

Memetic Search for Vehicle Routing with Simultaneous Pickup-Delivery and Time Windows

Shengcai Liu, Ke Tang*, Xin Yao

Guangdong Key Laboratory of Brain-Inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China

Abstract

The Vehicle Routing Problem with Simultaneous Pickup-Delivery and Time Windows (VRPSPDTW) has attracted much research interest in the last decade, due to its wide application in modern logistics. Since VRPSPDTW is NP-hard and exact methods are only applicable to small-scale instances, heuristics and meta-heuristics are commonly adopted. In this paper we propose a novel Memetic Algorithm with efficient local search and Extended neighborhood, dubbed MATE, to solve this problem. Compared to existing algorithms, the advantages of MATE lie in two aspects. First, it is capable of more effectively exploring the search space, due to its novel initialization procedure, crossover and large-step-size operators. Second, it is also more efficient in local exploitation, due to its sophisticated constant-time-complexity move evaluation mechanism. Experimental results on public benchmarks show that MATE outperforms all the state-of-the-art algorithms, and notably, finds new best-known solutions on 12 instances (65 instances in total). Moreover, a comprehensive ablation study is also conducted to show the effectiveness of the novel components integrated in MATE. Finally, a new benchmark of large-scale instances, derived from a real-world application of the JD logistics, is introduced, which can serve as a new and more challenging test set for future research.

Keywords: Vehicle routing problem, memetic algorithm, combinatorial optimization, industrial application

1. Introduction

Reverse logistics plays an important role in modern transportation. Generally, it is related to bi-directional flow of goods regarding delivery and pickup activities, where the former refers to shipping goods to the customers, while the latter refers to the opposite. Because of its significant effect on lowering costs associated with energy consumption and reducing the environmental impact, reverse logistics has been incorporated into many regular delivery systems in various fields such as library books distribution [1], grocery distribution [2], and parcel delivery [3].

In the literature, the problem involving bi-directional flow of goods has often been referred to as the pickup and delivery problem (PDP). According to the surveys on PDP [4, 5], it can be further categorized into 3 different types: 1) many-to-many PDP where each commodity may have multiple origins and destinations and any location may be the origin or destination of multiple commodities; 2) one-to-many-to-one PDP where some commodities are delivered from a depot to many customers, while other commodities are collected at customers and delivered to the depot; 3) one-to-one PDP where each commodity has

a single origin and a single destination between which it must be delivered. The most widely studied variant of the second type or one-to-many-to-one PDP, is the vehicle routing problem with simultaneous pickup and delivery (VRPSPD) [1, 6], due to the ever-growing trend toward recycling and product reuse.

This paper studies a practical situation of VRPSPD, which frequently occurs in modern logistics systems, e.g., JD logistics and CAINIAO logistics. In these systems, in addition to delivering commodities purchased by customers from online, one must also plan the collection of used, defective, or obsolete products from customers; moreover, in order to provide satisfactory service, either delivery or pickup needs to be operated within predefined time windows. In the literature, this problem is referred to as the vehicle routing problem with simultaneous pickup-delivery and time windows (VRPSPDTW) [7].

It has been shown that VRPSPDTW is NP-hard since it can be trivially reduced to VRPSPD (which is NP-hard) [8]. Hence exact algorithms [8, 7] can only be used to find optimal solutions for small-scale instances (with number of customers smaller than 25). As a result, researchers and practitioners are usually interested in developing meta-heuristics to find high-quality solutions within reasonable computational time. The adopted search paradigms include differential evolution [9], genetic algorithm [7], simulated annealing [10, 11], swarm intelligence optimization

*Corresponding author

Email addresses: liusc3@sustech.edu.cn (Shengcai Liu), tangk3@sustech.edu.cn (Ke Tang), xiny@sustech.edu.cn (Xin Yao)

[12], variable neighborhood search [13], and adaptive large neighborhood search [14].

However, despite the rich literature, in this area there still exist several important issues. The first is that none of the current methods is able to perform both well and robustly across different types of problem instances. In particular, on many practical instances, the solutions found by current methods are not satisfactory. The root cause for this is that current methods are generally not very effective in either exploration or exploitation. Typically they use a single insertion operator to construct initial solutions [7, 10, 11] and conduct local search via traditional small step size move operators [9, 7, 10–13]. As a result, the diversity among the generated solutions is rather limited and the search process tends to get trapped in local optima, leading to unsatisfactory exploration. Moreover, for existing methods the computational costs incurred by local search are generally very high [7, 11–13], which severely affects the efficiency of local exploitation. The second issue is that little attention has been paid to large-scale instances. Currently the only publicly available benchmark set in the literature [7] contains problem instances with customer number up to 100. However, with the ever-growing of big cities, a real-world application might involve many more customers [15]. Therefore, it is necessary to introduce a benchmark set containing large-scale real-world problem instances, to further facilitate the research in this area.

This paper is aimed at addressing the above issues. Specifically, we propose a highly effective memetic algorithm (MA) to solve VRPSPDTW. As an important area of evolutionary computation, MAs combine global search strategies (e.g., crossover) with local search heuristics, and have been shown to be very effective on a wide variety of combinatorial optimization problems [16–19]. More importantly, MAs are the state-of-the-art approaches for many variants of the vehicle routing problems (VRP) [20–31]. Since MA is a generic framework, appropriately instantiating it for a specific problem is non-trivial. Generally, effective MAs should make good use of domain-specific knowledge in its main algorithm components (e.g., initialization, crossover operator and local search), and meanwhile achieve a good balance between exploration and exploitation. The main contributions of this work can be summarized as follows.

1. From the algorithmic perspective, the proposed MA involves several novel ingredients, which simultaneously enable effective exploration and efficient exploitation. First, we propose an initialization procedure by combining a construction heuristic with population-based search in an intelligent way, which can construct an initial population with high diversity. Second, we propose a new crossover operator for VRPSPDTW based on route inheritance and regret-based insertion. Third, we design a highly effective local search procedure for VRPSPDTW, which can flexibly search in a large neighborhood of a solution

by switching between move operators with different step sizes. Moreover, we describe a sophisticated move evaluation process, which enables evaluating any neighborhood solution (issued from the move operators) in constant time, thus dramatically reducing the computational costs. Finally, we incorporate these components into the MA framework, and propose the **M**emetic **A**lgorithm with efficient local search and **E**xtended neighborhood (MATE) for VRPSPDTW.

2. From the computational perspective, MATE shows excellent performance on existing benchmark instances (65 instances in total). In particular, it outperforms all the four state-of-the-art algorithms. Notably, on 12 instances from the benchmark, MATE finds **new** best-known solutions.
3. From the benchmarking perspective, we introduce a new instance set for VRPSPDTW, which can serve as a new and more challenging benchmark in this field. Compared to existing synthetic ones, the new instances are derived from a real-world application of JD logistics, and are with larger scales. The evaluation results of MATE on the new benchmark are also reported.

The rest of the paper is organized as follows. Section 2 presents a literature review on the areas closely related to VRPSPDTW. Section 3 formally defines the problem. Section 4 first presents the framework of MATE, followed by its detailed implementation. Section 5 compares MATE with the state-of-the-art algorithms on the existing benchmark and introduces a new benchmark derived from a real-world application. Finally, Section 6 concludes the paper.

2. Related Work

As aforementioned, VRPSPDTW is a variant of VRPSPD. The latter was first studied in [1] for a book distribution system involving 22 customers and two vehicles. The solution was obtained by clustering customers into two groups and then solving the traveling salesman problem (TSP) for each group. The results in [1] showed that, compared to traditional one-directional logistics, bi-directional logistics can achieve substantial time/distance savings. Since then numerous studies have been conducted on VRPSPD. The proposed approaches can be categorized into three groups: exact approaches, heuristics and meta-heuristics. There exist several exact algorithms for VRPSPD in the literature, including branch-and-price algorithm [32] with commodity-flow formulation, branch-and-cut algorithm [33, 34] and branch-and-cut-and-price algorithm [35] with vehicle-flow formulation.

Compared to exact approaches, heuristics and meta-heuristics for solving VRPSPD have attracted much more research interest in the last decade, due to the fact that the problem is NP-hard [8]. A number of construction heuristics, tour partitioning [36], parallel savings heuristic

[37], and residual capacity and radical surcharge (RCRS) heuristic [2], have been proposed. Compared to construction heuristics, meta-heuristics equipped with local search procedures can often obtain even better solutions. Early research on meta-heuristics for solving VRPSPD mainly focused on tabu search (TS). Various techniques, such as record-to-record travel approximation [38] and reactive mechanism [39] have been proposed to enhance the performance of TS. Later many other meta-heuristics have also been applied to solve VRPSPD, such as iterated local search (ILS) [40], adaptive local search (ADL) [41], ant colony optimization (ACO) [42], particle swarm optimization (PSO) [43], and genetic algorithm (GA) [44]. One may refer to [6] for a comprehensive review on existing approaches for VRPSPD.

VRPSPD with time-window constraints, i.e., VRPSPD-TW, is probably the most studied variant of VRPSPD, due to its wide application in modern logistics. The problem was first introduced in [8], where the considered objective is to minimize the total travel distance (TD). A branch-and-cut-and-price algorithm was also proposed. Optimal solutions were obtained on small-scale instances with up to 20 customers. Later the same objective was considered in [9, 10], in which a differential evolution (DE) approach and a simulated annealing (SA) approach were proposed, respectively. The DE approach uses a decimal coding to construct an initial population and involves several new problem-specific move operators. It was tested on instances with 8 and 40 customers, and the results were competitive. The SA approach uses a sequential route construction heuristic to generate an initial solution, and then uses a SA procedure to improve the solution by searching in its neighborhood. The evaluation results of the approach on instances with 10, 15 and 50 customers were also reported.

Another line of research in VRPSPD-TW takes into account the number of used vehicles (NV), since the usage of a vehicle will result in its depreciation. In particular, the primary goal is to minimize NV, and the second one is to minimize TD. Such a setting was first considered in [7], in which a co-evolutionary GA (Co-GA) was proposed. Co-GA uses modified cheapest-insertion heuristics to generate initial solutions, and maintains two populations in the evolutionary process for diversification and intensification, respectively. The well-known Solomon benchmark [45] was modified in [7] to generate 65 instances with 10, 25, 50, and 100 customers. Co-GA was then compared with the commercial solver CPLEX and the results showed the former can find better solutions within a comparatively shorter period of time. Since then the benchmark introduced in [7] has become the most widely used benchmark in this area. For convenience, we refer to this benchmark as *wc* (authors' initials) set in this paper.

