Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)

A Hybrid Genetic Algorithm for the Vehicle Routing Problem with Roaming Delivery Locations

Quang Anh Pham, Minh Hoàng Hà *, Duy Manh Vu, Huy Hoang Nguyen,

ORLab, Faculty of Computer Science, Phenikaa University, Hanoi, Vietnam
hoang.haminh@phenikaa-uni.edu.vn

Abstract

The Vehicle Routing Problem with Roaming Delivery Locations (VRPRDL) is a variant of the Vehicle Routing Problem (VRP) in which a customer can be present at many locations during a working day and a time window is associated with each location. The objective is to find a set of routes such that (i) the total traveling cost is minimized, (ii) only one location of each customer is visited within its time window, and (iii) all capacity constraints are satisfied. To solve the problem, we introduce a hybrid genetic algorithm which relies on problem-tailored solution representation, mutation, local search operators, as well as a set covering component exploring routes found during the search to find better solutions. We also propose a new split procedure which is based on dynamic programming to evaluate the fitness of chromosomes. Experiments conducted on the benchmark instances clearly show that our proposed algorithm outperforms existing approaches in terms of stability and solution quality. We also improve 49 best known solutions of the literature.

Introduction

E-commerce is the process of purchasing or selling goods electronically through online platforms or the Internet and has grown rapidly, especially during the last decades. In 2020, retail e-commerce sales worldwide reached about 4.28 trillion US dollars, and e-retail revenues are estimated to hit over 6.38 trillion US dollars in 2024 (Coppola 2021). This rapid growth of e-commerce opens many challenges for last mile logistics, creating a variety of variants of the Vehicle Routing Problem (VRP), one of the most studied problems in the field of Operations Research (Irnich, Toth, and Vigo 2014). In the traditional VRP-based transportation model, customers are supposed to wait at home or a specific location to receive their orders. However, in practice, they are likely not in the same address during the planning horizon. The customers can appear and get the orders in different locations, e.g., home, workplace, school, mall, church, and so on. Therefore, we can deliver goods to a trunk of customer's car instead of the customer's home. This helps complete the delivery process for customers regardless of their current location as long as they arrive there by their car. This service option is referred to as "*in-car*" delivery or "*trunk*" delivery which is provided by Amazon (Harbison 2018). Several auto manufacturers also support this technology, such as Volvo and General Motors (J. Hawkins 2018).

Recently, a new variant of the VRP called the Vehicle Routing Problem with Roaming Delivery Locations (VRPRDL) is introduced for modeling this delivery strategy by (Reyes, Savelsbergh, and Toriello 2017) who show that applying trunk delivery results in reducing the total traveling distance compared to the home delivery. The VRPRDL is formally defined as follows. Let $G = \{N, A\}$ be a complete directed graph with the set of vertices $N = \{0, 1, ..., n\}$ and the set of arcs $A = \{(i, j) : i, j \in N, i \neq j\}$. Vertex 0 is the depot and other vertices represent locations at which customers can appear. Each arc $(i, j) \in A$ is associated with a travel time t_{ij} and a cost c_{ij}. The triangle inequality is satisfied for both the travel time and cost. We denote that P is the set of customers requiring a delivery during the planning period $[0, T]$. At the depot, there is a homogeneous fleet of vehicles available for delivery tasks; each has a capacity of Q. The delivery for customer $p \in P$ is characterized by a demand quantity d_p and a geographical profile which specifies where and when a delivery can be made. Let $N_p \subseteq N$ denote the cluster of locations that customer p will visit during the planning horizon. Each location $i \in N_p$ has a non-overlapping time window $[e_i, l_i]$ during which the customer can be served. By duplicating locations to handle the case where two customers can appear at the same location, we may assume $N_p \cap N_{p'} = \emptyset$ for two different customers $p, p' \in P$. For convenience, in this paper, the terms "customer" and "cluster" is used exchangeably. We also define an artificial customer corresponding to the depot. This customer is indexed as 0 and has demand $d_0 = 0$ and set of locations $N_0 = \{0\}$ with time window $[e_0, l_0] = [0, T]$. As defined in (Reyes, Savelsbergh, and Toriello 2017), for customer p appearing at k locations, e.g., $N_p = \{i_1, i_2, \dots, i_k\}$, the time windows associated with these locations considered in the VRPRDL must satisfy the following attributes:

$$\begin{aligned} & e_{i_1} = 0, \quad l_{i_k} = T, \\ & e_{i_j} = l_{i_{j-1}} + t_{i_{j-1} i_j} \quad \forall j \in [2, k] \end{aligned} \tag{1}$$

These conditions are highly relevant to the real-world case where a customer's itinerary starts and ends at a location (e.g., home), his/her vehicle moves from one place to an-

*Corresponding author

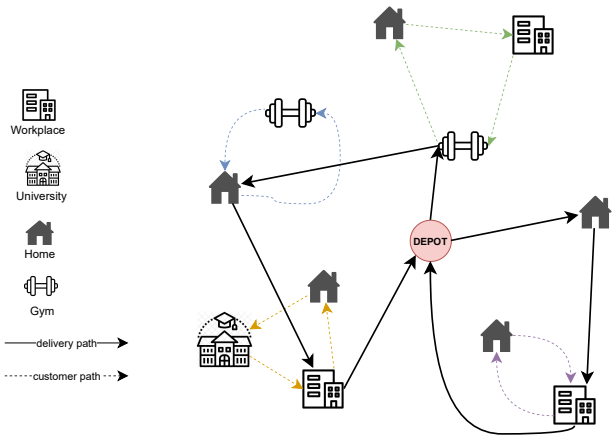

Figure 1: Example of a VRPRDL solution

other, takes the same travel time as the delivery vehicles and cannot be served during this period. It is worth mentioning that one can consider inequality $e_{i_j} \geq l_{i_{j-1}} + t_{i_{j-1}i_j}$ instead of equality (1) to model more general cases. This inequality can be handled by our proposed solution method without any change.

The goal of the VRPRDL is to find a set of m feasible vehicle routes with minimum total traveling cost serving each customer at only one location such that: (i) a vehicle starts and ends its route at the depot, (ii) a service at a location must start within its time window, (iii) the total load of each vehicle is not permitted to exceed its capacity Q, and (iv) the duration of each route never exceeds T. As originally defined in (Reyes, Savelsbergh, and Toriello 2017), the fleet size is not restricted, thus m is considered as a decision variable. Figure 1 illustrates a VRPRDL solution using two vehicles that depart from a depot and serve five customers. Each customer has a home location where he/she starts and ends the daily itinerary.

