instances having 50 and more customers by VNS-GRAMPS, and the other 10 out of 24 best solutions were obtained by CPLEX. The lowest average gap obtained is 13.01% for VNS-GRAMPS, and the average gaps for UB and VNS were reported as 39.05% and 18.44%, respectively. With regards to CPU times, VNS-GRAMPS requires shorter CPU times for the problems with 20 customers, but needed much longer amount of time for the problem instances with 50 and more customers.

The following research avenues could be worth exploring. For instance, in real life, it is often not practical to visit backhaul customers after completing all deliveries. Hence, one way forward is to analyse other extensions of MDHFVRPB including different pickup and delivery orders such as mixed and simultaneous. In this study, we only increased the size of the RCL dynamically but this could be made more flexible by allowing both the increase as well as the decrease. Another approach that integrates VNS with other meta-heuristics such as large neighbourhood search instead of GRASP could also be worthwhile studying. One way forward would be to hybridise one of the meta-heuristics with exact methods to yield an effective matheuristic resulting in tight bounds and exciting mathematical properties.

Acknowledgement

The authors acknowledge the support of the Scientific and Technological Research Council of Turkey (TÜBİTAK), grant number 1001 - 213M438. The authors are also grateful to both referees whose invaluable comments have improved the content as well as the presentation of the paper.

Conflict of interest

The authors declare that they have no conflict of interest.

References

Ahmadi S, Osman IH (2005) Greedy random adaptive memory programming search for the capacitated clustering problem. European Journal of Operational Research 162(1):30 – 44, DOI http://dx.doi.org/10.1016/j.ejor.2003.08.066, logistics: From Theory to Application

Baker BM (1992) Further inprovements to vehicle routeing heuristics. Journal of the Operational Research Society 43(10):1009–1012, DOI 10.1057/jors.1992.152

Berbeglia G, Cordeau JF, Gribkovskaia I, Laporte G (2007) Static pickup and delivery problems: a classification scheme and survey. TOP 15(1):1–31, DOI 10.1007/s11750-007-0009-0

Cordeau JF, Gendreau M, Laporte G (1997) A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks 30(2):105–119, DOI 10.1002/(SICI)1097-0037(199709)30:2⟨105::AID-NET5⟩3.0.CO;2-G

Golden B, Assad A, Levy L, Gheysens F (1984) The fleet size and mix vehicle routing problem. Computers & Operations Research 11(1):49 – 66, DOI https://doi.org/10.1016/0305-0548(84)90007-8

Irnich S (2000) A multi-depot pickup and delivery problem with a single hub and heterogeneous vehicles. European Journal of Operational Research 122(2):310 – 328, DOI http://dx.doi.org/10.1016/S0377-2217(99)00235-0

Kocatürk F, Tütüncü GY, Salhi S (2020) Supplementary Material for The Multi-Depot Heterogeneous VRP with Backhauls: Formulation and a Hybrid VNS with GRAMPS Meta-heuristic Approach. `https://drive.google.com/file/d/1NQEoSYvzyzSLfVMyQLree9Bzwv1Jv0ZL/view`, [Online; accessed 22-September-2020]

Koç C, Laporte G (2018) Vehicle routing with backhauls: Review and research perspectives. Computers & Operations Research 91:79 – 91, DOI https://doi.org/10.1016/j.cor.2017.11.003

Koç C, Laporte G, Tükenmez I (2020) A review of vehicle routing with simultaneous pickup and delivery. Computers & Operations Research 122:104987, DOI https://doi.org/10.1016/j.cor.2020.104987

Li J, Pardalos PM, Sun H, Pei J, Zhang Y (2015) Iterated local search embedded adaptive neighborhood selection approach for the multi-depot vehicle routing problem with simultaneous deliveries and pickups. Expert Systems with Applications 42(7):3551 – 3561, DOI http://dx.doi.org/10.1016/j.eswa.2014.12.004

Mladenović N, Hansen P (1997) Variable neighborhood search. Computers & Operations Research 24(11):1097 – 1100, DOI https://doi.org/10.1016/S0305-0548(97)00031-2

Nagy G, Salhi S (2005) Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. European Journal of Operational Research 162(1):126 – 141, DOI http://dx.doi.org/10.1016/j.ejor.2002.11.003, logistics: From Theory to Application

de Oliveira FB, Enayatifar R, Sadaei HJ, Guimarães FG, Potvin JY (2016) A cooperative coevolutionary algorithm for the multi-depot vehicle routing problem. Expert Systems with Applications 43:117 – 130, DOI https://doi.org/10.1016/j.eswa.2015.08.030

Penna PHV, Subramanian A, Ochi LS, Vidal T, Prins C (2019) A hybrid heuristic for a broad class of vehicle routing problems with heterogeneous fleet. Annals of Operations Research 273(1):5–74, DOI 10.1007/s10479-017-2642-9

Salhi S (2017) Heuristic Search: The Emerging Science of Problem Solving. Palgrave Macmillan, DOI 10.1007/978-3-319-49355-8

Salhi S, Nagy G (1999) A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. The Journal of the Operational Research Society 50(10):1034–1042

Salhi S, Sari M (1997) A multi-level composite heuristic for the multi-depot vehicle fleet mix problem. European Journal of Operational Research 103(1):95 – 112, DOI http://dx.doi.org/10.1016/S0377-2217(96)00253-6

Salhi S, Wassan N, Hajarat M (2013) The fleet size and mix vehicle routing problem with backhauls: Formulation and set partitioning-based heuristics. Transportation Research Part E: Logistics and Transportation Review 56:22 – 35, DOI http://dx.doi.org/10.1016/j.tre.2013.05.005

Salhi S, Imran A, Wassan NA (2014) The multi-depot vehicle routing problem with heterogeneous vehicle fleet: Formulation and a variable neighborhood search implementation. Computers & Operations Research 52, Part B:315 – 325, DOI http://dx.doi.org/10.1016/j.cor.2013.05.011, recent advances in Variable neighborhood search

Sze JF, Salhi S, Wassan N (2016) A hybridisation of adaptive variable neighbourhood search and large neighbourhood search: Application to the vehicle routing problem. Expert Systems with Applications 65:383 – 397, DOI https://doi.org/10.1016/j.eswa.2016.08.060

Taillard ED (1999) A heuristic column generation method for the heterogeneous fleet vrp. RAIRO - Operations Research 33(1):1–14, DOI 10.1051/ro:1999101

Toth P, Vigo D (1997) An exact algorithm for the vehicle routing problem with backhauls. Transportation Science 31(4):372–385, DOI 10.1287/trsc.31.4.372

Tütüncü GY (2010) An interactive {GRAMPS} algorithm for the heterogeneous fixed fleet vehicle routing problem with and without backhauls. European Journal of Operational Research 201(2):593 – 600, DOI http://dx.doi.org/10.1016/j.ejor.2009.03.044

Tütüncü GY, Carreto CA, Baker BM (2009) A visual interactive approach to classical and mixed vehicle routing problems with backhauls. Omega 37(1):138 – 154, DOI http://dx.doi.org/10.1016/j.omega.2006.11.001

Vidal T, Crainic TG, Gendreau M, Prins C (2014) Implicit depot assignments and rotations in vehicle routing heuristics. European Journal of Operational Research 237(1):15 – 28, DOI https://doi.org/10.1016/j.ejor.2013.12.044