A Hybrid Differential Evolution Algorithm for the Online Meal Delivery Problem

Jing-fang Chen[1], Shengyao Wang[2], Ling Wang[1], Jie Zheng[1],
Ying Cha[2], Jinghua Hao[2], Renqing He[2], Zhizhao Sun[2]
[1]*Department of Automation*, *Tsinghua University*, Beijing, China
[2]*Meituan-Dianping Group*, Beijing, China
cjf17@mails.tsinghua.edu.cn, wangshengyao@meituan.com,
wangling@mail.tsinghua.edu.cn, j-zheng18@mails.tsinghua.edu.cn,
{chaying, haojinghua, herenqing, sunzhizhao}@meituan.com

***Abstract*—In recent years, the online food ordering (OFO) platforms have arose fast and brought huge convenience to people in daily life. Under the scenario of a realistic OFO platform, this paper addresses an online meal delivery problem (OMDP). To reduce the search space, the OMDP is decomposed into two sub-problems, i.e., the pickup and delivery problem and the order dispatching problem. To solve each sub-problem effectively, a hybrid differential evolution algorithm is proposed, which is fused by the DE-based phase to plan routes and the heuristic-based phase to determine order dispatching schemes. In the DE-based routing phase, a heuristic considering the urgency of orders is designed to generate the initial population with certain quality. Besides, a mutation operator is developed to enhance the exploration and a crossover operator embedded with local search is designed to enhance the exploitation. In the heuristic-based dispatching phase, a regret heuristic is presented to produce good dispatching solutions by introducing the influences between delivery persons. Numerical tests have been carried out and computational results demonstrate the effectiveness of the proposed algorithm.**

***Index Terms*—online meal delivery problem, pickup and delivery problem, differential evolution, regret heuristic**

I. Introduction

A. Backgroud

On online food delivery platforms, diners are allowed to order meals from a wide range of restaurants only with a tap of smart phone. Take Meituan-Dianping, a Chinese food ordering platform, for example, the overall procedure of delivering the online meal orders is illustrated in Fig. 1. After the customers make orders, the Meituan platform will push the orders to the restaurants and dispatch the orders to delivery persons, also called as riders. Then, the riders will pick up meals from the corresponding restaurants after the meals have been prepared and deliver them to the customers. Such convenience attracts more and more customers and promotes the prosperity of meal delivery operations. According to the revenue statement [1] of Meituan-Dianping, for the third quarter of 2019, the number of food delivery orders increased by 38.1% to 2.5 billion from 1.8 billion in the same period of 2018, and the Gross Transaction Volume increased by 40.0% to RMB111.9 billion from RMB80.0 billion in the same period of 2018. However, the huge amount and ongoing growth of orders also pose a challenge for the online food delivery platforms in last mile logistics. In Meituan-Dianping, online orders are received almost every minute throughout the country so that the platform needs to dispose the orders in a very short time. Besides, it also should be considered that how to reduce the traveling distance and serve more orders with a limited number of riders to increase delivery efficiency. More importantly, orders must be delivered on time to earn customer satisfaction in competitive market. Therefore, the intrinsic nature of the online meal delivery problem (OMDP) makes it a difficult and primary issue for the online food delivery platforms.

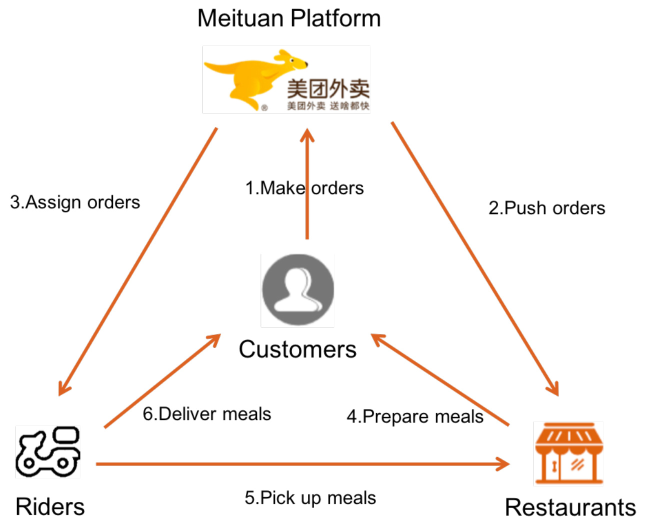


Fig. 1: The procedure of delivering meals.

B. Literature Review

In the literature, it lacks of relative research works studying the above scenario, and the research topic that most relates to the OMDP is the pickup and delivery problem (PDP) or the vehicle routing problem (VRP).