The VRPRDL and its variants have received significant attention from the literature since its first introduction in (Reyes, Savelsbergh, and Toriello 2017). The first exact method is a branch-and-price (B&P) algorithm proposed by (Ozbaygin et al. 2017), which can solve optimally instances of up to 60 customers and 237 locations within a few minutes. An improved exact method called branch-and-price-and-cut is introduced in (Tilk, Olkis, and Irnich 2021). It can find optimal solutions for most of existing instances except seven 120-customer instances. In terms of heuristic methods, a hybrid metaheuristic based on the principle of Greedy Randomized Adaptive Search Procedure (GRASP) and Large Neighborhood Search (LNS) is proposed in (Reyes, Savelsbergh, and Toriello 2017). (Dumez, Lehuédé, and Péton 2021) study the Vehicle Routing Problem with Delivery Options (VRPDO), which can be seen as an extended version of the VRPRDL in several aspects. First, deliveries in the VRPDO can take place at shared delivery locations (SDLs), like lockers or pick-up points, not only at home or in a car trunk as in the VRPRDL. Second, when a customer orders a parcel in the VRPDO, he/she chooses multiple delivery options, sorted by preference level, and a constraint related to service quality must be guaranteed. To solve the new problem, the authors design a LNS with specific ruin and recreate operators combined with numerous operators from the literature. In addition, a set partitioning problem is periodically used to reassemble routes with the aim of finding better solutions. When tested on VRPRDL instances, the algorithm produces all provably optimal solutions and improves some best-known results in running times of less than 1 minute.

The VRPRDL can also be considered as a special case of the Generalized Vehicle Routing Problem with Time Windows (GVRPTW) in which the time windows can be overlapped and thus do not satisfy conditions (1). The GVRPTW is introduced by (Moccia, Cordeau, and Laporte 2012) in the context of a school bus routing problem. Also in this work, an incremental tabu search is designed for solving the problem. More recently, (Dumez et al. 2021) proposed a LNS-based metaheuristic that can deal with many variants of the GVRPTW. This algorithm is an improved version of the one introduced in (Dumez, Lehuédé, and Péton 2021) and has four configurations obtained from combining a large neighborhood search with two exact components: the set partitioning and the Balas-Simonetti neighborhood (Balas and Simonetti 2001). An adaptive mechanism is also used for mixing these parts effectively. For convenience, we denote this algorithm as LNS*. In total, four configurations of LNS* find seven new best-known solutions on the VRPRDL instances. It is worth mentioning that, to solve the VRPRDL instances, LNS* requires to have the fleet size obtained from previous results in (Tilk, Olkis, and Irnich 2021) to limit the number of vehicles. Finally, (Yuan et al. 2021) introduce a Column Generation Based Heuristic (CGBH) to deal with the GVRPTW. The method combines several components including a construction heuristic, a route optimization procedure, local search operators, and the generation of negative reduced cost routes. The algorithm is compared with the B&P of (Ozbaygin et al. 2017) and provides better solutions on 19 VRPRDL instances. Similar to LNS*, the number of available vehicles is required and fixed manually so that CGBH can work on the VRPDRL instances.

In this paper, we propose an efficient hybrid genetic algorithm to solve the VRPRDL. The new features of our algorithm are as follows:

- We represent a VRPRDL solution as a sequence of all the customers also called a giant tour, instead of routes constituted by delivery locations. By this way, we can efficiently narrow the search space around giant tours including an appropriate number of visits. To evaluate the fitness of chromosomes, we design a *Split* procedure based on Dynamic Programming. Given a giant tour, our *Split* procedure can determine trip delimiters as well as select delivery locations for each customer to produce an optimal solution for the VRPRDL.
- We adapt classical local search operators for improving VRPRDL solutions. Several additional data structures are used to speed up the evaluation of local search moves.
- We integrate the *Split* and local search procedures into

a hybrid genetic algorithm to solve the VRPRDL. This algorithm also features a mutation component for population diversification and a set covering component for exploring improved solutions based on a set of potential routes generated during the search process. The computational results obtained on the benchmark instances from the literature show that our metaheuristic outperforms existing algorithms in terms of stability and solution quality.

Solution Method

In this section, we describe our new metaheuristic named HGA-SC for the VRPRDL. It is the combination of a well-known hybrid genetic algorithm (HGA) and a set-covering (SC) approach. Our method is inspired by the Hybrid Genetic Search with Adaptive Diversity Control (HGSADC) of (Vidal et al. 2014). Generally, HGSADC is a hybrid algorithm combining the exploration capability of genetic algorithms with efficient local search-based operators and diversity management mechanisms. The method divides its population into two sub-populations corresponding to feasible and infeasible solutions. In HGSADC, the population diversity is considered as an objective to be optimized along with solution quality through individual evaluations and selections.

Compared with the original HGSADC, the diversity management mechanism of HGA-SC is simpler when we manage the population containing only feasible solutions. By experiments, we observe that the diversity contribution has a negligible impact on the HGA-SC performance. Hence, it is not included in our evaluation of an individual. All the remaining components of HGSADC are redesigned specifically to solve the VRPRDL. In addition, our method contains two new complementary components: a mutation for diversification and a SC component for intensification.

The main steps of our HGA-SC are described in Algorithm 1 where $fit(S)$ presents the fitness (objective value) of solution S. The stopping conditions of the algorithm are controlled via the time budget T_{max} and the maximal number of iterations without improvement It_{NI}. Set Ω denotes the pool of vehicle routes obtained from solutions in population Pop during the search process, and the size of Ω is limited to the input parameter $maxPool$. Two parameters μ and λ are used to control the minimum and maximum sizes of the population. The variable $iter_{NI}$ keeps track of how many iterations have passed since the last new best solution was found and $iter$ denotes the current number of iterations. The algorithm starts by initializing route pool Ω, population Pop, variables $iter$ and $iter_{NI}$, and recording the best solution S^*. The main part of our algorithm is the iterative process (Lines 4-26). In each iteration, genetic operators are used to choose two parents from the population, combine them to produce an offspring C, and mutate it randomly (Lines 5-7). Offspring C is next educated by a local search procedure and then added to the population (Lines 8-9). The routes of C are also extracted and inserted into the route pool Ω at Line 10. The value of $iter_{NI}$ changes according to whether the new best solution is found (Lines 12, 14 and 22). A survivor selection phase is triggered if the population size is greater than a given value (Lines 15-16). Finally, after every It_{SC} iterations or when the size of route pool exceeds $maxPool$, the SC phase is executed and the pool Ω will be cleared (Lines 17-25). The algorithm stops after It_{NI} iterations without improvement or when the time budget T_{max} is reached. We now discuss the main operators of the algorithm in detail.

Algorithm 1: HGA-SC

Input : μ, λ, p_{mut}, It_{NI}, It_{SC}, T_{max}, $maxPool$
Output: Best found solution S^*

1 Route pool $\Omega \leftarrow \emptyset$
2 Initialize population Pop ;
3 $iter \leftarrow 0$, $iter_{NI} \leftarrow 0$, $S^* \leftarrow \arg\min_{x \in Pop} fit(x)$
4 **while** $iter_{NI} < It_{NI}$ and $running\ time < T_{max}$ **do**
5 Select parent solutions P_1 and P_2 from Pop
6 Generate offspring C from P_1 and P_2
7 Mutate offspring C with probability p_{mut}
8 Educate offspring C
9 Insert C into population
10 Add routes of C to Ω
11 **if** $fit(S^*) < \min_{x \in Pop} fit(x)$ **then**
12 $iter_{NI} \leftarrow 0$, $S^* \leftarrow \arg\min_{x \in Pop} fit(x)$
13 **else**
14 $iter_{NI} \leftarrow iter_{NI} + 1$
15 **if** $|Pop| > \mu + \lambda$ **then**
16 Select survivors
17 **if** $iter$ % $It_{SC} = 0$ or $|\Omega| \geq maxPool$ **then**
18 Obtain C_{SC} from solving the SC problem with route pool Ω
19 Remove duplicate requests in C_{SC}
20 Educate C_{SC}
21 **if** $fit(C_{SC}) < fit(S^*)$ **then**
22 $iter_{NI} \leftarrow 0$
23 Add C_{SP} to Pop
24 **if** $|\Omega| \geq maxPool$ **then**
25 $\Omega \leftarrow \emptyset$
26 $iter \leftarrow iter + 1$