Following [7], a number of approaches, a parallel SA approach (p-SA) [11], a swarm intelligence approach (IGAFSA) [12], an adaptive large neighborhood search approach (ALNS-PR) [14] and a two-stage hybrid approach (VNS-BSTS) [13], have been proposed to solve VRPSPD-TW concerning

minimizing NV and TD. The SA approach adopts a slow cooling schedule and a randomized local search procedure. It was tested on three 10-customer instances, three 25-customer instances, three 50-customer instances and six 100-customer instances from the *wc* set. On average, the solutions found by it were 0.22% better (regarding TD) than Co-GA. p-SA is a parallel variant of the SA approach. It uses the RCRS heuristic to generate initial solutions and adopts a master-slave paradigm in which multiple SA procedures are run independently and simultaneously. Competitive results on the *wc* set were obtained, with 28 new best-known solutions in total (12 with lower NV and 16 with lower TD). In addition, the evaluation results of p-SA on 30 large-scale instances, with customers of 200, 400, 600, 800 and 1000, were also reported. However, the large-scale benchmark is not publicly available. IGAFSA is a nature-inspired approach that allows infeasible solutions to enhance diversification. On 39 instances from the *wc* set, it found better solutions with lower TD than p-SA. ALNS-PR uses seven removal operators and three reinsertion operators to repeatedly destroy and reconstruct a solution, and involves a path-relinking component for intensification. The results in [14] showed that ALNS-PR significantly outperformed all the previous approaches on the *wc* set. It found new best-known solutions for 48 instances in total (17 with lower NV and 31 with lower TD). VNS-BSTS is the most recent approach for VRPSPD-TW. It has a two-stage solution framework. In the first stage, it utilizes variable neighborhood search with a learning-based objective function to minimize NV, while in the second stage it uses tabu search to further minimize TD. VNS-BSTS showed competitive performance on the *wc* set, and found several new best-known solutions.

3. Notations and Problem Definition

Given a number of customers who require both pickup service and delivery service within certain time windows, the target of VRPSPD-TW is to send out a fleet of capacitated vehicles, which are stationed at a depot, to meet the customer demands with the minimum total cost. Formally, the problem is defined on a complete graph $G = (V, E)$ with $V = \{0, 1, 2, \dots, M\}$ as the node set and E as the arc set defined between each pair of nodes, i.e., $E = \{\langle i, j \rangle | i, j \in V, i \neq j\}$. For convenience, the depot is always denoted as 0 and the customers are denoted as $1, \dots, M$. Each arc $\langle i, j \rangle \in E$ is associated with a travel distance $dist(i, j)$ and a travel time $time(i, j)$. Each node $i \in V$ is associated with 5 attributes, i.e., a delivery demand d_i , a pickup demand p_i , a time window $[a_i, b_i]$ and a service time s_i . d_i is the amount of goods to deliver from the depot to customer i and p_i is the amount of goods to pick up from customer i that must be delivered to the depot. a_i and b_i are the start and the end of the time window in which the customer receives service. Arrival of a vehicle at customer i before a_i results in a wait before service can begin; while arrival after b_i is infeasible. Finally,

s_i is the time spent by the vehicle to unload/load goods at customer i . Note that for the depot, i.e., 0, a_0 and b_0 are the earliest time the vehicles can depart from the depot and the latest time the vehicles can return to the depot, respectively, and $d_0 = p_0 = s_0 = 0$.

A fleet of J identical vehicles, each with a capacity of Q and a dispatching cost u_1 , is initially located at the depot. The vehicles depart from the depot and then serve the customers, and finally return to the depot. Thus a solution S to VRPSPDTW is represented by a set of vehicle routes, i.e., $S = \{R_1, R_2, \dots, R_K\}$, in which each route R_i consists of a sequence of nodes that the vehicle visits, i.e., $R_i = (h_{i,1}, h_{i,2}, \dots, h_{i,L_i})$, where $h_{i,j}$ is the j -th node visited in R_i , and L_i is the length of R_i . For the sake of brevity, in the following we temporarily omit the subscript i in R_i , i.e., $R = (h_1, h_2, \dots, h_L)$. The total travel distance of R , denoted as $TD(R)$, is:

$$TD(R) = \sum_{j=1}^{L-1} \text{dist}(h_j, h_{j+1}). \quad (1)$$

The time of arrival at and the time of departure from h_j , denoted as $\text{arr}(h_j)$ and $\text{dep}(h_j)$, respectively, can be computed recursively via the following equations:

$$\begin{aligned} \text{dep}(h_1) &= a_0 \\ \text{arr}(h_j) &= \text{dep}(h_{j-1}) + \text{time}(h_{j-1}, h_j), \quad j > 1 \\ \text{dep}(h_j) &= \max \{ \text{arr}(h_j), a_{h_j} \} + s_{h_j}, \quad j > 1. \end{aligned} \quad (2)$$

The vehicle load on arrival at h_j , denoted as $\text{load}(h_j)$, is:

$$\begin{aligned} \text{load}(h_1) &= \sum_{j=1}^L d_{h_j} \\ \text{load}(h_j) &= \text{load}(h_{j-1}) - d_{h_{j-1}} + p_{h_{j-1}}, \quad j > 1. \end{aligned} \quad (3)$$

The total cost of S , denoted as $TC(S)$, consists of two parts: the dispatching cost of the used vehicles, which is $u_1 \cdot K$, and the transportation cost, which is the total travel distance of S multiplied by cost per unit of travel distance u_2 . The objective is to find a S with the minimum TC, as presented in Eq. (4):

$$\begin{aligned} \min_S TC(S) &\triangleq u_1 \cdot K + u_2 \cdot \sum_{i=1}^K TD(R_i) \\ \text{s.t. : } &K \leq J \\ &h_{i,1} = h_{i,L_i} = 0, \quad 1 \leq i \leq K \\ &\sum_{i=1}^K \sum_{j=2}^{L_i-1} \mathbb{I}[h_{i,j} = x] = 1, \quad 1 \leq x \leq M \\ &\text{load}(h_{i,j}) \leq Q, \quad 1 \leq i \leq K, 1 \leq j \leq L_i \\ &\text{arr}(h_{i,j}) \leq b_{h_{i,j}}, \quad 1 \leq i \leq K, 2 \leq j \leq L_i \\ &\text{dep}(h_{i,1}) \geq a_0 \text{ and } \text{arr}(h_{i,L_i}) \leq b_0, \quad 1 \leq i \leq K \end{aligned} \quad (4)$$

where the constraints are: 1) the number of used vehicles cannot exceed the number of available ones; 2) each

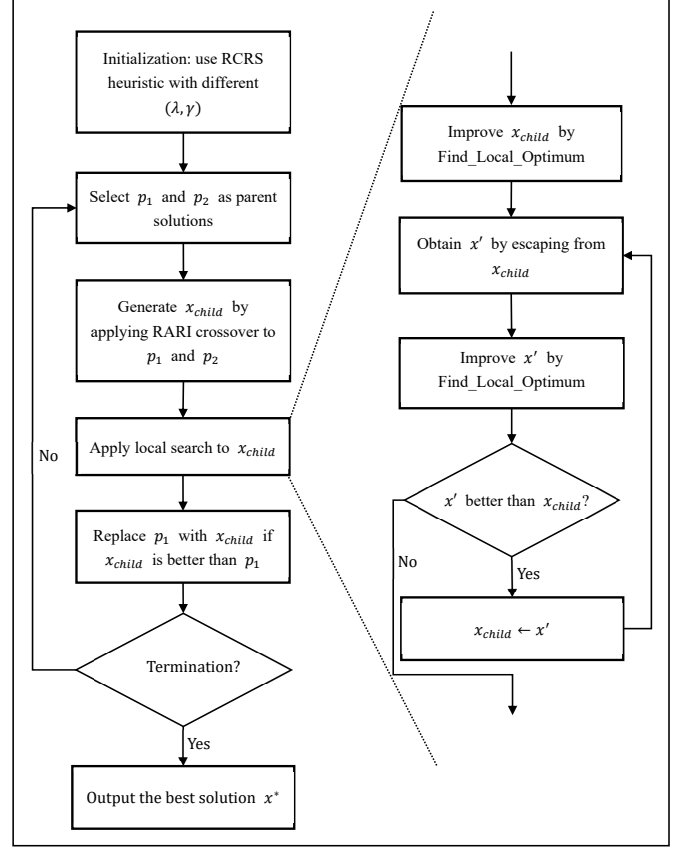


Figure 1: The flow chart of the proposed memetic algorithm.

route must start from and return to the depot; 3) each customer must be served exactly once (note $\mathbb{I}[\cdot]$ is the indicator function); 4) vehicles cannot be overloaded during transportation; 5) service to each customer must be performed within that customer's time window; 6) vehicles can only depart after the start of the time window of the depot (a_0) and must return to the depot before the end of its time window (b_0). It is noted that the two different objectives considered in the literature of VRPSPDTW (see Section 2) are both special cases of Eq. (4). Specifically, the dispatching cost u_1 could be set to 0 to only minimize TD. The ratio of u_1 to u_2 , i.e., u_1/u_2 , could be set to a sufficiently large number such that minimizing NV is the primary goal and minimizing TD is the second one.

The main characteristics of VRPSPDTW lie in the capacity aspect and the temporal aspect. The former is that customers can simultaneously have delivery demand and pick-up demand. Since these two types of demands have different effects on the vehicle load (one will increase the load while the other will decrease the load), such a characteristic causes difficulty assigning appropriate customers to vehicles under the capacity constraint. The latter is that each customer is associated with a hard time window, which further increases the difficulty planning the service order for a particular vehicle under the time-window constraint. In addition to taking the above constraints into

Algorithm 1: The General Framework of MATE

input : A VRPSPDTW instance; population size, N ; longest consecutive generations without improvement, G_{max}

output : the best found solution x^*

```
1  $\{x_1, x_2, \dots, x_N\} \leftarrow \text{Initialization}();$ 
2  $x^* \leftarrow \text{Select\_Best}(x_1, x_2, \dots, x_N);$ 
3 repeat
4    $\pi(\cdot) \leftarrow$  a random permutation of  $1, \dots, N$ ;
5   for  $i \leftarrow 1$  to  $N$  do
6      $p_1 \leftarrow x_{\pi(i)}, p_2 \leftarrow x_{\pi(i+1)};$ 
7      $x_{child} \leftarrow \text{Crossover}(p_1, p_2);$ 
8      $x_{child} \leftarrow \text{Local\_Search}(x_{child});$ 
9      $x_{\pi(i)} \leftarrow \text{Select\_Best}(x_{child}, p_1);$ 
10  end
11   $x^* \leftarrow \text{Select\_Best}(x^*, x_1, x_2, \dots, x_N);$ 
12 until  $x^*$  not improved in the last  $G_{max}$  generations;
13 return  $x^*$ 
```

account, considering the exponential search space of VRPSPDTW, it is also crucial for the algorithm to maintain sufficient exploration as well as efficient exploitation, to find high-quality feasible solutions.

