An Adaptive Variable Neighborhood Search for a Heterogeneous Fleet Vehicle Routing Problem with Three-Dimensional Loading Constraints

Abstract—The paper addresses the heterogeneous fleet vehicle routing problem with three-dimensional (3D) loading constraints (3L-HFVRP), a new practical variant of the combined routing and loading problem. In this problem, the loads consist of a set of three-dimensional, rectangular shaped items. The fleet is composed of heterogeneous vehicles with different weight and space capacities. The objective is to serve all customers by selecting a set of vehicles such that the total transportation cost is minimized. The cost consists of the fixed cost of the selected vehicles and their travel cost. In addition, loading sequence related constraints frequently encountered in realistic applications are respected when loading and unloading the items. To solve this challenging problem, we develop an adaptive variable neighborhood search (AVNS) which utilizes an extreme point based first fit heuristic to find a feasible loading pattern for each route. We design two strategies to accelerate the loading and routing processes. The Trie data structure is used to record the loading information of routes already visited and to control the computational effort spent for each route. The Fibonacci heap data structure is used to maintain all of the possible moves and vehicle type assignments, which avoids the duplicated evaluation of some moves and unnecessary loading check of unpromising solutions. The robustness and effectiveness of the proposed algorithm is validated by computational tests performed both on some newly generated 3L-HFVRP instances and well-known benchmark instances from the literature for two simplified VRP variants: the capacitated vehicle routing problem with 3D loading constraints (3L-CVRP) and the pure heterogeneous fleet vehicle routing problem (HFVRP). The numerical experiments show that the proposed AVNS outperforms other algorithms in 3L-CVRP and improves several best known solutions reported in the literature. The results obtained for the pure HFVRP are very close to the best known solutions.

I. Introduction

Loading freights and delivering them to geographically dispersed customers play a central role in the distribution and logistics management as it represents a high proportion of the running costs of companies. The most fundamental of such routing problem is known as the vehicle routing problem (VRP) which describes the distribution of freights from a central depot to a set of customers using a homogeneous vehicle fleet. VRP has been well studied and many variants have been presented. In most of these variants, the loading of freights is simplified so that the geometrical shape of the freight is ignored and only its weight is taken into account. In practice, decision makers usually have to deal with transporting and loading simultaneously, especially if the loading is non-trivial. For example, when an item is stacked on other items, the stacking stability, fragility and ease of unloading for operators should be carefully considered. Thus, the solutions of these variants may not be practical due to the loading issues. Due to the advances in the computational power of modern computers, complex variants that align better with real-life applications have become increasingly popular in the research community. Problems such as the capacitated vehicle routing problem with two dimensional (2D) loading constraints (2L-CVRP) and with three dimensional (3D) loading constraints (3L-CVRP) have been studied. To the best of our knowledge, in all previous variants that combine the VRP and 3D loading constraints, the vehicles are
assumed to be homogeneous and only one type of vehicle is considered. However, transportation using heterogeneous vehicles seems to be a common practice. In industry, as suggested in [1], the fleet of vehicles in a company is rarely homogeneous. They become heterogeneous over time as the company incorporates vehicles of different features into the original fleet. Sometimes a company has to hire vehicles to handle an unusually large-scale transportation task. A mixed fleet of vehicles also provides the flexibility to design a more cost-effective distribution plan.

This paper introduces and addresses an important practical extension of the VRP, the Heterogeneous Fleet of Vehicle Routing Problem with 3D Loading Constraints (3L-HFVRP). In this problem, the depot holds a heterogeneous vehicle fleet, the customer demands are formed by a set of 3D rectangular, weighted items. The objective is to select a set of vehicles and determine the route for each vehicle to serve all of the customers and minimize the total cost. The cost consists of the fixed cost of each selected vehicle and the travel cost determined by the product of the vehicle unit travel cost and the travel distance of this vehicle. The vehicle types are heterogeneous with respect to their capacity, loading space, fixed cost and unit travel cost, and the number of each type vehicle is unlimited. A route is considered feasible only if there is a feasible loading plan for the items demanded by the customers on this route. The loading plan must meet a series of operational constraints that are frequently encountered in real-world applications, such as the stacking stability, fragility, orientation and the last in, first out (LIFO) policy.

Unlike the 3L-CVRP, the vehicle types in the 3L-HFVRP are heterogeneous. Thus, the 3L-HFVRP provides a more practical, accurate, effective strategy for freight transportation. Many real-world applications fit into this integrated model, such as the delivery of beds, night tables, chests of drawers, wardrobes and accessories [2].

The 3L-HFVRP is also of vital importance in theoretical research. It is a combination of two well-known non-deterministic polynomial hard (NP-hard) optimization problems, namely the heterogeneous fleet vehicle routing problem (HFVRP) and 3D bin packing problem (3BPP). Thus the proposed 3L-HFVRP is also NP-hard.

To solve this problem, we develop an adaptive variable neighborhood search (AVNS) method. An extreme point based first fit heuristic is incorporated to find a feasible loading plan for the given routes. More specifically, the savings algorithm is used to generate the initial solution. Then, variable neighborhood search (VNS) is employed to explore
the solution space systematically. When all of the possible moves of VNS cannot improve the solution, a diversification procedure is invoked to adjust the search trajectory. We propose two diversification methods and each time one of the methods is selected probabilistically according to its historical performance. During the procedure, the program only searches within the feasible solution space. A candidate solution is accepted as the new solution only when it is determined as feasible in terms of the loading constraints. To speed up the process, two data structures are creatively used to reduce the calls to the loading check, which is the most time-consuming operator. The Trie data structure is used to record the feasibility information of the routes already visited and store the times checked by our loading heuristic to help control the computational effort spent on that route. The Fibonacci heap data structure is used to maintain all of the possible moves of the current solution and the vehicle type assignments, which avoid calling the loading heuristic for unpromising moves. To show the effectiveness of our approach, the proposed AVNS is adapted and extensively tested on the 3L-CVRP and pure HFVRP instances. The results demonstrate that our algorithm outperforms all of the other methods on the 3L-CVRP, by producing excellent results for all problem instances and even improving some best known results. In addition, our method produces solutions that are quite close to the optimal solutions for the HFVRP instances. Last but not least, we generate many instances for the 3L-HFVRP and provide the results of our algorithm on these instances.