Solution Representation and Evaluation

In HGA-SC, each VRPRDL solution is implicitly represented by a permutation of customers (called *giant tour*), instead of a set of routes in which each route is explicitly represented by a permutation of locations (also called *complete VRPRDL solution*). Each giant tour is obtained by encoding the sequence of the clusters visited in a tour excluding the depot clusters N_0 (*trip delimiters*). This way of representation, first introduced in (Prins 2004) for VRP solutions, is suitable for applying genetic operators without indicating individual routes. A *Split* procedure based on dynamic programming to solve a shortest path problem is then required to find the complete solution for each giant tour. It optimally re-inserts trip delimiters into the customer sequence and selects the location to visit for each customer. Because the number of customers is in general much less than the num-

Algorithm 2: Split procedure

Input : A permutation of customers: $\mathcal{T} = (cus_1, cus_2, \dots, cus_{|P|})$

Output: The minimal cost to serve all customers $cost_{|P|}$

1 Initialize $cost_i \leftarrow \infty \; \forall i \in [1, |P|]$
2 $cost_0 \leftarrow 0$
3 **for** $i = 1 \rightarrow |P|$ **do**
4 $load \leftarrow 0$
5 **for** $j = i \rightarrow |P|$ **do**
6 $load \leftarrow load + d_{cus_j}$
7 **if** $load > Q$ **then**
8 *break*
9 $cost_j \leftarrow \min\big(cost_j, cost_{i-1} +$ CalCost$(cus_i, cus_{i+1}, \dots, cus_j)\big)$

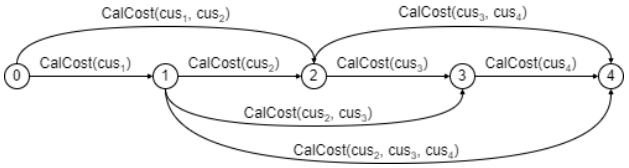

Figure 2: Auxiliary weighted graph G' with $|P| = 4, Q = 10, d_{cus_1} = d_{cus_2} = d_{cus_3} = 4$ and $d_{cus_4} = 2$

ber of locations, using giant tours with customers instead of complete representation with trip delimiters and locations to represent VRPRDL solutions allows to significantly narrow the search space.

The *Split* procedure is presented in Algorithm 2. Given a giant tour $\mathcal{T} = (cus_1, cus_2, \dots, cus_{|P|})$, it constructs an auxiliary directed acyclic graph G' with $|P| + 1$ nodes indexed from 0 to $|P|$. Each subsequence of customers $(cus_i, \dots, cus_j)$ that can be considered as one route (the total demand is not greater than Q) is modeled by an arc $(i-1, j)$ of graph G'. The cost of this arc is equal to the cost of its corresponding route and computed by using the function CalCost() which is described in detail below. Figure 2 depicts an example of graph G' with four customers and vehicle capacity $Q = 10$. Note that, some arcs are not included due to the capacity constraint. For instance, the total demand of all customers is 14, exceeding the capacity $Q = 10$. Therefore, arc (0, 4) is not present in graph G'. Finally, the $Split$ algorithm solves the shortest path problem from node 0 to node $|P|$ using Bellman's algorithm on the generated graph G'. The output of this procedure is the minimal cost of VRPRDL solution corresponding to giant tour $\mathcal{T}$, equal to $cost_{|P|}$.

We now describe function CalCost() to compute the cost of arc $(i-1, j)$ in G' corresponding to customer sequence $(cus_i, cus_{i+1}, \dots, cus_j)$ in giant tour $\mathcal{T}$ satisfying the condition $\sum_{k=i}^{j} d_{cus_k} \leq Q$. A trivial approach is to verify all feasible options by iterating over every location of each customer to find the best solution. However, the complexity of this approach is $\mathcal{O}(\prod_{k=i}^{j} |N_k|)$, which is exponential. To speed up the process, we reuse the Dynamic Programming (DP) algorithm proposed in (Reyes, Savelsbergh, and Toriello 2017).

The procedure works as follows. First, a state of the DP is represented by a 3-tuple (p, τ, v), where $p \in \{start, cus_i, \dots, cus_j, end\}$ implies the current visited customer ($start$ and end denote the artificial customers related to the depot), $v \in N_p$ is the location where customer p is served, and $\tau \in [e_v, l_v]$ is the time when the vehicle leaves location v. We define $F(p, \tau, v)$ is the mininal cost of a partial route $(start, \dots, p)$ that visits location v of customer p at time τ. The DP performs the following recursive calls:

$$F(start, 0, 0) = 0 \tag{2}$$

$$F(cus_i, \tau, v) = \min_{u \in N_{start}} \{F(start, \tau', u) + c_{uv}\} \quad \text{if } e_u \leq \tau' \leq \min(l_u, \tau - t_{uv}) \tag{3}$$

$$F(cus_k, \tau, v) = \min_{u \in N_{cus_{k-1}}} \Big\{F(cus_{k-1}, \tau', u) + c_{uv}\Big\} \quad \text{if } e_u \leq \tau' \leq \min(l_u, \tau - t_{uv}) \quad \forall i < k \leq j \tag{4}$$

$$F(end, \tau, 0) = \min_{u \in N_{cus_j}} \Big\{F(cus_j, \tau', u) + c_{u0}\Big\} \quad \text{if } e_u \leq \tau' \leq \min(l_u, \tau - t_{uv}) \tag{5}$$

Finally, the result of function CalCost() with input sequence $(cus_i, ..., cus_j)$ is computed by

$$\text{CalCost}(cus_i, \dots, cus_j) = \min_{\tau \in [0,T]} F(end, \tau, 0) \tag{6}$$

As shown in (Reyes, Savelsbergh, and Toriello 2017), thanks to the special structure of time windows in the VRPRDL 1, the complexity of the DP to compute the cost of sequence $(cus_i, \dots, cus_j)$ can be estimated as $\mathcal{O}\big((j - i + 1)^2 \max_{i \leq k \leq j} |N_k|^2\big)$. Let N_{max} be the largest cluster and B be the maximum number of customers in a route. The overall complexity of our *Split* algorithm is computed as:

$$\begin{aligned}\mathcal{O}(|P| \sum_{k=1}^{B} (k^2 |N_{max}|^2)) &= \mathcal{O}(|P||N_{max}|^2 \sum_{k=1}^{B} k^2) \\ &= \mathcal{O}(|P||N_{max}|^2 B^3) = \mathcal{O}(|P|B^3|N_{max}|^2)\end{aligned} \tag{7}$$

Furthermore, we observe that the customer sequence in iteration j (Line 9) is created by adding only one more customer cus_j into the customer sequence in iteration $j-1$. Hence, at each iteration of the loop in Line 5, we implement the function CalCost() that allows it to reuse DP's states from the previous iteration. This acceleration allows the *Split* algorithm to be done in $\mathcal{O}(|P|B^2|N_{max}|^2)$.