4. Memetic Algorithm for VRPSPDTW

In this section we first describe the general framework of MATE, and then elaborate on its three main components (initialization, crossover and local search) in turn. The flow chart of the algorithm is also illustrated in Figure 1. Note that these components ensure that the generated solutions always satisfy the constraints of capacity and time windows. Moreover, if a generated solution's route number (i.e., the number of used vehicles) exceeds the number of available ones, then the solution will be immediately discarded. In other words, during the entire run of MATE, each individual in the population always corresponds to a feasible solution to the VRPSPDTW instance, and its fitness is the inverse of its total cost as defined in Eq. (4).

4.1. General Framework

As presented in Algorithm 1, the proposed algorithm, called MATE, combines population-based evolutionary search and local optimization. More specifically, the population consists of N individuals, where N is a parameter. After initialization (line 1), MATE enters an evolutionary process. In each generation (lines 4-11), each individual in the population is selected once as parent p_1 and once as parent p_2 , in a random order (lines 4 and 6). Such a mechanism could promote the population diversity since each individual has exactly the same chance of being selected. For each pair of parents, MATE generates an offspring solution via the crossover operator (line 7), and then tries

Algorithm 2: The Initialization Procedure

input : population size, N

output : $\{x_1, x_2, \dots, x_N\}$

```
1  $count \leftarrow 1;$ 
2 for  $i \leftarrow 1$  to  $\sqrt{N}$  do
3   for  $j \leftarrow 1$  to  $\sqrt{N}$  do
4      $\lambda \leftarrow \frac{1}{\sqrt{N}-1} \cdot (i-1);$ 
5      $\gamma \leftarrow \frac{1}{\sqrt{N}-1} \cdot (j-1);$ 
6      $x_{count} \leftarrow \text{RCRS}(\lambda, \gamma);$ 
7      $count \leftarrow count + 1;$ 
8   end
9 end
10 return  $\{x_1, x_2, \dots, x_N\}$ 
```

to improve it by local search (line 8). After that, the better one among the offspring solution and p_1 will replace the population member selected as p_1 (line 9). Therefore, no replacement will take place if the offspring solution is worse than p_1 . In other words, the loss of population diversity is allowed only if the average fitness of the population is improved. The iterations of generation in MATE terminate when the best found solution has not been improved (line 11) in the last consecutive G_{max} generations (line 12), in which case the algorithm is considered to have converged. Finally, the best found solution is returned (line 13).

4.2. Initialization

The initial population is constructed using the RCRS heuristic, which is an extension of the cheapest-insertion heuristic. A detailed description of the RCRS heuristic can be found in [2]. Specifically, RCRS heuristic builds a route by iteratively inserting an unassigned customer into the route at the position with the minimal value w.r.t the RCRS criterion, until no feasible insertions (regarding constraints of capacity and time windows) exist. If no feasible insertions exist, a new route is activated. The above procedure is repeated until all customers are assigned. The core of the RCRS heuristic is the RCRS criterion, which assesses how good a potential insertion is. The conventional cheapest-insertion heuristic uses the TD criterion, i.e., the extra travel distance caused by inserting a customer, which is usually considered short-sighted. The RCRS criterion extends the TD criterion in two aspects. First, the remaining vehicle capacity after an insertion is taken into account, i.e., the residual capacity (RC) criterion, which measures the degrees of freedom for future insertions. Second, the distances of customers to the depot are considered, i.e., the radial surcharge (RS) criterion, which seeks to avoid the unfavorable extra travel distances caused by inserting the remotely located customers which are "left over" to a late stage of the insertion procedure. RCRS criterion is a combination of RC criterion and RS criterion, with two weighting parameters $\lambda, \gamma \in [0, 1]$. Unfortunately, the best values of λ and γ vary across different problem instances.

Algorithm 3: The RARI Crossover Operator

```
input : parent solutions  $p_1, p_2$ 
output :  $x_{child}$ 
1  $x_{child} \leftarrow$  initialize an empty solution with no
   routes;
2 repeat
3   | copy random route from  $p_1$  to  $x_{child}$ ;
4   | copy random route from  $p_2$  to  $x_{child}$ ;
5 until no more inherited routes are feasible;
6 /* --- regret-based insertion --- */
7  $U \leftarrow$  all the remaining unassigned customers;
8 while  $U \neq \emptyset$  do
9   | foreach node  $v \in U$  do
10  | | calculate  $regret(v)$  according to Eq. (5);
11  | end
12  |  $v^* \leftarrow \arg \max_{v \in U} regret(v)$ ;
13  | Insert  $v^*$  into  $x_{child}$  at its best insertion
   | position;
14  |  $U \leftarrow U \setminus \{v^*\}$ ;
15 end
16 return  $x_{child}$ 
```

For a given instance it is hard to determine a good choice of λ and γ in advance.

However, in MATE we can take advantage of the population based search by trying different values of λ and γ for different individuals. As presented in Algorithm 2, for each of λ and γ , we start with the value of 0, and gradually increase it with a step size of $1/(\sqrt{N} - 1)$ until it reaches 1 (lines 4-5). In summary, for each of λ and γ , we use \sqrt{N} different values ranging from 0 to 1 that are equally spaced, thus obtaining a total of N different combinations of (λ, γ) , based on each of which the RCRS heuristic is used to construct an initial solution (line 6). Finally, it is required that the parameter N , i.e., the population size, is a square number., e.g., 4, 9, 16, etc.

Compared to previous methods [2, 11] that also use the RCRS heuristic to construct initial solutions, the proposed initialization approach has at least two-fold advantages. First, it eliminates the need of tedious tuning of the two parameters λ and γ of the RCRS heuristic, since it simultaneously tries very different values of λ and γ for different initial individuals. Second, it enables good coverage on the design space of (λ, γ) , which is beneficial to constructing an initial population with high diversity, thus promoting more sufficient exploration in the search space. The effect of the proposed initialization procedure on MATE’s performance is also investigated in the experiments (see Section 5.5).

4.3. Crossover Operator

An effective crossover operator should be able to transmit useful building blocks from parents to offspring, and meanwhile incorporate domain-specific heuristics to generate high-quality offspring that are significantly different

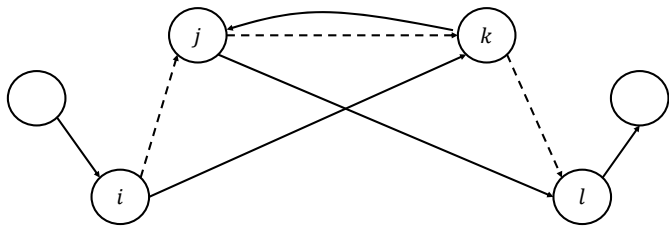
from either of the parents. In this paper, we propose a new route-assembly-regret-insertion (RARI) crossover.

As presented in Algorithm 3, RARI crossover first repeatedly chooses a random route from each parent solution in turns to insert into the offspring solution until no feasible insertion exists (lines 2-5). In other words, the offspring solution will inherit as many routes as possible from each parent equally. After that, the remaining unassigned customers (if there exist) will be inserted into the offspring solution to form a complete solution. In the literature of VRP, one commonly used procedure to insert these nodes is the two-step insertion [46], which first tries inserting them into existing routes of the offspring solution and then inserts the remaining nodes by cheapest-insertion heuristic. Here, we propose to use regret-based insertion (lines 7-15) to insert the unassigned customers. Such an approach uses a regret value, which represents the expected cost of inserting a node not in this iteration but in a future iteration, to assess how good a potential insertion is. Compared to cheapest-insertion which is usually considered short-sighted, regret-based insertion can avoid the postponing issue of cheapest-insertion heuristic—placing “difficult” nodes (which are expensive to insert) late in the process where there exist few opportunities for inserting them as many of the routes are already “full”. Moreover, compared to two-stage insertion, regret-based insertion is a simpler one-stage approach that simultaneously considers both the insertion of customers into existing routes and new routes. The comparison between two-stage insertion and regret-based insertion is also conducted in the experiments (see Section 5.5).

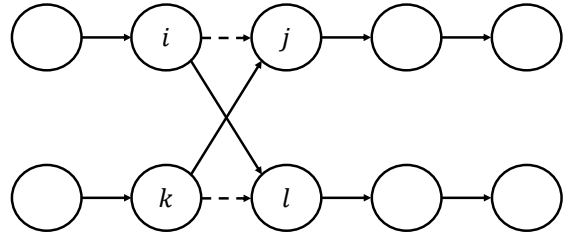
Specifically, the regret value for a node v is obtained as follows. Supposing that there exist m routes in the current partial solution, i.e., $\{R_1, R_2, \dots, R_m\}$, for each route R_i , the position and the cost c_i of the best feasible insertion (based on the objective value) of v into R_i is calculated. Moreover, the cost of inserting v into an empty route, i.e., a route from the depot to v and back to the depot, is also calculated, which is denoted as c_0 . Then c_0, c_1, \dots, c_m are sorted in ascending order, and the sorting results are denoted as $c_{\pi(0)}, c_{\pi(1)}, \dots, c_{\pi(m)}$, i.e., $c_{\pi(0)} \leq c_{\pi(1)} \leq \dots \leq c_{\pi(m)}$. The regret value of v , denoted as $regeret(v)$, is the difference in the cost of inserting v in its best route and its second best route:

$$regret(v) = c_{\pi(1)} - c_{\pi(0)}. \quad (5)$$

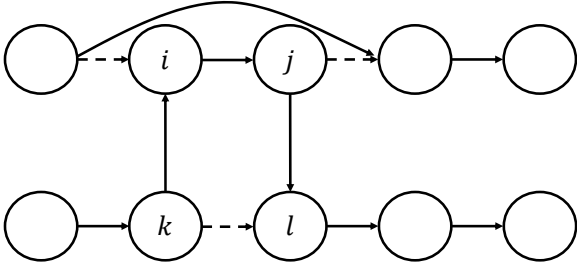
In each iteration of the regret-based insertion (lines 9-14), first the regret value of each currently unassigned customer is calculated (lines 9-11), and then the one with the maximum regret value (ties are broken by selecting the insertion with the lowest cost) will be inserted at its best feasible position (lines 12-14). Informally speaking, in each iteration the insertion that we will regret most if it is not done now will be carried out. The iterations terminate when all unassigned customers have been inserted into the offspring solution.