The remainder of this paper is organized as follows. The next section provides an overview of the relevant literature. Section III describes the 3L-HFVRP in detail. In Section IV, the detailed operators of the proposed AVNS are presented. The loading heuristic to produce the loading plan is discussed in Section V. The results for the pure HFVRP, 3L-CVRP and 3L-HFVRP are reported in Section VI. Section VII concludes the paper.

II. Literature Review

VRP has numerous variants [3]. One of the well studied variants is the HFVRP with an unlimited number of vehicles for each vehicle type. This HFVRP can be seen as a special case of the 3L-HFVRP, in which only the weight is considered when loading freight into the vehicle. There are three variants of the HFVRP: the HFVRP-F only considers fixed cost of vehicles and was first proposed by [4], the HFVRP-V only considers the unit travel cost [5], and HFVRP-FV considers both the fixed and travel costs [6]. Many algorithms have been reported for this problem, such as tabu search (TS) [7], hybrid evolutionary local search [8], VNS [9], [10], genetic algorithm [11], memetic algorithm [12], column generation approach [6], and iterated local search combined with set partitioning formulation [13]. Another well-known variant with a heterogeneous fleet fixes the number of vehicles of each type. Recently proposed algorithms include the record-to-record travel algorithm [14], multi-start adaptive memory programming [15], TS [16], and hybrid population heuristic [17].

The loading component of the 3L-HFVRP is closely related to the single container loading problem (SCLP) which aims to load boxes into a container so as to maximize its volume utilization. The SCLP is a particularly difficult problem and has attracted the attention of many researchers. Detailed description of loading heuristics can be found in [18]–[21]. The routing and loading problems have been studied widely but independently. The two problems have only been recently combined. The proposed 3L-HFVRP becomes the 3L-CVRP when the vehicles are homogeneous. The 3L-CVRP was first introduced by [22] and solved by invoking a TS to check the loading subproblem. A guided tabu search (GTS) sharing the advances of TS and guided local search was proposed and a bundle of loading heuristics was used to examine the loading constraints by [23]. The saving-based ant colony optimization (ACO), which makes use of two fast loading heuristics, was developed by [2]. This method is different from others as it allows infeasible solutions during the search by penalizing violations. Recently, a two-phase TS incorporating enhanced loading heuristics was addressed by [24]. Another TS method using the tree search for loading was reported in [25]. Both algorithms obtained excellent results and improved several best known solutions. A hybrid approach combining honey bee mating optimization for routing and six loading heuristics for the loading aspect was introduced by [26]. In [27], the routing subproblem was solved by a greedy randomized adaptive search procedure combined with the evolutionary local search (GRASP × ELS) algorithm. The loading component was examined by relaxing it to a resource constrained project scheduling problem. However, this method could only consider the 3D and orientation constraints, and additional constraints encountered frequently, such as fragility, stability, and LIFO constraints, cannot be dealt with.

The 3L-CVRP generalizes the routing and loading problem 2L-CVRP which assumes that the items cannot be
Each customer \( i (i = 1, \ldots, N) \) requires a set of \( m_i \) rectangular shaped items, denoted as \( I_i \), whose total weight is \( d_i \). Each item \( I_{ik} \in I_i \) has length \( l_{ik} \), width \( w_{ik} \) and height \( h_{ik} \). A fragility flag \( f_{ik} \) is associated with each item \( I_{ik} \), and \( f_{ik} \) is set to 1 if the item is fragile, otherwise \( f_{ik} = 0 \) for a non-fragile item.

We use the Cartesian coordinate system as in Fig. 1, where the length \( W_i \) is parallel to the x-axis, the height \( H_i \) is parallel to the y-axis, and the length \( L_i \) is parallel to the z-axis.

The objective of the 3L-HFVRP is to select a set of vehicles and the routes for those vehicles to service all customers, so that the total cost is minimized. The cost consists of the fixed cost of the selected vehicles and the travel cost of each route. Each vehicle must start and end at the central depot and each customer must be visited exactly once by one vehicle. The total weight of the items in one route should not exceed the capacity of the vehicle and a feasible route must satisfy the following constraints:

### III. Problem Description

Following the notations used in the literature, the 3L-HFVRP is defined on a complete graph \( G = (V, E) \), where \( V = \{0, 1, \ldots, N\} \) is the vertex set containing a central depot (node 0) and \( N \) customers, and \( E = \{(i, j) \mid i, j \in V, i \neq j\} \) is the undirected edge set. Each edge \((i, j) \in E\) is associated with a travel distance \( d_{ij}\) from the vertex \( i \) to \( j \). A fleet of \( T \) different types of vehicles is located at the depot and each type of vehicle \( t (t = 1, \ldots, T) \) has a weight capacity \( D_t \), fixed cost \( F_t \), unit travel cost \( V_t \) and 3D rectangular loading space of length \( L_t \), width \( W_t \) and height \( H_t \). Each vehicle has an opening at the rear door that is as large as the \((W_t \times H_t)\) plane. The number of vehicles for each type is unlimited. As a vehicle of larger capacity usually has higher costs and greater fuel consumption, without a lack of generality, we assume that \( D_1 \leq D_2 \leq \cdots \leq D_T \), \( F_1 \leq F_2 \leq \cdots \leq F_T \) and \( V_1 \leq V_2 \leq \cdots \leq V_T \). The travel cost of each edge \((i, j) \in E\) by a vehicle type \( t \) is \( C_{ij}^t = V_t \times d_{ij} \).

The total cost of a route \( R \) served by a vehicle type \( t \) is defined as \( C_{R}^t = F_t + \sum_{\forall (i,j) \in R} d_{ij} \times V_t \).