Genetic Operators

During the generation of a new individual, three genetic operators selection, crossover, and mutation are performed sequentially before the education phase. In the selection phase, we select a pair of parents based on the *Rank-based Roulette Wheel Selection* (RRWS) (Holland 1992), which provides a consistent scale across the population and is unaffected by super-individuals or fitness value distribution to avoid a risk of premature convergence to a local optimum.

Algorithm 3: LS procedure

Input : A complete VRPRDL solution S, a list of neighborhood operators $\mathcal{N}$
Output: An improved solution S_{imp}

1 $S_{imp} \leftarrow S, fit(S_{imp}) = fit(S)$
2 **for** $p \in P$ **do**
3 **for** $q \in \pi(p)$ **do**
4 **for** $k = 1 \rightarrow |\mathcal{N}|$ **do**
5 $S' \leftarrow move(S_{imp}, (p, q), \mathcal{N}_k)$
6 **if** $fit(S') < fit(S_{imp})$ **then**
7 $S_{imp} \leftarrow S'$
8 **break**

A child C is then created by performing the *Ordered Crossover* (OX) (Oliver, Smith, and Holland 1987) on two parent solutions P_1 and P_2 obtained from the selection phase. First, it takes a randomly selected sub-sequence of parent P_1, copies it into child C, and then after the last node of this sub-sequence, appends circularly clusters in parent P_2 from the same position, skipping any inserted clusters. This crossover operator is widely used in evolutionary algorithms proposed to solve VRP variants (Prins 2004; Vidal et al. 2014). Finally, with a given probability p_{mut}, we perform a mutation component to increase the population diversity. This component randomly swaps n_{mut} pairs of customers in the giant tour.

Local Search for Education

To intensify the search in our HGA-SC, after an offspring is generated by using the genetic operators, its complete VRPRDL solution obtained from *Split* algorithm is improved through an $education$ phase. In this process, a Local Search (LS) procedure described in Algorithm 3 is repeatedly performed until no better solution is found. It is worth mentioning that, only in (Yuan et al. 2021), LS operators are used in the CG-based algorithm to deal with the VRPRDL. In HGA-SC, we use more complex LS operators as well as special data structures to speed up the move evaluation, which are not proposed in (Yuan et al. 2021).

Given an incumbent solution S and a list of neighborhood operators $\mathcal{N}$. A neighborhood operator (or simply neighbourhood) $\mathcal{N}_k \in \mathcal{N}$ generates a range of neighboring solutions by applying a unique type of move. First, the LS procedure loops through all the customers (Line 2) in random order. Then, for each customer p, a set $\pi(p)$ of neighbor customers is traversed in an order sorted according to a specific metric at Line 3. For each pair of customers (p, q) and neighborhood operator $\mathcal{N}_k$, the LS operator creates solution S' by applying a move in $\mathcal{N}_k$ on solution S_{imp} (Line 5). After this modification, S' must contain an arc that links a location of cluster p with a location of cluster q. When an improvement is detected, we stop checking the remaining operators in $\mathcal{N}$ (Lines 6 to 8). This scheme is quite similar to $composite\ neighborhood$ technique which is widely used for exploring LS neighborhoods to solve VRPs (Schröder et al. 2020).

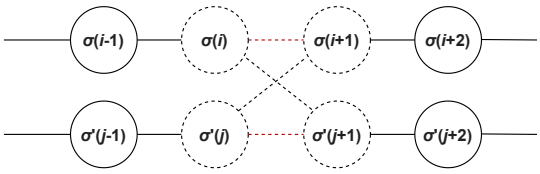

Figure 3: An illustration of 2-opt* move

.

In the following, we describe in details the construction of $\pi(p)$, the selection of operators in $\mathcal{N}$, and the evaluation of LS moves. For convenience, we represent route r in a VRPRDL solution as a sequence of visits $\sigma^r = [\sigma^r(1), \sigma^r(2), \ldots, \sigma^r(|\sigma^r|)]$, where $\sigma^r(i)$ is the ith visited customer. We also denote $loc_{\sigma^r(i)} \in N_{\sigma^r(i)}$ is the selected delivery location of customer $\sigma^r(i)$.

Neighborhood Structure. Designing an effective list of neighborhoods $\mathcal{N}$ is critical to the performance of LS. The possibility of finding a better solution increases with each additional operator. However, this requires more running time for exploring each new neighborhood. The balance between these two factors is very important for making an efficient LS procedure. In our HGA-SC, we use the following three neighborhoods for constructing $\mathcal{N}$:

- $\mathcal{N}_1$ (Swap and Relocate): This move swaps two disjoint visit sequences containing from 0 to 2 consecutive customers. During the process, it is permitted to reverse one or both sequences. This neighborhood can be used on a single route (*intra-route*) as well as two distinct routes (*inter-route*).
- $\mathcal{N}_2$ (2-opt*): This move cuts two different routes into the front and back parts, then reconnects the front part of the first route with the back part of the second route, and vice versa. The neighborhood is depicted in Figure 3).
- $\mathcal{N}_3$ (2-opt): In this move, a sequence of customers between two positions is inverted.

In (Yuan et al. 2021), the authors implement Swap and Relocation moves which exchange the position of only two customers and relocate a single customer to other positions, respectively. Their LS operators are also not performed in the intra-route fashion. Thus, they can be seen as a simple version of our Swap and Relocate. In addition, they do not design 2-opt* and 2-opt moves for the problem.

Similar to (Vidal et al. 2013), each customer $\sigma^r(i)$ has an ordered set $\pi(\sigma^r(i))$ that stores the neighbor customers $\sigma^{r'}(j)$ sorted by an increasing measure $\mathcal{M}(\sigma^r(i), \sigma^{r'}(j))$ which is in general the travel cost between these two customers. To adapt for VRPRDL where a customer has a set of delivery locations, we propose the following mesure to estimate the cost between two customers:

$$\mathcal{M}(\sigma^r(i), \sigma^{r'}(j)) = \min_{u=loc_{\sigma^r(i)}} \{c_{uv} : v \in N_{\sigma^{r'}(j)}\} \quad (8)$$

The above measure is dependent on the delivery location $loc_{\sigma^r(i)}$ of customer $\sigma^r(i)$ in the current solution. This could avoid changing the visit sequence before customer $\sigma^r(i)$ in route r. Therefore, using it in the formula for calculating $\mathcal{M}$

may be better than using the whole locations of its cluster. The LS procedure privileges exploring the neighboring solutions that contain pairs of customers with smaller values $\mathcal{M}$. More precisely, the search is biased towards moves that are promising for reducing the total cost of the current solution. This technique is often referred to as *granular search* (Toth and Vigo 2003), and applied successfully in many previous studies (e.g., (Vidal et al. 2014; Schneider and Löffler 2019)). The number of neighbor customers $|\pi|$ is in general limited to a given value to reduce the computation time. In our algorithm, we use the full neighborhood size ($|\pi| = |P|$) to achieve the best result.