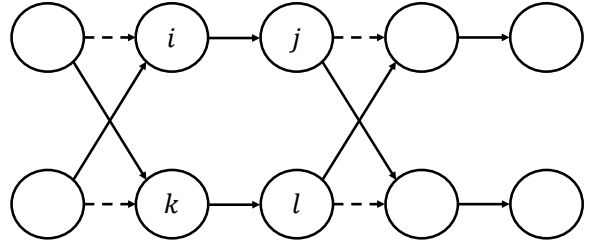
(a) *2-opt*. Inverting a subsequence of two consecutive customers j and k in a route.



(b) *2-opt**. Reconnecting the first part of one route at node i with the second part of another route at node l and vice versa.



(c) *or-opt*. Removing a subsequence of two consecutive customers i and j from a route, and then reinserting it between k and l in another route.



(d) *swap*. Exchanging a subsequence of two customers i and j in a route with a subsequence of two customers k and l in another route.

Figure 2: Illustrative examples of all move operators used. The dashed lines and the solid lines indicate the removed arcs and the new arcs in the routes due to performing the move, respectively.

4.4. Local Search Procedure

The local search procedure is the core component of MATE. Its effectiveness has a decisive impact on the performance of the algorithm. Existing VRPSPDTW algorithms [7, 11, 12] typically conduct local search via some traditional move operators, which modify only a small part of the current solution. In other words, these operators have small search step sizes which are expected to perform well in small search space. However, for difficult problems with large search spaces and many local optima, they may perform unsatisfactorily. Moreover, for existing algorithms the computational costs incurred by local search are very high, often accounting for the major part of the total costs. As a result, there is large room for further reducing the computational complexity of local search. To address these issues, we propose a novel local search procedure which can simultaneously achieve the following three goals. First, it can quickly identify high-quality local optima in a small search space. Second, it can jump out from the current local optimum to other promising regions. Third, its computational complexity is very low such that the incurred computational cost is reasonable.

To accomplish the first two goals, the local search procedure switches between move operators with different step sizes. As demonstrated in Algorithm 4, given a solution x , it first finds the local optimum around x in a small region defined by several traditional move operators (line 1). Then, it extends the search step size by applying the removal-and-reinsertion operator (lines 3-7), trying to jump

out from the local optimum. After that, the step size will be reduced again to identify the local optimum x' in the new local region (line 9). If x' is better than the current solution x , the latter will be updated and then used as the base solution for the next iteration (line 10). Otherwise the procedure will terminate (line 11) and the current solution x is returned (line 13).

4.4.1. Finding Local Optimum

To identify the local optimum around a given solution, the embedded Find_Local_Optimum sub-procedure in Algorithm 4 utilizes several traditional move operators [47], namely *2-opt*, *2-opt**, *or-opt* and *swap*, as illustrated in Figure 2. The *2-opt* operator inverts a subsequence of two consecutive customers in a route. The *2-opt** operator removes two arcs from two different routes to divide each route into two parts, and then reconnects the first part of the first route with the second part of the second route and vice versa. The *or-opt* operator removes a subsequence of one or two consecutive customers from a route, and then reinserts it into another position of the same route or a different route. The *swap* operator exchanges two subsequences of one or two consecutive customers, which may be on the same route (but not overlapping with each other), or on different routes.

Given a solution x , a best-improvement search is conducted in the neighborhoods of x defined by the four above-described move operators. That is, all solutions that can be reached by applying either of the four operators to x

Algorithm 4: The Local Search Procedure

input : solution x ; the lower and upper bounds
for the proportion of the removed nodes,
 ω_1, ω_2
output : x

```
1  $x \leftarrow \text{Find\_Local\_Optimum}(x)$ ;  
2 while true do  
3    $x' \leftarrow x$ ;  
4   /* escape from local optimum */  
5    $q \leftarrow$  a random number  $\in [\omega_1 M, \omega_2 M]$ ;  
6    $U \leftarrow$  remove  $q$  nodes from  $x'$ ;  
7   Insert  $U$  into  $x'$  via regret-based insertion;  
8   /* exploit the new local region */  
9    $x' \leftarrow \text{Find\_Local\_Optimum}(x')$ ;  
10  if  $x'$  is better than  $x$  then  $x \leftarrow x'$ ;  
11  else break;  
12 end  
13 return  $x$ 
```

are evaluated, and the best feasible solution among them, say \bar{x} , is then compared with x . If \bar{x} is better than x , the latter will be updated. This procedure is repeated until no further improvement can be found. By this means, it is ensured that a local optimum has been reached in each neighborhood w.r.t the move operators.

4.4.2. Escaping from Local Optimum

The above-described move operators modify only a small part (one or two routes) of the solution; thus the neighborhoods defined by them are actually a small region around the solution. Once a local optimum has been found in the region, an operator with a bigger step size is needed for jumping out of it [48, 49, 20]. We propose to use a removal-and-reinsertion operator to accomplish this goal. This operator first removes a number of customers from the solution (lines 5-6 in Algorithm 4), and then reinserts them into the solution again (line 7 in Algorithm 4). The number of the removed nodes, i.e., q , is a random number in $[\omega_1 M, \omega_2 M]$, where M is the total number of customers and ω_1, ω_2 are two parameters satisfying $0 < \omega_1 < \omega_2 < 1$. In this paper, ω_1, ω_2 are set to 0.2 and 0.4, respectively. In other words, 20-40% of all customers in the solution are rearranged by the removal-and-reinsertion operator; thus it is very likely that the obtained new solution will be quite different from the original one.

The node removal procedure seeks to remove customers that are correlated. More specifically, it first randomly removes a customer. Then two customers that are most correlated to the last removed customer are identified. After that, roulette wheel selection is used to select one of these two customers to remove. This procedure is repeated until q customers have been removed in total. The correlation of a customer j to a customer i is the weighted sum of the distance, the minimum waiting time, and the minimum

time-window violation on a direct service from i to j :

$$\begin{aligned} corr(i, j) = & dist(i, j) + \eta \cdot [\max\{a_j - time(i, j) - s_i - b_i, 0\} \\ & + \gamma \cdot \max\{a_i + s_i + time(i, j) - b_j, 0\}], \end{aligned} \quad (6)$$

where η is a factor that rescales time into distance and equals to the average distance between all nodes divided by the average travel time between all nodes. γ is a positive penalty factor (i.e., 10) for the time-window violation. The lower $corr(i, j)$ is, customer j is more correlated to customer i . Intrinsically, Eq. (6) measures how good customer j is as the next serviced one after customer i . Therefore it is expected to be reasonably easy to shuffle those highly-correlated customers (which have been removed from the current solution) around and thereby create perhaps new better solutions. After node removal, the node reinsertion is conducted once again using the regret-based insertion heuristic (see lines 6-15 in Algorithm 3), ensuring the final solution is feasible.

4.4.3. Move Evaluation in Constant Time

To accomplish the third goal of the local search procedure, i.e., keeping its computational complexity as low as possible, we propose to optimize the time complexity of move evaluation, i.e., calculating the cost and checking the feasibility of the solutions issued from the move operators (*2-opt*, *2-opt**, *or-opt* and *swap*), since it is the most time-consuming part of the local search procedure with a huge number of solutions being generated and evaluated. Recall that these move operators will modify at most two routes of a solution. Hence, to evaluate a move, one only needs to evaluate the changed routes, which could be trivially done in $O(n)$ by traversing the routes, where n is the route length. In this paper, we propose a new move evaluation approach for VRPSPDTW which has constant time complexity, i.e., $O(1)$.

The basic idea of sequence-based evaluation is that any move can be viewed as a separation of routes into subsequences, which are then concatenated into new routes. As illustrated in Figure 3, supposing the *or-opt* operator relocates a node from route R_1 to route R_2 (the upper part of the figure), the two new routes R_1 and R_2 issued from this move can be evaluated using concatenation of the subsequences (the lower part of the figure). Let σ and \oplus denote a subsequence and the concatenation operator, respectively. Let σ^0 denote a subsequence involving a single node i . For each route in a solution, we can recursively compute the following attributes of its all subsequences by concatenation.

Distance. The travel distance of σ^0 , $W(\sigma^0)$, is always 0. Moreover, the travel distance of a concatenation of two subsequences σ_1 and σ_2 is the sum of $W(\sigma_1)$ and $W(\sigma_2)$ plus the distance from the last node of σ_1 (denoted by j)

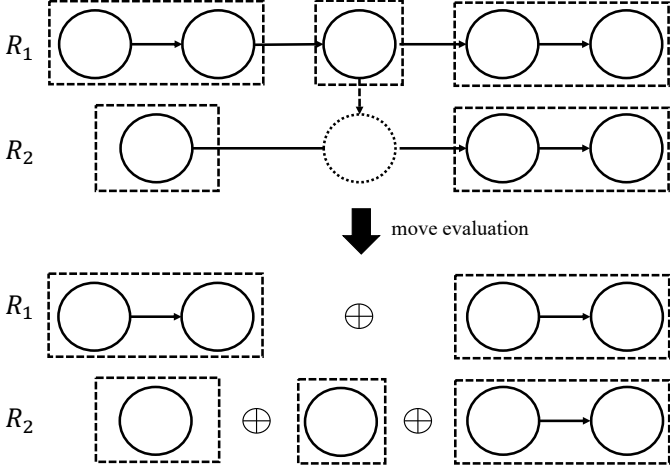


Figure 3: An illustrative example of evaluating a *or-opt* move, which relocates a node from route R_1 to route R_2 , using data of precomputed subsequences (dashed boxes) and the concatenation operator \oplus .

to the first node of σ_2 (denoted by k).

$$\begin{aligned} W(\sigma^0) &= 0 \\ W(\sigma_1 \oplus \sigma_2) &= W(\sigma_1) + W(\sigma_2) + \text{dist}(j, k). \end{aligned} \quad (7)$$

Capacity. Let $C_E(\sigma)$, $C_H(\sigma)$ and $C_L(\sigma)$ denote the initial, the highest, and the final loads on the vehicle during σ , respectively. For σ^0 , its initial load and final load are the delivery demand d_i and the pick-up demand p_i , respectively; its highest load is the maximum of these two. For a concatenation of σ_1 and σ_2 , its initial load is the sum of the initial loads of σ_1 and σ_2 , i.e., the total delivery demands of σ_1 and σ_2 ; its final load is the sum of the final loads of σ_1 and σ_2 , i.e., the total pick-up demands of σ_1 and σ_2 . The highest load during $\sigma_1 \oplus \sigma_2$ is the maximum of the highest loads during σ_1 and σ_2 , when concatenating σ_1 and σ_2 .