![FIGURE 2 An example of the 3L-HFVRP and its loading plan. (a) An example of the 3L-HFVRP, and (b) a possible loading plan for the vehicles in (a).](image-url)
1) **Loading constraint.** Items are loaded with their edges parallel to the sides of the vehicle. Every item has a fixed vertical orientation, but a horizontal 90° rotation is permitted. The length, width, and height of each item cannot exceed the space of the vehicle and there is no overlap between any pair of items.

2) **Support constraint.** A certain percentage $\alpha$ of item’s base must be supported by the vehicle or by other items.

3) **Fragility constraint.** Items can be placed on the top of each other, but non-fragile items cannot be stacked on any fragile items.

4) **LIFO constraint.** For any customer $i$, no items demanded by other customers visited later than $i$ can be placed above any item $I_k$ or between $I_k$ and the rear of the vehicle. Furthermore, each item must be unloaded using only a straight movement parallel to the L-edge.

Fig. 2 gives an example of 3L-HFVRP with two routes and a possible loading plan for these two routes.

We evaluate the influence of different loading constraints on the final solution by removing one or more constraints at a time and the results are reported in the section on computational experiments.

**IV. Proposed Solution Approach**

As the 3L-HFVRP is an NP-hard problem, it is impossible for exact methods to solve large instances in the real-world setting within reasonable amount of time. We focus on meta-heuristic methods to provide near-optimal solutions within reasonable computational time. Motivated by the successful application of the VNS scheme to VRP variants in [9], [38], [39], we use the VNS algorithm with an adaptive diversification procedure to explore the solution space thoroughly. For every candidate solution, an extreme point based first fit heuristic is developed to examine whether it has a feasible loading plan. The pseudo-code of proposed adaptive variable neighborhood search is outlined in Algorithm 1.

**Algorithm 1: The Adaptive Variable Neighborhood Search for the 3L-HFVRP.**

```plaintext
AVNS()
1 Generate an initial solution $S$ via the savings algorithm
2 Define a set of operators in the shake procedure $\text{NS}$
3 $S^* = S$, $\text{nonImp} = 0$
4 while time limit is not exceeded
5 $i = 1$
6 while $i \leq |\text{NS}|$ // attempt all of the shake procedures gradually
7 $K = \text{number of vehicles used in } S$
8 repeat $\min\{10, K\}$ times // better explore the neighboring space
9 shake $S$ to get a random point $S'$ with $\text{NS}$
10 $S' = \text{LOCALSEARCH}(S')$
11 if $S'$ is better than $S$
12 $S = S'$, $S^* = S'$, $i = 0$, $\text{nonImp} = 0$
13 break the loop
14 $i = i + 1$
15 $S = \text{DIVERSIFY}(S^*, \text{nonImp})$
16 $\text{nonImp} = \text{nonImp} + 1$
17 return $S$
```

An initial solution is generated by the savings algorithm and the set of shake procedures is defined. At each iteration of the VNS, the current solution is shaken with a shake procedure (Line 9) and improved by the local search (Line 10). If the obtained solution is better than the best solution, it is recorded as the best solution (Line 11). Our algorithm differs from the classical VNS as it shakes the current solution up to $\min\{10, K\}$ times, where $K$ is the number of vehicles used in the current solution, unless a better solution has been found (Line 8). This helps to better explore the neighboring search space of the current solution. If all the shake procedures are tried and no improving solution is found, an adaptive diversification procedure is invoked to perturb the best found solution so that the process can escape from the local optimum (Line 15). The VNS restarts its search on the solution generated by the diversification procedure. The above steps are repeated until the given running time elapses. The variable $\text{nonImp}$ records the number of successive calls of $\text{DIVERSIFY}$, based on whose solution the VNS failed to find an improving solution. We use $\text{nonImp}$ to control the strength of the perturbation in $\text{DIVERSIFY}$. The detailed presentation of each step is provided in the following subsections.

**A. Initial Solution Construction**

Clark and Wright’s savings algorithm [40] is adapted to construct the initial solution. Each customer is assigned to a route which can fully accommodate the customer’s demand and has the most economical cost. The algorithm iteratively merges two routes into one until no more feasible improving merges exist. For any two routes, all possible merges are attempted. These include merging two routes directly, merging the first route and the reverse of the second route, merging the reverse of the first route and the second route, and merging two reversed routes. The merge that results in the largest savings is performed. Savings is calculated as the total costs of two original routes minus the total cost of the new route. A merging is accepted only if the resulting route is feasible in terms of the loading constraints and the most suitable vehicle with the lowest cost is assigned to each new route. The obtained solution is thus always loading feasible.

**B. Shake Procedure**

The shake procedure is to enable the better exploration of the search space by generating various starting points for the local search procedure. The shake procedure has six different operators.

For the basic operator $\text{SSE2}$, two routes are randomly selected. For the first route, we randomly select a segment of two customers to be exchanged with a segment of two customers in the second route by traversing the list of customers in the second route from the front of the list to the back of the list. This is done systematically until a feasible solution is found. For the resulting routes, each vehicle type is examined, the vehicle that has the lowest cost and enables a feasible loading plan is selected.
For the basic operator SSE3, we choose a segment length of three instead of two. For the basic operator VSE, we randomly choose the segment length from one to three in the first route and systematically look for the first feasible exchange of with the second route. The length of segment used in the second route is between one to three. It is quite evident that it is harder to exchange segments that are longer than shorter. As such, we restrict our segment length to be at most three.

If we perform the SSE2, SSE3, and VSE once, the operators are called SSE21, SSE31, and VSE1, respectively. If we do it twice, the operators are then known as SSE22, SSE32, and VSE2, respectively. As such, we have six different non-basic operators from which the shake procedure can randomly choose to perform diversification.

C. Local Search Procedure

The local search procedure aims to improve the solution generated by the shake procedure. It consists of three simple but effective basic moves [30], [36], whose details are presented below:

1) customer-swap: Swap the positions of a pair of customers from the same route or different routes. Four old edges are deleted and four new edges are introduced.