Move Evaluations. For each local search move, we need to check the feasibility and compute the objective value of changed solution before making the next search decision (i.e., update the solution or switch to other neighborhoods). In general, these operations consume the largest part of running time of the algorithm. Thus, they need to be carefully designed to speed up the LS operators. In HGA-SC, we employ additional data structures to efficiently check capacity and time-window constraints, and fast compute the modified cost of solutions after applying the LS moves. It is important to note that during the LS process, a location can be replaced by other locations in the same cluster. Therefore, the delivery locations before and after the move may be different.

We adapt the technique introduced in (Vidal et al. 2014) where a local search move can be considered as a separation of routes into subsequences, which are then concatenated to construct new routes. Let two routes r and r' be represented by two visit sequences $\sigma^r = [\sigma^r(1), \sigma^r(2), .., \sigma^r(|\sigma^r|)]$ and $\sigma^{r'} = [\sigma^{r'}(1), \sigma^{r'}(2), .., \sigma^{r'}(|\sigma^{r'}|)]$, respectively. A swap move, which exchanges the position of two visits $\sigma^r(i)$ and $\sigma^{r'}(j)$, can yield following recombined sequences:

$$\varphi = [\sigma^r(1), \dots, \sigma^r(i-1)] \oplus [\sigma^{r'}(j)] \oplus [\sigma^r(i+1), \dots, \sigma_r(|\sigma_r|)] \quad (9)$$

$$\varphi' = [\sigma^{r'}(1), \dots, \sigma^{r'}(j-1)] \oplus [\sigma^r(i)] \oplus [\sigma^{r'}(j+1), \dots, \sigma^{r'}(|\sigma^{r'}|)] \quad (10)$$

These generated sequences denote two new routes after applying the swap move on r and r'. Both of them contain a concatenation of three subsequences. Here, $\oplus$ indicates the concatenation operator.

For each such subsequence $\sigma = [\sigma(1), \sigma(2), \dots, \sigma(|\sigma|)]$, we compute the total travel time $TT(\sigma)$, the earliest time $ET(\sigma)$ that the vehicle leaves the last customer $\sigma(|\sigma|)$'s location in the subsequence, the latest time $LT(\sigma)$ for starting the service at the first customer $\sigma(1)$, the total load $TL(\sigma)$, and the summation of travel costs $TC(\sigma)$. We can directly calculate these data for a sequence σ^0 involving a single customer p, as $TT(\sigma^0) = 0, ET(\sigma^0) = e_{loc_p}, LT(\sigma^0) = l_{loc_p}, TL(\sigma^0) = d_{loc_p}$, and $TC(\sigma^0) = 0$. By supposing $u = loc_{\sigma(|\sigma|)}$ and $v = loc_{\sigma'(1)}$, the computations of the same data for a concatenation of two sequences σ and σ' are shown in Equations 11 to 15.

$$TT(\sigma \oplus \sigma') = TT(\sigma) + t_{uv} + TT(\sigma') \quad (11)$$

$$ET(\sigma \oplus \sigma') = \max\{ET(\sigma) + t_{uv} + TT(\sigma'), ET(\sigma')\} \quad (12)$$

$$LT(\sigma \oplus \sigma') = \min\{LT(\sigma), LT(\sigma') - t_{uv} - TT(\sigma)\} \quad (13)$$

$$TL(\sigma \oplus \sigma') = TL(\sigma) + TL(\sigma') \quad (14)$$

$$TC(\sigma \oplus \sigma') = TC(\sigma) + c_{uv} + TC(\sigma') \quad (15)$$

Our proposed move evaluation procedure employs the above equations to first prepare data on relevant consecutive visit subsequences (and their reversal) in a route through a preprocessing phase. These data can be used directly to compute the cost improvement as well as validate the capacity and time-window constraints in $\mathcal{O}(1)$ when we fix the delivery locations of all customers. However, if we allow to modify the delivery locations of clusters during the process of the LS moves, our algorithm can explore wider search space and thus work better.

We now describe in detail how to efficiently evaluate a LS move for the VRPRDL allowing the modification of delivery locations in each cluster. First, for each inter-route neighborhood operator, we restrict some specific customers to be able to change their locations during the search as follows:

- $\mathcal{N}_1$ (Swap and Relocate): Only customers in the moving sequences or being adjacent to these sequences are permitted to change their delivery locations. We slightly modify the DP used in the function CALCOST() (Equations 2-5) by adjusting its initial state and result states. For simplicity, let us make an example with the swap move on two routes r and r' yielding two new associated sequences φ and φ' as described above (Equations 9-10). We perform the DP algorithm on two visit subsequences $[\sigma^r(i-1), \sigma^{r'}(j), \sigma^r(i+1)]$ and $[\sigma^{r'}(j-1), \sigma^r(i), \sigma^{r'}(j+1)]$ with their changed initial state (Equation 2) respectively as follows:

  $$F\big(start, ET(\varphi_1), loc_{start}\big) = C(\varphi_1) \;\; (start = \sigma^r(i-2)),$$

  $$F\big(start, ET(\varphi'_1), loc_{start}\big) = C(\varphi'_1) \;\; (start = \sigma^{r'}(j-2))$$

  in which $\varphi_1 = \big[\sigma^r(1), \dots, \sigma^r(i-2)\big]$ and $\varphi'_1 = \big[\sigma^{r'}(1), \dots, \sigma^{r'}(j-2)\big]$. Similarly, the result states (the left side of Equation 5) become $(\sigma^r(i+2), \tau, loc_{\sigma^r(i+2)})$ $\big($for sub-sequence $[\sigma^r(i-1), \sigma^{r'}(j), \sigma^r(i+1)]\big)$, and $(\sigma^{r'}(j+2), \tau, loc_{\sigma^{r'}(j+2)})$ $\big($for sub-sequence $[\sigma^{r'}(j-1), \sigma^r(i), \sigma^{r'}(j+1)]\big)$. The travel costs of two new aforementioned sequences φ and φ' are:

  $$TC(\varphi) = \min_{\tau \in [e_u, LT(\varphi_2)]} \big\{F(\sigma^r(i+2), \tau, u) + TC(\varphi_2)\big\}$$

  $$TC(\varphi') = \min_{\tau' \in [e_v, LT(\varphi'_2)]} \big\{F(\sigma^{r'}(j+2), \tau', v) + TC(\varphi'_2)\big\}$$

  where $u = loc_{\sigma^r(i+2)}, v = loc_{\sigma^{r'}(j+2)}, \varphi_2 = \big[\sigma^r(i+2), \dots, \sigma^r(|\sigma^r|)\big]$ and $\varphi'_2 = \big[\sigma^{r'}(j+2), \dots, \sigma^{r'}(|\sigma^{r'}|)\big]$. As shown in the previous section, in the neighborhood $\mathcal{N}_1$, the size of two swap sequences is in the range of 0-2 consecutive visits. Therefore, the maximum length of a subsequence that allows its elements to change their locations is bounded to 4, and the complexity of each evaluation move in this case is $\mathcal{O}(4^2|\mathcal{N}_{max}|^2) = \mathcal{O}(16|\mathcal{N}_{max}|^2)$.
- $\mathcal{N}_2$ (2-opt*): In this neighborhood operator, we need to split two different routes into front and back parts. The four customers (dash circles in Figure 3) related to the splitting positions are allowed to modify their delivery

locations. In other words, the LS procedure traverses through all possible locations of two customers in each reconnected arc (black dash line in Figure 3). The evaluation of this move can be done in $\mathcal{O}(|\mathcal{N}_{max}|^2)$.