$$\begin{aligned} C_E(\sigma^0) &= d_i \\ C_L(\sigma^0) &= p_i \\ C_H(\sigma^0) &= \max\{C_E(\sigma^0), C_L(\sigma^0)\} \\ C_E(\sigma_1 \oplus \sigma_2) &= C_E(\sigma_1) + C_E(\sigma_2) \\ C_L(\sigma_1 \oplus \sigma_2) &= C_L(\sigma_1) + C_L(\sigma_2) \\ C_H(\sigma_1 \oplus \sigma_2) &= \max\{C_H(\sigma_1) + C_E(\sigma_2), \\ &\quad C_L(\sigma_1) + C_H(\sigma_2)\}. \end{aligned} \quad (8)$$

Time Window. Let $D(\sigma)$ denote the minimum duration on σ . For the arrival time to the first node of σ , there exists an interval $[E(\sigma), L(\sigma)]$ which allows the minimum duration. For σ^0 , its minimum duration is the service time s_i , while $E(\sigma)$ and $L(\sigma)$ are the start and the end of the time windows of the node of σ^0 . For a concatenation of σ_1 and σ_2 , the following equations enable computing the same data on the concatenation of two subsequences σ_1 and σ_2 (let j and k denote the first node of σ_1 and the last node

of σ_2 , respectively).

$$\begin{aligned} D(\sigma^0) &= s_i \\ E(\sigma^0) &= a_i \\ L(\sigma^0) &= b_i \\ \Delta &= D(\sigma_1) + \text{time}(j, k) \\ \Delta_{WT} &= \max\{E(\sigma_2) - \Delta - L(\sigma_1), 0\} \\ \Delta_{TW} &= E(\sigma_1) + D(\sigma_1) + \text{time}(i, j) - L(\sigma_2) \\ D(\sigma_1 \oplus \sigma_2) &= D(\sigma_1) + D(\sigma_2) + \text{time}(j, k) + \Delta_{WT} \\ E(\sigma_1 \oplus \sigma_2) &= \max\{E(\sigma_2) - \Delta, E(\sigma_1)\} - \Delta_{WT} \\ L(\sigma_1 \oplus \sigma_2) &= \min\{L(\sigma_2) - \Delta, L(\sigma_1)\}. \end{aligned} \quad (9)$$

Move Evaluation. The sequence-based evaluation approach is used in the best-improvement search in the neighborhoods of solution S , i.e., the Find_Local_Optimum sub-procedure. Specifically, the attributes of all the subsequences (and their reversal) of S are precomputed using the above equations, and will be used for checking the feasibility and computing the costs of routes issued from the move operators. Supposing a route R is generated by applying a move operator and $R = \sigma_1 \oplus \sigma_2 \oplus \dots \oplus \sigma_t$. We first retrieve the precomputed attributes of $\sigma_1, \dots, \sigma_t$, and then calculate the corresponding attributes of R by applying the above equations. The feasibility of R can be checked in the following way: 1) R is feasible regarding time-window constraint if $\Delta_{TW} \leq 0$; 2) R is feasible regarding capacity constraint if $C_H \leq C$, where C is the vehicle capacity. Moreover, we use the total travel distance W to calculate the transportation cost of R , $u_2 \cdot W$, where u_2 is the cost per unit of travel distance. The four move operators used in this paper correspond to a concatenation of less than five subsequences, i.e., $t \leq 5$. Hence, given the data on subsequences, any move evaluation is performed in constant time. Note that if S is updated, the attributes of the changed routes need to be recomputed in $O(n^2)$, where n is the route length. However, this is acceptable since in the search process the number of updates is smaller in orders of magnitude than the number of move evaluation.

4.5. Discussion

It is worth mentioning that the novel components of MATE, i.e., the initialization, crossover and local search procedures, are also applicable to other VRP variants. The reason is that the core components of these procedures, i.e., the RCRS heuristic, the route-inheritance heuristic equipped with regret-based insertion, and the move operators with different step sizes, can be smoothly used as long as the problem being solved involves capacitated vehicles and concerns minimizing the total travel distance, which is very common in many VRP variants. Moreover, the efficient move evaluation approach can also be used to accelerate existing VRPSPDTW algorithms, since it is applicable to all classical move operators (not just the ones used in this paper) which correspond to the concatenation of finite number of subsequences.

5. Computational Study

To evaluate the effectiveness of MATE, we compared it with a number of state-of-the-art algorithms on existing benchmark set and a new benchmark set derived from a real-world application. After that, we conducted a comprehensive ablation study to assess the contribution of the novel components integrated in MATE. Finally, the effect of different values of the parameter N (the population size) was investigated. The source code of MATE, as well as all the benchmark sets used in the experiments, are made at <https://github.com/senshineL/VRPenstein>.

5.1. Benchmark Sets

The commonly used (also the only publicly available) benchmark set in the literature of VRPSPDTW, referred to as the *wc* set in this paper (see Section 2), was generated by [7] through modifying the well-known Solomon benchmark [45]. The *wc* set contains 65 instances in total, including 9 small-scale instances (three 10-customer instances, three 25-customer instances, and three 50-customer instances) and 56 medium-scale instances (100-customer instances). Note all these instances are defined in the two-dimensional Euclidean space; that is, each node i (customer or depot) has a coordinate (x_i, y_i) and the distance between two nodes i and j is the Euclidean distance, i.e., $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. This is a special case of the problem formulation given in this paper (see Section 3), where the distances between nodes are explicitly given no matter whether they are defined in the Euclidean plane. According to the distribution of the customers' locations as well as the intensity of the time-window constraints and the capacity constraints, instances in the *wc* set could be further categorized into 6 types (subsets), namely Cdp1~, Cdp2~, Rdp1~, Rdp2~, RCdp1~ and RCdp2~, where the categories refer to:

1. Cdp: customers' locations are clustered;
2. Rdp: customers' locations are uniform randomly distributed;
3. RCdp: customers' locations are a mix of random and clustered locations;
4. Type 1: customers have narrow time windows and vehicle's capacity is small;
5. Type 2: customers have large time windows and vehicle's capacity is large.

For all instances in the *wc* set, minimizing NV (number of used vehicles) is the primary goal and minimizing TD (total distance) is the second one. To meet this, for the objective function Eq. (4), the ratio of the dispatching cost of each vehicle (u_1) to the cost per unit of travel distance (u_2), i.e., u_1/u_2 , should be set to a sufficiently large number (see Section 3). Following [7], in the experiments we set u_1 and u_2 to 2000 and 1, respectively.

As aforementioned, the *wc* set contains only synthetic instances of small or medium scales. To further assess the potential of the algorithms, we introduce a new benchmark set derived from the distribution system of JD logistics. In

Table 1: Algorithm Settings of MATE. “ M ” is the number of the customers.

	Description	Value
G_{max}	longest consecutive generations without improvement	50
ω_1, ω_2	lower and upper bounds for the proportion of the removed nodes	0.2, 0.4
N	population size	$\begin{cases} 16, & M \leq 100 \\ 36, & M > 100 \end{cases}$

this system, in addition to delivering the goods purchased by customers, one also needs to collect goods (e.g., defective goods or goods in need of maintenance) from customers, and both of these operations must be executed in predefined time windows to provide satisfactory service. Therefore in essence it could be modeled as the VRPSPDTW problem considered here. Further, original data was collected from this system, which contains requests occurring during a period of time in a city. The total number of requests is 3000. We then sampled from the data to generate instances with 200, 400, 600, 800, and 1000 customers. For each problem scale, we generated 4 instances, which finally gave us a benchmark set of 20 instances. The new benchmark set is called the *jd* set. Unlike the *wc* set, for this new set there is no priority for minimizing either NV or TD. Instead, u_1 and u_2 are directly given based on the estimated values in the application, and the objective is to minimize TC (total cost), i.e., the sum of the dispatching cost and the transportation cost, as defined in Eq. (4).

5.2. Compared Algorithms and Algorithm Settings

We considered four state-of-the-art algorithms¹ in the comparative study, including Co-GA [7], p-SA [11], VNS-BSTS [13], and ALNS-PR[14]. Each of these algorithms has been tested on part or all of the instances in the *wc* set, and for each instance in the *wc* set, at least one of the four algorithms has been reported to obtain the best-known solution. Hence, for the *wc* set, we directly obtained the best testing results of these algorithms from the original publications. For the new *jd* set, we chose Co-GA as the competitor since it is the only open-sourced algorithm among the four, and the parameter settings reported in the original publication were used.

¹The well-known TSP solver LKH (version 3.0) [50] is also applicable to VRPSPDTW. However, in our preliminary experiments, we found LKH generally took prohibitively long time (typically two days) to find a feasible solution to the problem instances considered. Therefore we did not consider it during our experiments.

Table 2: Comparison between CPLEX, Co-GA, and the proposed MATE on small-scale instances in the *wc* set. For each instance, the best performance regarding TC (i.e., $2000 \cdot NV + TD$) is highlighted in grey. “*M*” is the number of customers of the instance. “Time” refers to the rescaled computation time in seconds. “*a*” indicates the “out-of-memory” values. “Avg±std” is the average and the standard deviation of NV and TD of the 30 solutions found by the 30 independent runs. “Best” is the best solution regarding TC among the 30 solutions found by the 30 independent runs.

Instance <i>M</i>	CPLEX			Co-GA			MATE Avg±std		Best		
	NV	TD	Time	NV	TD	Time	NV	TD	NV	TD	Time
RCdp1001 10	3	348.98	1	3	348.98	1	3.00±0.00	348.98±0.00	3	348.98	1
RCdp1004 10	2	216.69	1387	2	216.69	1	2.00±0.00	216.69±0.00	2	216.69	1
RCdp1007 10	2	310.81	23	2	310.81	1	2.00±0.00	310.81±0.00	2	310.81	1
RCdp2501 10	5	551.05	15	5	551.05	3	5.00±0.00	551.05±0.00	5	551.05	1
RCdp2504 25	7 ^a	738.32 ^a	448193	4	473.46	2	4.00±0.00	473.46±0.00	4	473.46	1
RCdp2507 25	7 ^a	634.20 ^a	405429	5	540.87	3	5.00±0.00	540.87±0.00	5	540.87	1
RCdp5001 25	9	994.18	302146	9	994.18	17	9.00±0.00	994.18±0.00	9	994.18	1
RCdp5004 50	14 ^a	1961.53 ^a	774570	6	725.59	21	6.00±0.00	733.21±0.00	6	733.21	9
RCdp5007 50	13 ^a	1814.33 ^a	1427129	7	809.72	20	7.00±0.00	809.72±0.00	7	809.72	9

The detailed settings of MATE are summarized in Table 1. MATE has four parameters that need to be set, i.e., N , G_{max} , ω_1 and ω_2 , where N is the population size, G_{max} is the longest consecutive generations without improvement, and ω_1, ω_2 are the lower and upper bounds for the proportion of the nodes removed by the removal-and-reinsertion operator in the local search procedure (see Algorithm 4). We set G_{max} to 50 and set ω_1 and ω_2 to considerably large values, i.e., 0.2 and 0.4, respectively, such that the newly generated solution is likely to be very different from the original one. For parameter N , given limited time budgets, there exists a trade-off between promoting the exploration in larger search space (i.e., large N) and facilitating more refined search in local areas (i.e., small N). We consider the latter has direct and decisive impact on the algorithm’s performance, especially when the search space is too large to sufficiently explore. Therefore, for instances with $M \leq 100$, we set N to 16. Otherwise, we set N to 36 (note N must be a square number). The effect of different values of N will be further analyzed in Section 5.6.