2) customer-shift: A customer is erased and shifted to another place in the same route or a different route. Three edges are changed.

3) route-interchange: It can be performed within any single route or any route pair. Within a given route (also called 2-opt), two original edges are deleted and the middle segment is reversed. Two new edges are introduced to connect the segments as a route. It is useful to eliminate intersections within a route. When an intersection occurs between a route pair, one edge from each route is eliminated to cut the route at the starting part and terminating part, then the starting part of the first route is concatenated with the terminating part of the second route, and vice versa.

For each resulting new route, the suitable vehicle with the lowest cost is assigned. An empty dummy route is maintained for operating flexibility. For example, it is easy to assign a single customer to a new route or split a route into two.

Our local search method uses the best admissible decent strategy to systematically explore the solution space. In the beginning, the three move types are marked as available; then, the following iterations are repeated. In each iteration, one of the available move types is randomly selected. Tentative moves of the chosen type are exhaustively examined and the most improving feasible move is identified and performed on the candidate solution. If no improving move is found, the chosen type is marked unavailable in the next iteration; otherwise, all three types are marked as available. The iterations repeat until there are no available types, i.e., no move of any type can further improve the solution. It is easy to see that the complexity of each move type is \(O(N^2)\). It is time-consuming to examine the loading feasibility of every move to locate the best feasible move. We accelerate the procedure with the help of the Fibonacci heap, which is a special priority queue structure with fast insertion, deletion, update and minimum retrieval capabilities [41]. For each move type, all of the possible moves and the possible vehicle type assignments for the resulting routes are stored in a Fibonacci heap. They are sorted by the corresponding objective cost in ascending order. However, the loading feasibility of these moves is not checked. Every time the most improving feasible move is the need for one move type, we just repeat to get the move in the top of the heap and check its loading feasibility until we meet a feasible move. According to the property of Fibonacci heap, this move is the feasible move with the lowest cost. When a move is executed, only a limited set of moves are affected. Assuming the performed move is related to two routes, \(R_1\) and \(R_2\), only the moves involved with any customer belonged to \(R_1\) or \(R_2\) must be updated.

The use of Fibonacci heaps can help to prevent many unnecessary calls in the loading check procedure. Once a feasible move is found at the top of the heap, the loading feasibility checks for the rest of the moves in the heap are canceled. This strategy reduces the computational time dramatically because the loading check is the most time-consuming operator.

D. Adaptive Diversification Mechanisms

After the local search method terminates at a local optimum, the diversification procedure DIVERSIFY is called to adjust the search trajectory. We concentrate on a promising space by perturbing the best found solution to obtain a new initial solution. We propose two mechanisms the ruin-reconstruct (RR) and concat-split (CS) to diversify the search process. One mechanism is selected each time according to their historical performances.

Algorithm 2: The Adaptive Diversification Procedure.

```plaintext
DIVERSIFY (\(S\), nonimp)  
1 // u is used to record the selected approach, 1 indicates RR and 2 indicates CS
2 // callNumber stores the number of times that mechanism i performed
3 // impNumber records the number of times that mechanism i leads to a better solution
4 // If the best found solution \(S'\) has been updated
5 impNumber\(_i\) = impNumber\(_i\) + 1 // update the corresponding value
6 Evaluate the score\(_i\) = \(1 + \text{impNumber}\(_i\)\) / callNumber\(_i\) for approach 1 and 2
7 Calculate the probability \(p_i = \text{score}\(_i\) / \text{score}\(_1\)\) for approach 1 and 2
8 Select an approach probabilistically according to \(p_1\) and \(p_2\) and set the value \(u\)
9 callNumber\(_e\) = callNumber\(_e\) + 1 // update the corresponding call times
10 if \(u == 1\)
11 \(S = RR(S', \text{nonimp})\)
12 else
13 \(S = CS(S', \text{nonimp})\)
14 return \(S\)
```
The pseudo-code of the diversification procedure is provided in Algorithm 2. Two variables, `callNumber`, and `impNumber`, are used to record the historical performance for mechanism $i$ (1 for RR, 2 for CS). They are initialized as 0, then `callNumber`, increases by one whenever the mechanism $i$ is performed (Line 9), and `impNumber`, increases by one if a new best solution is obtained based on the initial solution generated by mechanism $i$ (Lines 4–5). A score of 5 is evaluated (Line 9), and increases by one whenever the mechanism $i$ is performed. Our diversification procedure chooses mechanism $i$ to perform with the probability $p_i$, which is calculated in Line 6. This diversification strength increases with the minimum incremental cost and the corresponding vehicle is assigned to the route. If the current customer cannot be served by an existing vehicle, a new suitable vehicle is added. The rebuilding process terminates when a complete solution is generated.

It becomes increasingly difficult to escape from a local optimum during the process. The diversification strength increases with the `nonImp`, which is the number of successive iterations in which no improving solution is found based on the solution generated by DIVERSIFY (see Algorithm 1 for details). Excessive diversification should be avoided to retain the good elements of the original solution. The number of the erased customers is defined as $\min(0.5 \times N, 0.1 \times N + \text{nonImp})$, where $N$ is the number of total customers.

2) Concat-Split approach: The concat-split (CS) approach is based on the principle of concatenating the 3L-HFVRP solution into a giant tour that is a sequence of customers, like the classical traveling salesman problem. The edge with the highest cost is eliminated to obtain a new sequence, followed by the optimal split procedure to convert the giant tour back into the 3L-HFVRP solution. The split approach has been proven to be effective for numerous vehicle routing problems [43], [12], [32]. The proposed CS method takes advantages of this approach and is tuned to diversify our search process. Our CS is different from the original split approach as the stage of erasing the expensive edge is introduced with the aim of identifying and eliminating the poor features from the current solution. It is executed as the following steps.