For the intra-route LS moves on a route r, there is no restriction on any customers. This means whenever evaluating a move performed on route r, we call function CALCOST($\overline{\sigma}^r$) for the new visit sequence of r obtained after applying the move. We note that computing function CALCOST() can be quite expensive. However, we observe that, for the VRPRDL instances of the literature, the average length of routes in solutions is not large. Thus, our implementation for the intra-route moves is still fast and efficient.

Population Management

The population size of our HGA-SC varies from μ to μ + λ. In the initialization phase, μ individuals are obtained through randomly creating giant tours. The survivor selection phase is triggered when the size of population Pop exceeds the value μ + λ to remove weak and *clone* individuals until retaining μ individuals. In (Vidal et al. 2014), once the best found solution does not change in a specific number of iterations, the population undergoes a $Diversification$ phase that keeps a portion of best individuals and inserts randomly created individuals until the size of population reaches μ. We do not use it in our algorithm because our preliminary experiments show that this restart procedure is quite time-consuming when solving the VRPRDL, leading to tedious performance of the algorithm.

Set Covering Approach

In (Dumez, Lehuédé, and Péton 2021; Dumez et al. 2021), a Set Covering (SC) operator is added to the LNS-based metaheuristics to efficiently intensify the search. We also reuse this idea for our HGA-SC. As far as we know, this paper is the first attempt to use the SC component in genetic algorithms to solve VRP variants. Other related studies only consider the SC and GA as two separate parts (Alvarenga, Mateus, and De Tomi 2007), or exploit the SC for computing the fitness function (Walteros, Medaglia, and Riaño 2015).

As described above, Ω is the route pool that is generated by extracting feasible routes from each offspring during the search. We can reassemble the routes in Ω to create a better solution via solving a SC formulation, which uses several variables and parameters as follows:

- v_r: parameters representing the cost of route $r \in \Omega$.
- β_r: binary variables equal to 1 if route $r \in \Omega$ is selected in the new solution, and 0 otherwise.
- α_r^p: binary parameters equal to 1 if route $r \in \Omega$ visits customer $p \in P$, and 0 otherwise.

The SC problem can be mathematically formulated as:

$$\min \quad \sum_{r \in \Omega} v_r \beta_r \tag{16}$$

$$\text{s.t.} \quad \sum_{r \in \Omega} \alpha_r^p \beta_r \geq 1 \qquad \forall p \in P \tag{17}$$

$$\beta_r \in \{0, 1\} \qquad \forall r \in \Omega \tag{18}$$

The objective (16) minimizes the total travel cost. Constraints (17) ensure that each customer must be visited at least once. They can make a customer be visited many times, thus leading to infeasible VRPRDL solutions. To deal with this issue, we employ a simple greedy procedure to remove redundant customer nodes. We refer the readers to (Dumez, Lehuédé, and Péton 2021; Dumez et al. 2021) for more information. We also note that this approach only works if both travel times and costs satisfy the triangle inequality, which is always true for the VRPRDL instances.

During the process of managing the route pool, a hashing data structure is implemented for detecting duplicate routes. A route is considered as duplicate if it contains the same node set as an existing one. Between two routes containing the same node set, we retain only the one of lower cost. Additionally, when the size of Ω exceeds its limit, we remove all of its elements. This partially contributes to keeping the running time of the SC component reasonable and diversifying the search.

Computational Experiments

The HGA-SC is implemented in C++ and run on an AMD Ryzen 7 3700X @ 3.60GHz and 16.0 GB RAM under Window 10. We use CPLEX 12.10 for solving the SC problem. We conduct our experiments on four sets of VRPRDL and VRPHRDL instances proposed in (Ozbaygin et al. 2017). In these instances, the number of customers varies from 15 to 120 and each customer appears at most 6 locations. All instances and detailed results are available online[1].

Comparison Results

The best solutions for the available VRPRDL instances are found by LNS* in (Dumez et al. 2021) and CGBH in (Yuan et al. 2021). We now compare the results obtained by HGA-SC with those provided by CGBH and the best configuration among four versions of LNS*. To solve the VRPRDL instances, both LNS* and CGBH need a fixed integer as input representing the maximal number of vehicles that can be used. This value for each instance is set manually (for CGBH) or to the number of routes in the corresponding solution obtained from the branch-and-price-and-cut of (Tilk, Olkis, and Irnich 2021) (for LNS*). Two algorithms LNS* and CGGH are tested on an Intel Xeon X5650 at 2.57 GHz, and an Intel(R) Core(TM) i5–6200U CPU at 2.30 GHz, respectively. According to Passmark Software[2], these machines have single-thread rating performance approximately twice as weak as our processor.

Preliminary experiments suggest the following termination conditions of our algorithm can lead to a good trade-off between the solution quality and speed: It_{NI} is set to 5000 and T_{max} is set to 1800 seconds for instances with 120 customers and 360 seconds for smaller instances. We then calibrate the remaining parameters of HGA-SC by using the automatic configuration package IRACE (López-Ibáñez et al. 2016). Table 1 shows the tested and selected parameter values. For each instance, similarly to LNS* in (Dumez et al.

[1] http://orlab.com.vn/home/download

[2] https://www.cpubenchmark.net

Parameters	Set or range of values	Selected values
μ	{20, 40}	40
λ	{20, 40, 80}	80
p_{mut}	[0.5, 1.0]	0.8
n_{mut}	[2, 10]	10
It_{SC}	{1000, 1500, 2000}	2000
$maxPool$	{30000}	30000

Table 1: Parameter tuning results

Set	#Ins	n	Method	$\overline{gap}$	$\overline{gap^*}$	max	max^*	$time$
$\mathcal{B}_1$	30	15-60	CGBH	–	0.00	–	0.00	1.91
			LNS*	0.00	0.00	0.01	0.00	60.00
			HGA-SC	**0.00**	**0.00**	**0.00**	**0.00**	142.00
$\mathcal{B}_1$	10	120	CGBH	–	0.06	–	0.22	62.12
			LNS*	0.05	0.00	0.25	0.00	360.00
			HGA-SC	**0.01**	**0.00**	**0.03**	**0.00**	1696.56
$\mathcal{B}_2$	30	15-60	CGBH	–	0.00	–	0.05	6.28
			LNS*	0.02	0.00	0.55	0.00	60.00
			HGA-SC	**0.00**	**0.00**	**0.00**	**0.00**	158.08
$\mathcal{B}_2$	10	120	CGBH	–	0.37	–	1.39	241.00
			LNS*	0.48	0.20	2.43	1.47	360.00
			HGA-SC	**0.18**	**-0.04**	**0.98**	**0.00**	1799.55
$\mathcal{B}_3$	20	40	CGBH	–	0.01	–	0.15	3.27
			LNS*	0.01	0.00	0.12	0.00	60.00
			HGA-SC	**0.00**	**0.00**	**0.00**	**0.00**	168.08
$\mathcal{B}_4$	20	40	CGBH	–	–	–	–	–
			LNS*	0.14	0.01	1.21	0.21	60.00
			HGA-SC	**0.03**	**0.00**	**0.00**	**0.00**	170.29

Table 2: Summary of comparison results

2021), we run our HGA-SC 5 times to investigate its stability. Note that CGBH in (Yuan et al. 2021) is run only once on each instance.