All the experiments went through 30 independent runs, on an Intel Xeon E5-2699A v4 machine with 128 GB RAM and 22 cores (2.40 GHz, 55 MB Cache), running Centos 7.5. Both MATE and Co-GA were implemented in C++.

5.3. Results on Small-scale and Medium-scale Instances

The commercial mathematical programming software CPLEX has been used in [7] to find the optimal solutions for the small-scale instances ($M \leq 50$) in the *wc* set. The best results of CPLEX and Co-GA on these instances, obtained from [7], in comparison with the testing results of MATE, are presented in Table 2. Moreover, to make the comparison as fair as possible, we rescale the computation time of all approaches into a common time measure that takes into account the used CPUs. Specifically, we relate the Passmark scores of the CPUs used in the computational studies of the papers to the score of our E5-2699

V4. Each Passmark score refers to the performance of a single core of the respective CPU². The rescaled times in seconds are given in Table 2. Note that for the *wc* set, minimizing NV has a higher priority than minimizing TD; this is achieved by setting u_1 and u_2 in Eq. (4) to 2000 and 1, respectively. Therefore for these instances, the TC of a solution equals to $2000 \cdot NV + TD$, and a solution is better than another if the former has smaller TC. From Table 2 it can be observed that CPLEX managed to solve five of the instances, on which MATE and Co-GA also found optimal solutions. For the other four instances, CPLEX prematurely terminated due to the “out-of-memory” condition, while the two meta-heuristic algorithms found much better solutions than CPLEX, in a much shorter time. Moreover, MATE consumed less computation time than Co-GA, although they achieved very close solution quality. Overall, both of them performed much better than CPLEX, with the performance gap becoming even bigger as the problem scale increasing. It is worth mentioning that MATE performed very stably on these instances, achieving the standard deviation of 0 across 30 independent runs.

In general, the algorithms’ performance on instances of larger scales is of more interest. Table 3 presents the testing results of MATE and the best results of the four compared algorithms, obtained from the original publications, on the medium-scale instances ($M = 100$) in the *wc* set. The rescaled computation times of MATE and the best-performing algorithm among the competitors (i.e., ALNS-PR), are also presented. The efficacy of MATE can be evaluated from two perspectives, i.e., the best and the average performance it has achieved in 30 independent runs. We first take a closer look at the best performance since the compared algorithms’ results reported in Table 3 are their

²The Passmark score of the Core2 Quad 2.4G used in [7] is 957. For the i5-6600 used in [14], it is 2272. For our E5-2699 V4, it is 1037. See www.passmark.com for more details.

Table 4: Comparison between Co-GA and the proposed MATE on large-scale instances in the *jd* set. The total costs (TC) are reported. For each instance, the best performance is highlighted in grey; the average performance of an algorithm is indicated in bold if it is significantly better than the other algorithm based on 30 independent runs, according to the Wilcoxon rank-sum test with significance level $p = 0.05$.

Instance M	Co-GA			MATE			p-value
	Avg \pm std	Best	Time	Avg \pm std	Best	Time	
F201 200	67506 \pm 712	69301	675	66097 \pm 292	65106	133	0.0000
F202 200	66734 \pm 766	68513	2423	66038 \pm 422	65012	202	0.0002
F203 200	67651 \pm 713	69480	1046	67090 \pm 332	65980	240	0.0010
F204 200	66247 \pm 590	67557	773	65851 \pm 326	64747	180	0.0071
F401 400	127626 \pm 960	129258	7200	123261 \pm 446	122319	773	0.0000
F402 400	133307 \pm 1069	135032	7200	128091 \pm 410	126887	341	0.0000
F403 400	126660 \pm 1212	129191	7200	122306 \pm 682	120130	656	0.0000
F404 400	130807 \pm 993	133203	6026	125242 \pm 359	124517	1995	0.0000
F601 600	195250 \pm 1599	198688	7200	184119 \pm 608	182504	1161	0.0000
F602 600	202907 \pm 1235	205101	7200	188891 \pm 645	187236	1767	0.0000
F603 600	201579 \pm 1809	205827	7200	188050 \pm 621	186644	914	0.0000
F604 600	200232 \pm 1830	203499	7200	188110 \pm 790	186289	2465	0.0000
F801 800	235445 \pm 2870	241995	7200	214634 \pm 561	213661	1506	0.0000
F802 800	238457 \pm 1500	240912	7200	213276 \pm 292	212752	2560	0.0000
F803 800	236211 \pm 2136	241150	7200	214870 \pm 318	214126	2288	0.0000
F804 800	231765 \pm 1959	237389	7200	210845 \pm 429	209431	3497	0.0000
F1001 1000	N/A	N/A	N/A	314914 \pm 1096	312606	2014	N/A
F1002 1000	N/A	N/A	N/A	311718 \pm 1153	309158	5100	N/A
F1003 1000	N/A	N/A	N/A	313989 \pm 981	311377	4403	N/A
F1004 1000	N/A	N/A	N/A	311415 \pm 943	308816	3712	N/A

best performance. Overall, MATE is the best-performing algorithm in Table 3. From the row headed “No.best”, it can be found that MATE obtained the best solutions on 42 out of 56 instances, which is the most among all algorithms. More specifically, these instances are evenly distributed across all instance subsets, including 8 out of 12 in the Rdp1~ subset, 7 out of 9 in the Cdp1~ subset, 7 out of 8 in the RCdp1~ subset, 7 out of 11 in the Rdp2~ subset, 8 out of 8 in the Cdp2~ subset, and 5 out of 8 in the RCdp2~ subset. Considering the node distribution and constraint intensity are different across different instance types (see Section 5.1), such results indicate that MATE performs both strongly and robustly. In comparison, ALNS-PR, the second best-performing algorithm, performed not well on the RCdp1~ subset, finding best solutions for only 2 out of 8 instances in the set.

The row headed “w-d-l” in Table 3 presents the statistics of comparing the best performance of MATE with the best performance of other algorithms. Compared to Co-GA, p-SA and VNS-BSTS, MATE exhibits remarkable performance advantage. This is due to the fact that, unlike MATE, these algorithms have rather limited ability in exploring the search space since they neither construct high-diversity initial solutions nor adopt large-step-size operators to escape from local optima. As a result, MATE can usually identify more promising regions than others, and therefore have higher probability of finding high-quality solutions. Although the performance advantage of MATE over ALNS-PR is slight, the former still shows higher efficiency and consumes less computation time than the latter. This is due to the fact that MATE incorporate

the constant-time-complexity move evaluation approach, which can dramatically improve the search efficiency. In Section 5.5 we will further verify the effectiveness of all the novel components integrated in MATE. Finally, it is notable that MATE managed to find **new** best-known solutions on 12 instances in the *wc* set, especially considering this benchmark set has been widely used in the literature for more than 9 years.

It is worth mentioning on 24 out of 56 instances, MATE consistently found the best solution in every single run across 30 independent runs, and on almost half of the instances, i.e., 26 out of 56, the average performance of MATE across 30 independent runs is the same as its best performance. Comparing to the best performance of Co-GA, p-SA and VNS-BSTS, the average performance of MATE is still significantly better. Such results demonstrate that MATE performs very stably, although it is a randomized algorithm in nature. In conclusion, all of the above observations are evidence that MATE performs better than the state-of-the-art algorithms on a wide range of problem instances, and in particular, it has significantly raised the performance bar on the *wc* set.

5.4. Results on Large-scale Instances

Table 4 presents the testing results of Co-GA and MATE on large-scale instances in the *jd* set. To prevent the algorithms from running prohibitively long, we limit the maximum computation time to 7200 seconds. Unlike the *wc* set, for this set there is no priority for minimizing either NV or TD; thus the TC of the solutions are reported. In the experiments, Co-GA kept crashing on instances with

Table 5: Average PDR for the five MATE variants on each subset of the *wc* set. For each subset, the highest average PDR is highlighted in grey. “Avg.PDR” refers to the average PDR on all instances in the *wc* set.

Subset	w/o ED	w/o RI	w/o LS	w/o ES	w/o FL
<i>wc_{small}</i>	0.00%	0.00%	0.01%	0.00%	0.00%
Rdp1~	0.21%	0.18%	1.17%	0.35%	0.78%
Cdp1~	1.24%	1.07%	0.55%	0.76%	0.12%
RCdp1~	0.80%	0.21%	1.53%	1.41%	0.46%
Rdp2~	1.14%	0.94%	1.77%	0.64%	0.66%
Cdp2~	0.00%	0.00%	0.37%	0.01%	0.09%
RCdp2~	0.03%	0.43%	2.14%	0.17%	0.40%
Avg.PDR	0.51%	0.42%	1.06%	0.47%	0.38%

1000 customers; for these instances “N/A” is reported. Moreover, nonparametric tests (Wilcoxon rank-sum test with significance level $p = 0.05$) were conducted to compare the performance of Co-GA and MATE, with p-value shown in the last column. It can be seen that MATE obtained significantly better solutions than Co-GA on every instance in the *jd* set, and also consumed much less computation time than Co-GA. The gap between the best performance of Co-GA and MATE across 30 independent runs, i.e., $[TC_1 - TC_2]/TC_2$ where TC_1 and TC_2 are the best performance of Co-GA and MATE respectively, is 5.37%, 6.65%, 9.48% and 13.12%, averaged on 200-customer, 400-customer, 600-customer and 800-customer instances, respectively. For average performance, the corresponding gaps are 1.16%, 3.90%, 6.78%, and 10.34%. It can be observed as the problem scale grows, the performance gap between Co-GA and MATE also becomes larger.