Step (1), Concat: To connect the routes as a giant tour, we should consider introducing low-cost edges and bring the diversification. First, a route is randomly selected and put into
the empty initial giant tour. Then, other routes are iteratively selected and connected at the end of giant tour until all the routes are in the giant tour. In each iteration, the unlinked routes are sorted by increasing distance between the last customer of the giant tour and the first customer of route, and the route ranked at the $i$th place is chosen with the probability 

$$p_i = 2(M-i+1)/M(M+1) \quad (M \text{ is the number of unlinked routes})$$

so that a route with a higher rank has a higher probability of being selected. The final obtained customer sequence is our original giant tour.

Step (2), Change: If we perform the split procedure on the giant tour obtained in Step (1) directly, the generated new solution might be a better one in terms of cost, but in most cases it is the same as the original solution from our preliminary experiments. Thus, it is necessary to alter the giant tour before splitting it.

We alter the giant tour through eliminating the most expensive edge based on the fact that long edges are usually not contained in the optimal solution. Note that the nodes of this edge should not be the end points of different routes. In particular, we pick the edge $(i, j)$ that maximizing the utility function $U(i, j) = (d_{ij} / \text{avg}_{ij}) / (1 + e_{ij})$ used in [30], where $e_{ij}$ is the times of edge $(i, j)$ eliminated, and $\text{avg}_{ij}$ is the average distance of all edges starting from nodes $i$ and $j$. This function helps select the edge with relatively large distance compared to its neighborhoods and having been eliminated for few times. Therefore it leads to a more balanced edge selection than simply choosing the longest edge. After the edge is eliminated, the giant tour is cut into two parts. We relink the first part after the last part such that the two nodes $i, j$ are the two end points of the resulting giant tour. In addition, we also randomly select nonImp/2 pairs of customers and exchange their positions in order to introduce more diversification.

Step (3), Split: The split procedure first constructs an auxiliary graph $H = (X, A, Z)$ with the given giant tour as shown in the Fig. 3, where $X$ is a set of $N+1$ nodes indexed from 0 to $N$ with node 0 as a dummy node, $A$ represents the arc set, and $Z$ provides the cost of arcs. The nodes $1, \ldots, N$ correspond to the customer sequence in the given giant tour $T = (v_1, …, v_N)$. One arc $(i, j)$ is created only if the route with customers $(v_{i+1} \ldots v_j)$ can be feasibly served by at least one type of vehicle in terms of loading constraints. And the most economical vehicle type is assigned to this route. So its weight is the route cost $c_R$ defined in Section III.

An optimal split corresponds to a min-cost path from node 0 to $N$ in $H$, which can be computed using Dijkstra’s algorithm. Building the graph and solving the shortest path problem can be done in $O(N^2)$ time. In practice, the required time is much less because of the tight loading constraints which dramatically reduce the number of arcs. For more detailed description and the pseudo-code of the split procedure, the reader is referred to [43] and [32].

Step (4), Extract: Each arc in the auxiliary graph $H$ corresponds to a feasible route. This step converts the arcs of the optimal path into routes to obtain a new solution.

<table>
<thead>
<tr>
<th>Algorithm 3: RandomLSPack Procedure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RandomLSPack ($R, t$)</td>
</tr>
<tr>
<td>1 Let $n$ be the number of items in $R$</td>
</tr>
<tr>
<td>2 if $n \leq \text{max_enumeration_num}$ then</td>
</tr>
<tr>
<td>3 $I$ = all of the items demanded by the customers in $R$</td>
</tr>
<tr>
<td>4 for each permutation $I'$ of $I$</td>
</tr>
<tr>
<td>5 if FirstFitPack($I', t$) $\leq L,$</td>
</tr>
<tr>
<td>6 return $\text{success}$</td>
</tr>
<tr>
<td>7 else</td>
</tr>
<tr>
<td>8 for each sorting rule</td>
</tr>
<tr>
<td>9 $I$ = sort sequence of all of the items demanded by the customers in $R$</td>
</tr>
<tr>
<td>10 $L$ = FirstFitPack($I$, $t$) // the length used to load all of the items</td>
</tr>
<tr>
<td>11 if $L \leq L_t$</td>
</tr>
<tr>
<td>12 return $\text{success}$</td>
</tr>
<tr>
<td>13 for $k$ = 1 to $n$ // attempt with other sequences to load the items</td>
</tr>
<tr>
<td>14 Generate new sequence $I'$ by swapping two randomly selected items in $I$</td>
</tr>
<tr>
<td>15 $L'$ = FirstFitPack($I'$, $t$)</td>
</tr>
<tr>
<td>16 if $L' \leq L$</td>
</tr>
<tr>
<td>17 $I = I'$, $L = L'$</td>
</tr>
<tr>
<td>18 return $\text{success}$</td>
</tr>
</tbody>
</table>

Fig. 3 illustrates the proposed concat-split procedure for a 3L-HFVRP instance with six customers and two vehicles of type 3. Two routes are concatenated to obtain the giant tour. The most expensive edge (2, 3) is eliminated and a new giant tour $T (3, 4, 5, 6, 1, 2)$ is generated. The auxiliary graph is constructed and the optimal split is obtained via the shortest path algorithm. The three corresponding routes are extracted and a new diversified solution is generated.

An alternative method can be used at step (2). To eliminate the edge $(i, j)$ with the maximum utility function, the nodes $i, j$ are erased and reinserted after their nearest neighbors. These two step (2) methods are randomly chosen between with equal probability when concat-split mechanism is called.

V. The Extreme Point Based First Fit Heuristic for the Loading Subproblem

We use the procedure RandomLSPack to check the loading feasibility of a route. The RandomLSPack procedure is given in Algorithm 3. It takes a route tour $R$ containing the ordered customers visited by a vehicle and a parameter $t$ representing the vehicle type as inputs. If the procedure finds a feasible loading for this vehicle, it reports a const success, otherwise, it returns a const failure. We use different strategies based on $n$, which is the number of items required by the customers in $R$.