Table 2 summarizes the obtained comparison results for each instance set. For two sets $\mathcal{B}_1$ and $\mathcal{B}_2$, we separate the results into two parts: one for 120-customer instances and the other for smaller instances. Because the 120-customer instances significantly require more running time than other instances and are thus much harder to solve, the separation allows to observe more clearly the behavior of the algorithms on the large instances. The headings of Table 2 indicate the instance set (Column "Set"), the number of instances in the set (Column "#Ins"), and the number of customers in the instances (Column "n"). For each instance, we compute the deviation in percentage of the average objective value from the Best Known Solution (BKS) in the literature denoted as gap, and the deviation in percentage of the best solution found over 5 runs from the BKS denoted as gap^*. Column "$\overline{gap}$" shows the values averaged over all instances of gap, while the corresponding values for gap^* are reported in Column "$\overline{gap^*}$". Note that, since CGBH runs only once for each instance, the values "$\overline{gap}$" of CGBH are not available and marked as "–". We also do not report the results of set $\mathcal{B}_4$ for CGBH due to their unavailability. The maximum values of gap and gap^* over all instances are shown in Columns max and max^*, respectively. The last column "$time$" reports the average computation time in seconds to find the results for each method.

Table 2 clearly shows the dominance of our algorithm regarding solution quality when it provides better results in all criteria. The $\overline{gap^*}$ values of HGA-SC are never greater than 0.00, showing that our algorithm can produce all BKSs found in the literature. More remarkably, we improve two new best results for instances 34 and 39 of set $\mathcal{B}_2$. The relatively high values of max^* in case of LNS* and CGBH (1.39 and 1.47, respectively) imply that these algorithms have difficulty reaching BKSs when dealing with the largest instances of set $\mathcal{B}_2$. With regard to stability, we only compare HGA-SC and LNS* due to the availability of the results on multiple runs. As can be seen, the $\overline{gap}$ values of HGA-SC are always less than or equal to those of LNS*. The maximal deviation values max are always less than 1% while these values for LNS* can increase up to 2.43%. These demonstrate that HGA-SC is significantly more stable than LNS*.

In terms of running time, our algorithm is clearly slower than the others. However, we also recall that, to efficiently solve the VRPRDL instances, LNS* and CGBH need a predefined parameter input representing the maximal number of vehicles that can be used. The parameter for each instance is set manually (for CGGH) or taken from existing algorithms (for LNS*). Because the number of vehicles required in solutions of the VRPRDL instances is relatively large (approximately 20 in the instances with 120 customers), this value possibly allows both algorithms to narrow the search space, and thus to reduce the runtime significantly. On the contrary, our algorithm automatically determines an appropriate number of vehicles during the search. In many cases, we observe that, because of time window constraints, the number of vehicles in initial solutions is much larger than that in best-known solutions, partly leading to more challenges for the HGA-SC.

Sensitivity Analysis of Algorithm Components

In this section, we focus on analyzing the impact of our two new proposed components mutation and set covering, which are rarely applied in genetic algorithms to solve VRP variants. To achieve the goal, we evaluate three versions of the HGA-SC obtained from performing the following modifications:

- HGA-SC$_{base}$: removing both mutation and SC components.
- HGA-SC$_{noMut}$: removing only the mutation.
- HGA-SC$_{noSC}$: removing only the SC component.

For each version, we use the same setting of HGA-SC as described in the previous section. Table 3 shows the comparison between these configurations and HGA-SC on all sets of instances. In this table, Columns "Set" and "#Ins" have the same meaning as in Table 2. The selected comparison criteria are the summations of gap and gap^* values (see the previous section) of each instance set in Columns "$\sum gap$" and "$\sum gap^*$", respectively. In Column "#BKS", we present another criterion, which is the number of instances that an algorithm version can provide solutions better than or equal to BKSs (without ones found in this paper).

As can be seen in Table 3, the results demonstrate the positive impact of the mutation and the SC components on the performance of HGA-SC. When both components are omitted, the algorithm performs the worst in all three criteria. Using the mutation or SC separately makes the algorithms perform better. And the combination of both components

Set	#Ins	Configuration	$\sum gap$	$\sum gap^*$	#BKS
$\mathcal{B}_1$	40	HGA-SC$_{base}$	1.34	1.18	37
		HGA-SC$_{noMut}$	1.72	1.11	36
		HGA-SC$_{noSC}$	1.51	1.41	36
		HGA-SC	**0.07**	**0.00**	**40**
$\mathcal{B}_2$	40	HGA-SC$_{base}$	8.39	5.94	35
		HGA-SC$_{noMut}$	6.28	5.86	36
		HGA-SC$_{noSC}$	5.73	3.48	36
		HGA-SC	**1.84**	**-0.44**	**40**
$\mathcal{B}_3$	20	HGA-SC$_{base}$	0.00	0.00	20
		HGA-SC$_{noMut}$	0.00	0.00	20
		HGA-SC$_{noSC}$	0.00	0.00	20
		HGA-SC	**0.00**	**0.00**	**20**
$\mathcal{B}_4$	20	HGA-SC$_{base}$	1.63	0.21	19
		HGA-SC$_{noMut}$	1.63	0.21	19
		HGA-SC$_{noSC}$	**0.18**	0.00	20
		HGA-SC	0.53	**0.00**	**20**

Table 3: Sensitivity analysis on the impact of mutation and SC components

creates the best version of the algorithm. We believe this is due to the better balance between diversification and intensification of our metaheuristic when these components are added.

Scalability Analysis of HGS-SC

In this section, we analyze the impact of the number of customers, the number of locations, and the route length on the performance of HGS-SC. In (Yuan et al. 2021), the authors modified two instance sets $\mathcal{B}_1$ and $\mathcal{B}_2$ to create two new sets called $\mathcal{B}_1$-var and $\mathcal{B}_2$-var, respectively by reducing the customer demand and the travel times, then widening the time windows. We make an additional experiment by running our HGA-SC on 4 instance sets $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_1$-var and $\mathcal{B}_2$-var with only one stopping condition that limits the total number of iterations to 7000. For each instance, our HGA-SC is run once for a fair comparison with CGBH and five times for a convenient comparison with other metaheuristics in the future. Table 4 summarizes our results.

For each instance, we compute the deviations in percentage of the best solution found over 1 run (gap_1) and 5 runs (gap_5) from the BKS. The average values of gap_1 and gap_5 for each instance group are presented in Columns "$\overline{gap}_1$" and "$\overline{gap}_5$", respectively. The last three columns report the calculated values in the 1-run configuration. Column "$nbRou$" shows the average number of used vehicles in best found solutions. Column "$time$" represents the average running time in seconds of each method. To observe the performance of HGA-SC and CGBH before and after increasing the route length, the average computation time of these methods on sets $\mathcal{B}_1$ and $\mathcal{B}_2$ is also reported in Column "$time_{ori}$".