5.5. Effectiveness of Each Component in MATE

An ablation study was conducted to further assess the effectiveness of the novel components integrated into MATE. More specifically, we tested the following 5 different variants of MATE on the *wc* set, each of which is different from MATE in one component:

1. w/o ED: In initialization, it uses random values drawn from $[0,1]$ for (λ, γ) , instead of the proposed evenly-distributed values (lines 4-5 of Algorithm 2);
2. w/o RI: After doing route inheritance during crossover, it adopts the two-step insertion as described in [46] (see Section 4.3) to reinsert the unassigned customers, instead of the proposed regret-based insertion (lines 7-15 of Algorithm 3);
3. w/o LS: It has no local search procedure, i.e., line 8 in Algorithm 1 is removed;
4. w/o ES: In the local search procedure, it does not seek to escape from local optimum with the removal-and-reinsertion operator, i.e., lines 2-12 are removed from Algorithm 4;
5. w/o FL: In the local search procedure, it does not seek to identify local optimum, i.e., line 1 and line 9 in Algorithm 4 are removed.

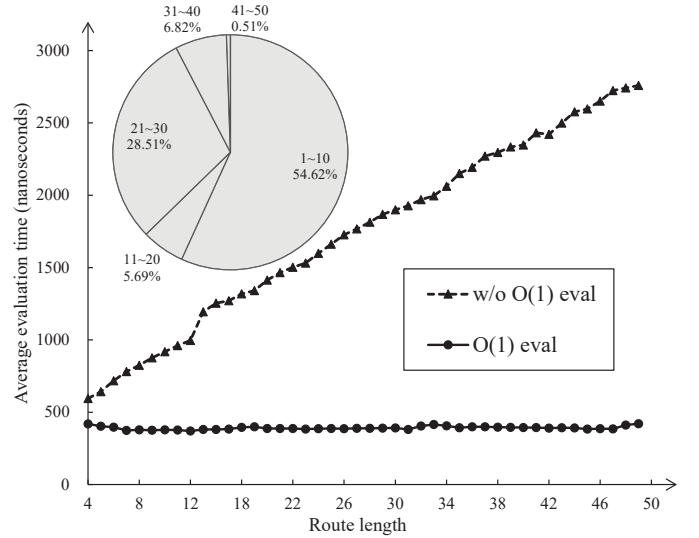


Figure 4: Average evaluation time (ns) at different route lengths when turning on/off the new evaluation approach. The proportions of different route lengths are also illustrated.

For each of the above variant, the performance degradation ratio (PDR) is computed as $[TC_1 - TC_2]/TC_2$ for each instance, where TC_1 and TC_2 are the average TC obtained by the variant and MATE across 30 independent runs. Therefore, the component being examined is useful for this specific instance only if the PDR > 0 , and the larger it is, the greater the component contributes to the performance of MATE. Table 5 presents the PDR averaged on each instance subset of the *wc* set, where *wc_{small}* refers to all the small-scale instances. It can be observed that, on average, none of the examined components has negative effect on the performance of MATE, and in most cases, they make MATE perform better. In particular, the results in the “w/o ED” column indicate that using evenly-distributed values for (λ, γ) is better than using random values, since the former enables more sufficient exploration in the design space of (λ, γ) . The results in the “w/o RI” column indicate that the regret-based insertion performs better than the two-step insertion, which is appealing considering the former is even simpler. As we expect, the results in the “w/o LS” column demonstrate that the local search procedure has the most significant contribution to the performance of MATE, and it is also the only one achieving positive effect on all subsets. The results in the last two columns (“w/o ES” and “w/o FL”) indicate that removing either small-neighborhood operators (i.e., identifying local optimum) or large-neighborhood operator (i.e., escape from local optimum) from the local search procedure will result in performance degradation, which on the other hand indicate that the strategy of switching between operators of different step sizes is effective in obtaining high-quality solutions.

One may notice that the new move evaluation approach was left out from the above analysis. Since it does not affect any algorithmic behavior of MATE, but serves as an accelerator, in the experiments we kept track of the average

Table 6: Average PDR for 7 different N values on different subsets. For each subset, the lowest average PDR is highlighted in grey. “ M ” is the number of customers of each instance in the subset.

Subset M	$N=4$	$N=9$	$N=16$	$N=25$	$N=36$	$N=49$	$N=64$
$wc_{small} 10,25,50$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Rdp1~ 100	0.67%	0.48%	0.90%	0.67%	0.09%	1.16%	1.27%
Cdp1~ 100	0.79%	1.37%	1.17%	1.38%	0.42%	0.00%	1.38%
Rdp2~ 100	1.19%	1.21%	1.17%	1.78%	0.28%	1.79%	0.00%
Cdp2~ 100	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
RCdp1~ 100	1.01%	1.40%	0.92%	2.13%	0.95%	0.09%	1.49%
RCdp2~ 100	0.09%	0.09%	0.08%	0.10%	0.00%	0.11%	0.09%
F20~ 200	0.00%	0.56%	0.77%	1.88%	2.11%	2.48%	2.22%
F40~ 400	0.42%	0.29%	0.21%	0.68%	0.64%	0.57%	0.63%

computation time spent on move evaluation at different route lengths, with the new approach turned on/off. Note that when it was turned off, the traditional approach which would traverse the involved routes was used. The results are presented in Figure 4, where “ $O(1)$ eval” refers to the new evaluation approach. In addition, the proportions of different lengths of routes encountered through the whole experiments are also shown. It can be clearly seen that the move evaluation time for the traditional approach grows linearly with route length, while for the new approach, the time is almost constant. These results have verified that the new approach indeed has time complexity of $O(1)$, and the longer the route is, the more time it saves. On instances in the wc set, the route lengths that occur most frequently are 1~10 and 21~30, on which the new approach can achieve nearly $1.5\times$ and $3.5\times$ speedups over the traditional approach, respectively. Averaged on all instances in the wc set, the new evaluation approach offers nearly $3.3\times$ speedup over the traditional approach.

5.6. Sensitivity Analysis of the Population Size N

The population size N is a user-defined parameter that has an important impact on the algorithm’s performance. A larger N directly enables more adequate coverage on the design space of (λ, γ) in the initialization procedure (see Section 4.2), leading to higher diversity among the initial population and therefore better exploration. On the other hand, a smaller N facilitates MATE to search more sufficiently in local areas, leading to better exploitation. Given a limited time budget, the value of N determines the trade-off between exploration and exploitation. To quantitatively investigate its impact, we tested MATE with different N , i.e., 4, 9, 16, 25, 36, 49 and 64, on instances of different scales. More specifically, we considered all the seven subsets of the wc set, and two additional subsets F20~ (containing four 200-customer instances) and F40~ (containing four 400-customer instances) of the jd set. For instances in the wc set, we set the maximum running time of MATE to 60s; while for instances in F20~ and F40~, we set the maximum running time to 600s. On each instance, MATE with each considered N , denoted as N_i , was tested for 30 independent runs, and the average TC,

denoted as TC_i , was computed. Then as before, for N_i the PDR against the best-performing N was computed, i.e., $[TC_i - TC_{best}]/TC_{best}$, where TC_{best} is the best performance achieved among all the considered N . Therefore a PDR of 0.00% means the corresponding N achieves the best performance on the instance. For each considered N , its average PDR on each subset is presented in Table 6. Note the larger the average PDR is, the worse the corresponding N performs on the subset, and an average PDR of 0.00% means the corresponding N performs the best on every instance in the subset.

One can make two important observations from Table 6. First, on the small-scale instances (the wc_{small} subset), the performance of MATE is completely insensitive to the value of N . Second, for instances of medium scales ($M = 100$), larger values of N (36, 49 and 64) enabling better exploration perform better. In particular, each of the three values has achieved the best average PDR on two such subsets, besides Cdp2~. On the other hand, as problem scale increases, the search space grows exponentially. In this case more sufficient search in local areas is becoming more important under limited time budgets; therefore a smaller value of N (4, 9 and 16) is better. In summary, the above results show that the parameter N is indeed important for MATE, and setting N to 36 in case of $M \leq 100$, and 16 in case of $M > 100$, is a reasonably good default choice if no other prior knowledge is suggested.

6. Conclusion

In this paper, we proposed a memetic algorithm, dubbed MATE, for solving VRPSPDTW. Compared to existing algorithms, MATE is novel in three aspects: initialization procedure, crossover operator and local search procedure. Based upon our comprehensive experimental studies, two main conclusions can be drawn. First of all, MATE is capable of finding better solutions than the state-of-the-art algorithms on a wide range of problem instances. Notably, MATE finds new best-known solutions on 12 instances in the existing benchmark (65 instances in total). Second, each novel component integrated into MATE has contributed to its overall performance, and the local search

procedure is the one with the biggest contribution. Moreover, a set of 20 new instances with 200, 400, 600, 800 and 1000 customers were derived from a real-world application of the JD logistics and utilized as a new benchmark set for the large-scale VRPSPDTW.

MATE can be further improved in two aspects. Currently the maximum length of the subsequences manipulated by the local-search operators is set to 2, which may affect the performance on optimizing lengthy routes. On the other hand, using a larger maximum length will introduce higher computational costs. An option is to enhance MATE with self-adaptation that dynamically adjusts the maximum length according to the routes being manipulated. Another possible improvement to MATE is a more refined scheme of conducting local search. It has been well recognized in the literature that not all individuals in the population deserve local search. Hence, mechanisms (e.g., heuristics or learned models) can be integrated into MATE to exclude those “unpromising” individuals from the local search procedure to further reduce the computational costs.

Another future direction is to gradually build MATE into a highly parameterized algorithm framework that supports solving more VRP variants, e.g., multi-depot VRP and VRP with electric fleet. In this way, one who needs to solve a specific type of VRP can utilize automation tools [51–55] to search in the configuration space built upon MATE, to obtain an effective algorithm for the specific problem of interest.