If $n$ is not larger than the user defined parameter max\_enumeration\_num (set to 8 in our implementation), we try all of the permutations of $I$ and call the process FirstFitPack (see Section V-A for details) on each permutation. FirstFitPack regards the length dimension of the vehicle as open and tries to pack all of the items into the open vehicle such that the used length of the vehicle is minimized. If the FirstFitPack
finds a loading pattern whose used length is less than the vehicle length \( L_v \), success will be returned. Otherwise, failure is returned.

If \( n \) is larger than the user defined parameter \( \text{max\_enumeration\_num} \), a random local search procedure is used to test some of the permutations of \( I \). We use two sorting rules to generate an initial sequence \( I \) of the items demanded by the customers in \( R \) (line 10). The first sorting rule sorts the items in the following steps:

1) If the LIFO constraint is not enforced, go to the next step. Otherwise, sort the items in the reverse visiting order of the corresponding customer and break ties using the rule in the next step.

2) If the fragility constraint is not enforced, go to the next step. Otherwise, sort the items so that the non-fragile items are before the fragile items and break ties using the rule in the next step.

3) If the support constraint is not enforced, go to the next step. Otherwise, sort the items by decreasing order of the base area, i.e., \( l_i \times w_i \) for item \( i \), and break ties using the rule in the next step.

4) Sort the items by decreasing order of volume.

The second sorting rule is the same as the first rule except that step 3 is replaced by the following step:

3) Sort the items by decreasing order of their length and break ties using the rule in the next step.

For each sorted sequence of items and a vehicle type, we use a heuristic loading procedure FirstFitPack to try to pack the items into the vehicle surface. If FirstFitPack finds a loading pattern whose used length is less than the vehicle length \( L_v \), success will be returned. Otherwise, we attempt to optimize the solution in terms of the used length through a random local search mechanism. The number of iterations is limited to the number of items \( n \) (lines 15–21). In each iteration, a new sequence is generated by swapping the positions of two randomly selected items and FirstFitPack is called on the new sequence. If the resulting length is less than the length of the old solution, we set our sequence to the new sequence. At any point in the process, if a loading pattern whose used length is less than the vehicle length is found, the procedure halts with success. If this does not occur, a failure will be returned at the end of the procedure.

To speed up our procedure, we use the data structure Trie to keep track of the loading feasibility of the routes already examined for a given vehicle type. Similar strategies are also used by other researchers, such as the tree structure [30], pool [2] and hash table [35]. However, our Trie stores not only the feasibility information of the routes, but also the number of times that RandomLSPack is called on the same route and vehicle type. Given a sequence of customers \( R \) representing a route and the vehicle type \( t \) to determine whether the route is feasible, we retrieve the information from Trie. If the stored feasibility information is success or the number of times that RandomLSPack is called on this route and the vehicle type reach the maximum allowed number \( \text{max\_call\_time} \), we just return the stored feasibility information. Otherwise, we call RandomLSPack again and update the feasibility information. The calling number is updated as follows. If the number of items demanded by the customers in \( R \) is not larger than \( \text{max\_enumeration\_num} \), all of the permutations of these items have been enumerated, there is no need to call RandomLSPack on this route again, and we set the calling number to \( \text{max\_call\_time} \). Otherwise, we increase the calling number by one. In our implementation, we set the maximum calling time \( \text{max\_call\_time} \) to \( \max(n, 100 \times (1 - V_i / (L_v \cdot W_i \cdot H_i))) \), where \( n \) is the number of items, \( V_i \) is the volume of all of the items and \( L_v \cdot W_i \cdot H_i \) is the volume of the vehicle surface. The larger the ratio between the item volume and vehicle volume, the fewer times RandomLSPack can be called. The larger this ratio is, the lower the probability that a feasible loading solution exists for this route so we spend less effort on this route.

The Trie can save time, in that, if RandomLSPack has found a feasible loading for a route, the stored information in Trie will avoid unnecessary calls of RandomLSPack on the same route later. We also limit the maximum calling number of RandomLSPack on the same route and vehicle type so that not too much time will be spent on routes without feasible loading plans.

FIGURE 4 The method to generate new extreme points.

FIGURE 5 Comparison of the performance of the AVNS and BKS on instances of the pure HFVRP.
A. Extreme Point Based First Fit Loading Heuristic

Our FirstFitPack procedure is given in Algorithm 4. It takes as input a sequence of items $I$ and a parameter $t$ representing the vehicle type. In the loading procedure, we regard the length dimension of the vehicle as unlimited and try to load all the items into the vehicle according to the given sequence so that the used length is minimized.

FirstFitPack places the items in $I$ one by one. FirstFitPack maintains a list of extreme points [44] that are the candidate positions to place an item in. Initially, the empty vehicle is represented by one extreme point $(0, 0, 0)$ (line 2). When a new item is placed, it occupies one extreme point and up to six new extreme points are introduced. The method to introduce the six extreme points is shown in Fig. 4. Assume that the corner of the item closest to the origin $O$ is $A$ and that the three adjacent corners of $A$ are $B$, $C$, and $D$. From each of $B$, $C$, and $D$, we can extend two rays along the opposite direction of the axis (given by $B_1$, $B_2$, $C_1$, $C_2$, $D_1$, and $D_2$). The place where the ray first intersects the surface of either another packed item or the vehicle is an extreme point.

At each step of our loading process (lines 4–9), we sort the list of extreme points. We check each extreme point in the sorted order. For each extreme point $p$, we try to place the first unpacked item in the sequence at this point. If the placement is feasible, we place it at $p$, update the extreme point list and proceed to the next step. Otherwise, we try the next point in the list. If all of the extreme points are examined and no feasible placement for the first unpacked item is found, the process terminates with an infinite length. The above process is repeated until all of the items are placed. The used length of the vehicle is returned at the end of the procedure.