Results in Column "$\overline{gap}_1$" show that HGA-SC still produces better solutions than CGBH. In particular, on two sets $\mathcal{B}_1$-var and $\mathcal{B}_2$-var with the 5-run configuration, we improve 47 solutions of CGBH and provide worse solutions in only 6 instances. The number of used vehicles in our final solutions is even never larger than that of CGBH except instance 35 of

Set	n	Method	$\overline{gap}_1$	$\overline{gap}_5$	$nbRou$	$time$	$time_{ori}$
$\mathcal{B}_1$-var	15-60	HGA-SC	**-0.18**	**-0.22**	**2.97**	290.39	177.08
		CGBH	0.00	-	3.03	62.29	1.91
$\mathcal{B}_1$-var	120	HGA-SC	**-1.76**	**-2.27**	**6.00**	3089.36	1688.71
		CGBH	0.00	-	7.80	1258.60	62.12
$\mathcal{B}_2$-var	15-60	HGA-SC	**-0.47**	**-0.47**	**2.37**	342.57	209.35
		CGBH	0.00	-	2.63	135.36	6.28
$\mathcal{B}_2$-var	120	HGA-SC	**-2.10**	**-2.75**	**5.00**	3678.24	2086.79
		CGBH	0.00	-	6.20	2577.19	241.00

Table 4: Summary of results for the scalability analysis

set $\mathcal{B}_1$-var. This confirms again the high quality of solutions found by our HGA-SC.

The results also show the scalability of HGA-SC. When increasing the route length, the average runtime of HGA-SC is almost double in the worst case (120-customer instances of set $\mathcal{B}_1$-var), and reaches up to 3678.24 seconds which is still adequate for daily or weekly planning. We also note that, to solve two sets $\mathcal{B}_1$-var and $\mathcal{B}_2$-var, the average runtime of CGBH is about 10-32 times longer than on the original instance sets $\mathcal{B}_1$ and $\mathcal{B}_2$. Therefore, the route length has a more serious impact on the performance of CGBH than that of HGA-SC.

Conclusion

In this article, we develop a hybrid genetic algorithm called HGA-SC to solve the VRPRDL problem, which has applications in scheduling the delivery services to the trunk of cars. Our algorithm has several new features, such as an implicit solution representation, a split procedure to fast evaluate chromosomes, local search operators, a swap-based mutation, and a set covering component. Experimental results obtained on the VRPRDL benchmark instances of the literature are compared with the state-of-the-art results, showing a good performance of our HGA-SC in terms of solution quality and stability. In addition, 49 new best known solutions are first found in this paper. We also conduct a sensitivity analysis to investigate the impact of two components mutation and set covering, which are rarely applied in genetic algorithms, on the algorithm performance. The results demonstrate the importance of these features. Future research directions include applying our metaheuristic to other problems with roaming delivery locations such as GVRPTW and VRPDO. To solve the existing instances of these problems, all components of HGA-SC need to be carefully redesigned to deal with the cases where the fleet size is limited.

Acknowledgments

This work is finished during the research stay of the corresponding author (Minh Hoàng Hà) at the Vietnamese Institute for Advanced Studies in Mathematics (VIASM). He wishes to thank this institution for their kind hospitality and support.

References

Alvarenga, G. B.; Mateus, G. R.; and De Tomi, G. 2007. A genetic and set partitioning two-phase approach for the

vehicle routing problem with time windows. *Computers & Operations Research*, 34(6): 1561–1584.

Balas, E.; and Simonetti, N. 2001. Linear Time Dynamic-Programming Algorithms for New Classes of Restricted TSPs: A Computational Study. *INFORMS Journal on Computing*, 13(1): 56–75.

Coppola, D. 2021. E-commerce worldwide - statistics & facts. https://www.statista.com/topics/871/online-shopping/#dossier-chapter1. Accessed: 2021-11-03.

Dumez, D.; Lehuédé, F.; and Péton, O. 2021. A large neighborhood search approach to the vehicle routing problem with delivery options. *Transportation Research Part B: Methodological*, 144: 103 – 132.

Dumez, D.; Tilk, C.; Irnich, S.; Lehuédé, F.; and Péton, O. 2021. Hybridizing large neighborhood search and exact methods for generalized vehicle routing problems with time windows. *EURO Journal on Transportation and Logistics*, 10: 100040.

Harbison, C. 2018. Amazon Car Delivery: How to Get Packages Delivered to Your Trunk with New Key Service. https://www.newsweek.com/amazon-car-delivery-key-trunk-how-cities-home-kit-where-service-899644. Accessed: 2021-11-03.

Holland, J. H. 1992. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

Irnich, S.; Toth, P.; and Vigo, D. 2014. *Chapter 1: The Family of Vehicle Routing Problems*, 1–33. MOS-SIAM Series on Optimization.

J. Hawkins, A. 2018. Amazon will now deliver packages to the trunk of your car. https://www.theverge.com/2018/4/24/17261744/amazon-package-delivery-car-trunk-gm-volvo. Accessed: 2021-11-03.

López-Ibáñez, M.; Dubois-Lacoste, J.; Cáceres, L. P.; Birattari, M.; and Stützle, T. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3: 43–58.

Moccia, L.; Cordeau, J.-F.; and Laporte, G. 2012. An incremental tabu search heuristic for the generalized vehicle routing problem with time windows. *Journal of the Operational Research Society*, 63(2): 232–244.

Oliver, I.; Smith, D.; and Holland, J. R. 1987. Study of permutation crossover operators on the traveling salesman problem. In *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*. Hillsdale, NJ: L. Erlhaum Associates, 1987.

Ozbaygin, G.; Karasan, O. E.; Savelsbergh, M.; and Yaman, H. 2017. A branch-and-price algorithm for the vehicle routing problem with roaming delivery locations. *Transportation Research Part B: Methodological*, 100: 115–137.

Prins, C. 2004. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12): 1985–2002.

Reyes, D.; Savelsbergh, M.; and Toriello, A. 2017. Vehicle routing with roaming delivery locations. *Transportation Research Part C: Emerging Technologies*, 80: 71–91.

Schneider, M.; and Löffler, M. 2019. Large composite neighborhoods for the capacitated location-routing problem. *Transportation Science*, 53(1): 301–318.

Schröder, C.; Gauthier, J.; Gschwind, T.; and Schneider, M. 2020. In-depth analysis of granular local search for capacitated vehicle routing. Technical report, Working Paper DPO-2020-03, Deutsche Post Chair—Optimization of Distribution.

Tilk, C.; Olkis, K.; and Irnich, S. 2021. The last-mile vehicle routing problem with delivery options. *OR Spectrum*, 1–28.

Toth, P.; and Vigo, D. 2003. The granular tabu search and its application to the vehicle-routing problem. *Informs Journal on Computing*, 15(4): 333–346.

Vidal, T.; Crainic, T. G.; Gendreau, M.; and Prins, C. 2013. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, 40(1): 475–489.

Vidal, T.; Crainic, T. G.; Gendreau, M.; and Prins, C. 2014. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234(3): 658–673.

Walteros, J. L.; Medaglia, A. L.; and Riaño, G. 2015. Hybrid algorithm for route design on bus rapid transit systems. *Transportation Science*, 49(1): 66–84.

Yuan, Y.; Cattaruzza, D.; Ogier, M.; Semet, F.; and Vigo, D. 2021. A column generation based heuristic for the generalized vehicle routing problem with time windows. *Transportation Research Part E: Logistics and Transportation Review*, 152: 102391.