References

- [1] H. Min, The multiple vehicle routing problem with simultaneous delivery and pick-up points, *Transportation Research Part A: General* 23 (5) (1989) 377–386.
- [2] J. Dethloff, Vehicle routing and reverse logistics: The vehicle routing problem with simultaneous delivery and pick-up, *OR Spectrum* 23 (1) (2001) 79–96.
- [3] G. Berbeglia, J. Cordeau, G. Laporte, Dynamic pickup and delivery problems, *European Journal of Operational Research* 202 (1) (2010) 8–15.
- [4] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, G. Laporte, Static pickup and delivery problems: a classification scheme and survey, *Top* 15 (1) (2007) 1–31.
- [5] M. Battarra, J. Cordeau, M. Iori, Pickup-and-delivery problems for goods transportation, in: P. Toth, D. Vigo (Eds.), *Vehicle Routing*, Vol. 18 of MOS-SIAM Series on Optimization, SIAM, 2014, pp. 161–191.
- [6] Ç. Koç, G. Laporte, İ. Tükenmez, A review of vehicle routing with simultaneous pickup and delivery, *Computers & Operations Research* 122 (2020) 104987.
- [7] H. Wang, Y. Chen, A genetic algorithm for the simultaneous delivery and pickup problems with time window, *Computers & Industrial Engineering* 62 (1) (2012) 84–95.
- [8] E. Angelelli, R. Mansini, The vehicle routing problem with time windows and simultaneous pick-up and delivery, in: A. Klöse, M. G. Speranza, L. N. V. Wassenhove (Eds.), *Quantitative Approaches to Distribution Logistics and Supply Chain Management*, Vol. 519 of Lecture Notes in Economics and Mathematical Systems, Springer, 2002, pp. 249–267.
- [9] L. Mingyong, C. Erbao, An improved differential evolution algorithm for vehicle routing problem with simultaneous pickups and deliveries and time windows, *Engineering Applications of Artificial Intelligence* 23 (2) (2010) 188–195.
- [10] S. Kassem, M. Chen, Solving reverse logistics vehicle routing problems with time windows, *The International Journal of Advanced Manufacturing Technology* 68 (1-4) (2013) 57–68.
- [11] C. Wang, D. Mu, F. Zhao, J. W. Sutherland, A parallel simulated annealing method for the vehicle routing problem with simultaneous pickup-delivery and time windows, *Computers & Industrial Engineering* 83 (2015) 111–122.
- [12] W. Huang, T. Zhang, Vehicle routing problem with simultaneous pick-up and delivery and time-windows based on improved global artificial fish swarm algorithm, *Computer Engineering and Applications* 52 (21) (2016) 21–29.
- [13] Y. Shi, Y. Zhou, T. Boudouh, O. Grunder, A lexicographic-based two-stage algorithm for vehicle routing problem with simultaneous pickup-delivery and time window, *Engineering Applications of Artificial Intelligence* 95 (2020) 103901.
- [14] J. Hof, M. Schneider, An adaptive large neighborhood search with path relinking for a class of vehicle-routing problems with simultaneous pickup and delivery, *Networks* 74 (3) (2019) 207–250.
- [15] K. Tang, J. Wang, X. Li, X. Yao, A scalable approach to capacitated arc routing problems based on hierarchical decomposition, *IEEE Transactions on Cybernetics* 47 (11) (2017) 3928–3940.
- [16] J. Deng, L. Wang, A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem, *Swarm and Evolutionary Computation* 32 (2017) 121–131.
- [17] Y. Du, L. Xing, J. Zhang, Y. Chen, Y. He, MOEA based memetic algorithms for multi-objective satellite range scheduling problem, *Swarm and Evolutionary Computation* 50 (2019) 100576.
- [18] E. Osaba, X. Yang, I. F. Jr., J. D. Ser, P. López-García, A. J. Vazquez-Pardavila, A discrete and improved bat algorithm for solving a medical goods distribution problem with pharmacological waste collection, *Swarm and Evolutionary Computation* 44 (2019) 273–286.
- [19] D. Trachanatzi, M. Rigakis, M. Marinaki, Y. Marinakis, A firefly algorithm for the environmental prize-collecting vehicle routing problem, *Swarm and Evolutionary Computation* 57 (2020) 100712.
- [20] K. Tang, Y. Mei, X. Yao, Memetic algorithm with extended neighborhood search for capacitated arc routing problems, *IEEE Transactions on Evolutionary Computation* 13 (5) (2009) 1151–1166.
- [21] Y. Mei, K. Tang, X. Yao, A memetic algorithm for periodic capacitated arc routing problem, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 41 (6) (2011) 1654–1667.
- [22] Y. Zhang, Y. Mei, K. Tang, K. Jiang, Memetic algorithm with route decomposing for periodic capacitated arc routing problem, *Applied Soft Computing* 52 (2017) 1130–1142.
- [23] J. Decerle, O. Grunder, A. H. E. Hassani, O. Barakat, A hybrid memetic-ant colony optimization algorithm for the home health care problem with time window, synchronization and working time balancing, *Swarm and Evolutionary Computation* 46 (2019) 171–183.
- [24] J. Decerle, O. Grunder, A. H. E. Hassani, O. Barakat, A memetic algorithm for multi-objective optimization of the home health care problem, *Swarm and Evolutionary Computation* 44 (2019) 712–727.
- [25] M. Chen, J. Chen, P. Yang, S. Liu, K. Tang, A heuristic repair method for dial-a-ride problem in intracity logistic based on neighborhood shrinking, *Multimedia Tools and Applications* (2020) 1–13.
- [26] N. R. Sabar, A. Bhaskar, E. Chung, A. Turkey, A. Song, A self-adaptive evolutionary algorithm for dynamic vehicle routing problems with traffic congestion, *Swarm and Evolutionary Computation* 44 (2019) 1018–1027.
- [27] N. R. Sabar, A. Bhaskar, E. Chung, A. Turkey, A. Song, An adaptive memetic approach for heterogeneous vehicle routing problems with two-dimensional loading constraints, *Swarm and Evolutionary Computation* 58 (2020) 100730.
- [28] M. Okulewicz, J. Mandziuk, A metaheuristic approach to solve dynamic vehicle routing problem in continuous search space,

- Swarm and Evolutionary Computation 48 (2019) 44–61.
- [29] Y. Wang, L. Wang, G. Chen, Z. Cai, Y. Zhou, L. Xing, An improved ant colony optimization algorithm to the periodic vehicle routing problem with time window and service choice, *Swarm and Evolutionary Computation* 55 (2020) 100675.
- [30] S. S. Choong, L. Wong, C. P. Lim, An artificial bee colony algorithm with a modified choice function for the traveling salesman problem, *Swarm and Evolutionary Computation* 44 (2019) 622–635.
- [31] R. Goscién, Two metaheuristics for routing and spectrum allocation in cloud-ready survivable elastic optical networks, *Swarm and Evolutionary Computation* 44 (2019) 388–403.
- [32] M. Dell’Amico, G. Righini, M. Salani, A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection, *Transportation Science* 40 (2) (2006) 235–247.
- [33] J. Rieck, J. Zimmermann, Exact solutions to the symmetric and asymmetric vehicle routing problem with simultaneous delivery and pick-up, *Business Research* 6 (1) (2013) 77–92.
- [34] A. Subramanian, E. Uchoa, A. A. Pessoa, L. S. Ochi, Branch-and-cut with lazy separation for the vehicle routing problem with simultaneous pickup and delivery, *Operations Research Letters* 39 (5) (2011) 338–341.
- [35] A. Subramanian, E. Uchoa, A. A. Pessoa, L. S. Ochi, Branch-cut-and-price for the vehicle routing problem with simultaneous pickup and delivery, *Optimization Letters* 7 (7) (2013) 1569–1581.
- [36] F. A. T. Montané, R. D. Galvão, Vehicle routing problems with simultaneous pick-up and delivery service, *Opsearch* 39 (1) (2002) 19–33.
- [37] Y. Gajpal, P. L. Abad, Saving-based algorithms for vehicle routing problem with simultaneous pickup and delivery, *Journal of Operational Research Society* 61 (10) (2010) 1498–1509.
- [38] J. Chen, T. Wu, Vehicle routing problem with simultaneous deliveries and pickups, *Journal of Operational Research Society* 57 (5) (2006) 579–587.
- [39] N. A. Wassan, A. H. Wassan, G. Nagy, A reactive tabu search algorithm for the vehicle routing problem with simultaneous pickups and deliveries, *Journal of Combinatorial Optimization* 15 (4) (2008) 368–386.
- [40] A. Subramanian, L. M. d. A. Drummond, C. Bentes, L. S. Ochi, R. Farias, A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery, *Computers & Operations Research* 37 (11) (2010) 1899–1911.
- [41] M. Avci, S. Topaloglu, An adaptive local search algorithm for vehicle routing problem with simultaneous and mixed pickups and deliveries, *Computers & Industrial Engineering* 83 (2015) 15–29.
- [42] Y. Gajpal, P. Abad, An ant colony system (acs) for vehicle routing problem with simultaneous delivery and pickup, *Computers & Operations Research* 36 (12) (2009) 3215–3223.
- [43] J. Ai, V. Kachitvichyanukul, A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery, *Computers & Operations Research* 36 (5) (2009) 1693–1702.
- [44] T. Vidal, T. G. Crainic, M. Gendreau, C. Prins, A unified solution framework for multi-attribute vehicle routing problems, *European Journal of Operational Research* 234 (3) (2014) 658–673.
- [45] M. M. Solomon, Algorithms for the vehicle routing and scheduling problems with time window constraints, *Operations Research* 35 (2) (1987) 254–265.
- [46] G. B. Alvarenga, G. R. Mateus, G. F. C. de Tomi, A genetic and set partitioning two-phase approach for the vehicle routing problem with time windows, *Computers & Operations Research* 34 (6) (2007) 1561–1584.
- [47] O. Bräysy, M. Gendreau, Vehicle routing problem with time windows, Part i: Route construction and local search algorithms, *Transportation Science* 39 (1) (2005) 104–118.
- [48] X. Yao, Simulated annealing with extended neighbourhood, *International Journal of Computer Mathematics* 40 (3-4) (1991) 169–189.
- [49] X. Yao, Dynamic neighbourhood size in simulated annealing, in: *Proceedings of the 2nd International Joint Conference on Neural Networks (IJCNN’92)*, Baltimore, MD, 1992, pp. 411–416.
- [50] K. Helsgaun, An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems, Tech. rep., Technical Report, Roskilde University, Roskilde, Denmark (2017).
- [51] S. Liu, K. Tang, Y. Lei, X. Yao, On performance estimation in automatic algorithm configuration, in: *Proceedings of the 34th AAAI Conference on Artificial Intelligence, AAAI’ 2020*, New York, NY, 2020, pp. 2384–2391.
- [52] S. Liu, K. Tang, X. Yao, Automatic construction of parallel portfolios via explicit instance grouping, in: *Proceedings of the 33rd AAAI Conference on Artificial Intelligence, AAAI’ 2019*, Honolulu, HI, 2019, pp. 1560–1567.
- [53] S. Liu, K. Tang, X. Yao, Generative adversarial construction of parallel portfolios, *IEEE Transactions on Cybernetics*, in press, DOI:10.1109/TCYB.2020.2984546 (2020).
- [54] K. Tang, S. Liu, P. Yang, X. Yao, Few-Shots Parallel Algorithm Portfolio Construction via Co-Evolution, *IEEE Transactions on Evolutionary Computation* 25 (3) (2021) 595–607.
- [55] W. Chen, S. Liu, K. Tang, A new knowledge gradient-based method for constrained bayesian optimization, arXiv preprint arXiv:2101.08743.