VI. Computational Results

This paper is the first to introduce the 3L-HFVRP. To test the effectiveness of our approach in terms of routing, we first test the AVNS on the pure HFVRP instances. We then test the AVNS on 3L-CVRP instances. We generate a set of new 3L-HFVRP instances and report the results obtained by the AVNS. Our algorithm is coded in C++, and all of the experiments are executed on an Intel Xeon E5430 with a 2.66 GHz (Quad Core) CPU and 8 GB RAM running the CentOS 5 Linux operating system. All of the problems are assigned a runtime limit: 900 CPU seconds for $N \leq 25$, 1800 CPU seconds for $25 < N \leq 50$ and 3600 CPU seconds for $N \geq 50$, where $N$ is the number of customers. As in the existing literature on
the 3L-CVRP, the support ratio \( a \) is set at 75%. The user defined parameter max\_enumeration\_num used in RandomLS-Pack is set to eight after preliminary tests. The AVNS experiments are run 30 times by setting the random seed from 1 to 30 unless explicitly stated. All of the benchmark instances used in this paper and the detailed computational results, including routing and loading plans, can be downloaded from [45].

A. Comparison on Pure HFVRP

To test the performance of the AVNS for routing, we first test the AVNS on pure HFVRP instances and compare it with the best known solutions (BKS) of recent algorithms such as CG [6], SMA\_D1 [12], VNS1 [9], ILS\_RVND [46], and ILS\_RVND\_SP [13]. The instances are classified into three categories according to the cost function: with fixed cost only (HFVRP-F, \( F = 1 \)), with travel cost only (HFVRP-V, \( F = 0 \)) and with both fixed and travel costs (HFVRP-FV). The percentage gap between the best results obtained by the AVNS and BKS is depicted in Fig. 5. Almost all of the BKS have been proven to be optimal except two instances (instance 20 of HFVRP-F and HFVRP-FV). Although our algorithm is not tailored for the pure routing problem and does not use the complicated moves used in other approaches, the obtained results are quite satisfying. Our algorithm performs quite well for the HFVRP as the gaps between the best results obtained by the AVNS and the BKS are limited to about 0.2% for all instances except one. The detailed comparison results are presented in Table 1 of the supplement material which can be downloaded from our website [45].

B. Results for 3L-CVRP

We investigate the effectiveness of the proposed algorithm by adapting our algorithm to test on the instances of the 3L-CVRP. The 3L-CVRP is a special case of the 3L-HFVRP as only one type of vehicles is used. The fixed cost is not considered and the unit travel cost is set to one. 3L-CVRP is slightly different from our 3L-HFVRP because 3L-CVRP has a fixed number of available vehicles. To obtain a solution subject to the vehicle number limit, we adjust the objective function by giving a very large penalty (1,000,000 in our implementation) to each extra vehicle. The benchmark instances contain 27 instances generated in [22] and 12 larger scale instances introduced by [23]. The results are compared with other algorithms reported in the literature, TS [22], GTS [23], ACO [2], DMTS [24], VRLH1 [25] and HA [26]. The computational environments for these approaches are summarized.
in Table 2 of the supplement material. The ACO, DMTS and VRLH1 were run 10 times, here our AVNS runs 30 times. And we compare the average cost obtained for each instances during the runs. As our approach is not designed for 3L-CVRP, we do not limit the number of used vehicles in the initial solution construction and the VNS search procedure. As a result, the solutions for instances 9 and 12 in set 2 found by the AVNS used one more vehicle than the limit. Thus, in order for comparison, we exclude these two instances from set 2 in the following comparison.

Fig. (6a) and Fig. (6b) compare the average cost obtained by the AVNS and other existing algorithms on the set 1 and 2 respectively. In addition, in order to test the effect of each constraint on the final solution, we also test different versions by removing one or more constraints. From these two figures, we can see that the average cost incurred by the AVNS is smaller than all the existing approaches for each test version. The constraint affecting AVNS the most is the LIFO constraint and the least is the fragility constraint. The average costs found by each algorithms on these two sets are given in Table 1; and the details of the cost on each instance are given in Tables 4, 5, 6 and 7 of the supplement material which can be downloaded from [45].

To investigate the convergence behavior of our AVNS, we perform an additional experiment on these instances by setting the random seed to 1. For each instance, we record the solution found by the AVNS at the time of 1, 2, 4, ..., 2048, 3600s. The results are shown as Fig. 7, where we divide the instances into groups based on the number of customers N. For each group, we plotted the gap between the average cost and the final average cost found by the AVNS as the computational time increases. More specifically, for each group, assume the final average cost found by AVNS is c_f and the average cost found at time t is c_t, then the gap (%) is defined as 100 × (c_f/c_t − 1). We can see from these figures that our AVNS exhibits rapid convergence in most of these instances. For the small instances with N ≤ 25, the AVNS converges after 512s on all instances; for the medium instances with 25 < N ≤ 50, most of the instances found its best solutions before 2048s; for the instances with N ≥ 50, our AVNS converges near 3600s on the majority of the instances.

### C. Results for 3L-HFVRP

To our knowledge, this is the first exploration of the 3L-HFVRP. We first present our method to generate the instances of 3L-HFVRP, which is similar to the method used in [22] for the 3L-CVRP. The benchmarks are derived from the 36 Euclidean HFVRP instances used in Section VI-A by adding the 3D attribute to the items and vehicles and retaining the other attributes. A standard loading space is set as L = 120, W = 50 and H = 60. For each vehicle type t (t = 1, ..., T), a random q_t is uniformly selected from the interval [0.7, 1.3] and the corresponding loading space is defined as L_t = q_t × L, W_t = q_t × W and H_t = q_t × H. The order q_1 < q_2 < ... < q_T is maintained. For each customer i, a number m_i of requested items is uniformly and randomly generated from 1 to 3. For each item I_i, its length l_i, width w_i and height h_i are uniformly and randomly generated from the corresponding intervals [0.2L, 0.6L], [0.2W, 0.6W] and [0.2H, 0.6H], respectively. In addition, every item has a 20% probability of being fragile.

Table 2 provides the average results for the various 3L-HFVRP versions with different constraints, where the best cost c_best, average cost c_avg and the average time to find the best cost t_eth(s) over 30 runs are reported. For each version, with some constraints removed, we also calculate the improvement over the fully constrained version in terms of the average best cost and average cost. The results are shown in the columns gap (%) For example, excluding fragility improves the average best cost by 100 × (4662.81/4796.70 − 1). According to the results, the LIFO is the hardest constraint. Excluding the LIFO constraints improves the average cost by up to 6.35%, while fragility, being the least influential constraint, results in an improvement by only 2.51% with its exclusion. When only the 3D-loading constraint is imposed, the average cost improves by up to 8.91%. The detailed results are provided in Table 9 of the supplement material which can be downloaded from our website [45].

The investigation of convergence behavior of our AVNS on the 3L-HFVRP is performed as that for the 3L-CVRP, and the results are depicted in Fig. 8. According to the Figure, the algorithm converges fast for small scale instances, finding

### Table 1 Comparison of the average cost between AVNS and existing approaches on 3L-CVRP instances set (Set 2 excluding instance 9 and 12).

<table>
<thead>
<tr>
<th>SET</th>
<th>CONSTRAINTS</th>
<th>TS</th>
<th>GTS</th>
<th>ACO</th>
<th>DMTS</th>
<th>VRLH1</th>
<th>HA</th>
<th>AVNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ALL CONSTRAINTS</td>
<td>1042.26</td>
<td>997.18</td>
<td>966.67</td>
<td>962.08</td>
<td>953.79</td>
<td>960.10</td>
<td>942.12</td>
</tr>
<tr>
<td>1</td>
<td>NO FRAGILITY</td>
<td>1014.49</td>
<td>965.14</td>
<td>945.04</td>
<td>941.08</td>
<td>934.38</td>
<td>935.92</td>
<td>920.00</td>
</tr>
<tr>
<td>1</td>
<td>NO SUPPORT</td>
<td>939.53</td>
<td>934.96</td>
<td>919.69</td>
<td>913.43</td>
<td>902.55</td>
<td>903.18</td>
<td>892.04</td>
</tr>
<tr>
<td>1</td>
<td>NO LIFO</td>
<td>951.19</td>
<td>950.59</td>
<td>916.25</td>
<td>887.71</td>
<td>912.05</td>
<td>912.47</td>
<td>874.32</td>
</tr>
<tr>
<td>1</td>
<td>LOADING ONLY</td>
<td>876.31</td>
<td>876.39</td>
<td>856.66</td>
<td>846.44</td>
<td>864.54</td>
<td>858.07</td>
<td>842.13</td>
</tr>
<tr>
<td>2</td>
<td>ALL CONSTRAINTS</td>
<td>–</td>
<td>2951.96</td>
<td>–</td>
<td>2883.81</td>
<td>–</td>
<td>2895.63</td>
<td>–</td>
</tr>
<tr>
<td>2</td>
<td>NO FRAGILITY</td>
<td>–</td>
<td>2915.83</td>
<td>–</td>
<td>2850.90</td>
<td>–</td>
<td>2792.34</td>
<td>–</td>
</tr>
<tr>
<td>2</td>
<td>NO SUPPORT</td>
<td>–</td>
<td>2837.74</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>2799.87</td>
<td>–</td>
</tr>
<tr>
<td>2</td>
<td>NO LIFO</td>
<td>–</td>
<td>2807.83</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>2799.87</td>
<td>–</td>
</tr>
<tr>
<td>2</td>
<td>LOADING ONLY</td>
<td>–</td>
<td>2733.35</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>2729.87</td>
<td>–</td>
</tr>
</tbody>
</table>

### Table 2 The summary results of AVNS on 3L-HFVRP instances.

<table>
<thead>
<tr>
<th>CONSTRAINTS</th>
<th>c_best</th>
<th>gap_best</th>
<th>c_avg</th>
<th>gap_avg</th>
<th>t_eth(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL CONSTRAINTS</td>
<td>4796.70</td>
<td>–</td>
<td>4915.70</td>
<td>–</td>
<td>1895.1</td>
</tr>
<tr>
<td>NO FRAGILITY</td>
<td>4662.81</td>
<td>–2.79</td>
<td>4792.29</td>
<td>–2.51</td>
<td>1804.4</td>
</tr>
<tr>
<td>NO SUPPORT</td>
<td>4555.71</td>
<td>–5.02</td>
<td>4667.12</td>
<td>–5.06</td>
<td>1871.3</td>
</tr>
<tr>
<td>NO LIFO</td>
<td>4482.16</td>
<td>–6.56</td>
<td>4603.71</td>
<td>–6.35</td>
<td>1835.9</td>
</tr>
<tr>
<td>LOADING ONLY</td>
<td>4272.08</td>
<td>–10.94</td>
<td>4369.51</td>
<td>–8.91</td>
<td>1620.5</td>
</tr>
</tbody>
</table>
We introduce a new variant of the integrated routing and loading problem, 3L-HFVRP, and generate benchmark instances for future studies. The vehicle fleet is heterogeneous in capacity, loading space, and fixed and variable costs. The number of each type of vehicle is unlimited. The customer demand consists of 3D, rectangular, weighted items. It is a challenging NP-hard problem and has many practical applications.

We develop a hybrid VNS algorithm to intensively explore the solution space and an adaptive diversification mechanism is incorporated to adjust the search trajectory. An extreme point-based first heuristic is frequently invoked to produce the loading pattern for the candidate solutions. And the loading is the most time-consuming operator, the data structures Trie and Fibonacci heap are creatively used to speed up the process, which dramatically reduces the loading check for the routes.

The proposed algorithm was extensively tested on instances of the pure HFVRP, 3L-CVRP and 3L-HFVRP. For the pure HFVRP the obtained results are quite close to the optimum. The AVNS outperforms the existing algorithms and improves several best known results for the 3L-CVRP.

References