The PDP is a common problem in many fields, such as logistics, ambulatory services, and robotics [2]. The typical characteristic of PDP is to collect objects or people from origin depots and transport them to the destinations. As a generalization of the VRP, the PDP has attracted the interest

TABLE I: Notations

Notation	Description
T	The current time to dispatch new orders.
i, j, k	The index of orders, riders and route nodes respectively.
m	The number of riders.
Q	The set of riders $Q = \{q_1, q_2, ..., q_m\}$.
n^{new}	The number of new orders.
O^{new}	The set of new orders. $O^{new} = \{o^{new}(1), o^{new}(2), \ldots, o^{new}(n^{new})\}$.
n_j	The number of old orders that have been dispatched to rider q_j.
O_j^{old}	The set of old orders dispatched to rider q_j. $O_j^{old} = \{o_j^{old}(1), o_j^{old}(2), \ldots, o_j^{old}(n_j)\}$.
n^{old}	The number of old orders. $n^{old} = n_1 + n_2 + ... + n_m$.
O^{old}	The set of old orders. $O^{old} = \{o^{old}(1), o^{old}(2), \ldots, o^{old}(n^{old})\}$.
n	The number of all orders to be served. $n = n^{new} + n^{old}$
O	The set of all orders. $O = \{o(1), o(2), \ldots, o(n)\} = O^{new} \cup O^{old}$
i^+	The pickup node of order $o(i)$.
i^-	The delivery node of order $o(i)$.
P^{old}	The pickup nodes set of old orders. $P^{old} = \{i^+ \| o(i) \in O^{old}\}$.
P^{new}	The pickup nodes set of new orders. $P^{new} = \{i^+ \| o(i) \in O^{new}\}$.
D^{old}	The delivery nodes set of old orders. $D^{old} = \{i^- \| o(i) \in O^{old}\}$.
D^{new}	The delivery nodes set of new orders. $D^{new} = \{i^- \| o(i) \in O^{new}\}$.
P	The set of all pickup nodes. $P = P^{old} \cup P^{new}$.
D	The set of all delivery nodes. $D = D^{old} \cup D^{new}$.
L	The site nodes set of riders. $L = \{l_1, l_2, ..., l_m\}$.
$R_{0,j}$	The old route of rider q_j. $R_{0,j} = \{r_{0,j}(0), r_{0,j}(1), r_{0,j}(2), \ldots, r_{0,j}(n_{0,j})\}$.
$R_{i,j}$	The new route of rider q_j after dispatching $o(i)$ to q_j. $R_{i,j} = \{r_{i,j}(0), r_{i,j}(1), r_{i,j}(2), \ldots, r_{i,j}(n_{i,j})\}$, $i > 0$.
DT_i	The due time of $o(i)$.
$t(i_1, i_2)$	The travel time between node i_1 and node i_2.
$d_(i_1, i_2)$	The travel distance between node i_1 and node i_2.
PT_i	Preparation time, i.e., the time when the meal of $o(i)$ is prepared by the restaurant.
$TT_{i,j}$	The total tardiness of route $R_{i,j}$.
$TD_{i,j}$	The travel distance of route $R_{i,j}$.
$RC_{i,j}$	The route cost of route $R_{i,j}$.
$C_{i,j}$	The cost of dispatching $o(i)$ to q_j.
CM	The cost matrix. $CM = (C_{i,j})_{n^{new} \times m}$.
$X_{i,j}$	A binary variable for dispatching. If $o(i)$ is dispatched to q_j, then $X_{i,j} = 1$;otherwise, $X_{i,j} = 0$.
TC	The total cost of dispatching all new orders.

of numerous researchers. Consequently, various algorithms have been developed in the past years. Ruland and Rodin [3] formulated the PDP into an integer program and presented a branch-and-cut algorithm by exploring its polyhedral structure. Dumas et al. [4] developed a dynamic programming algorithm with a column generation scheme to solve the PDP with time windows (PDPTW). Ropke and Cordeau [5] proposed a branch-and-cut-and-price algorithm by solving two sub-problems in column generation to obtain lower bounds. Besides the exact algorithms, a number of effective heuristics and metaheuristics have been presented as well. To solve the multi-vehicle PDPTW, Lu and Dessouky [6] proposed an insertion-based construction heuristic considering the classical incremental distance measure and the cost of reducing the time window slack. Şahin et al. [7] combined the mechanism of tabu search and simulated annealing to deal with the multi-vehicle PDP with split loads. Pankratz [8] proposed a grouping genetic algorithm with a group-oriented genetic encoding for solving the PDPTW. Ai and Kachitvichyanukul [9] developed a particle swarm optimization algorithm with random key-based encoding and decoding method for VRP with simultaneous pickup and delivery (VRPSPD). Besides, other population-based evolutionary algorithms, such as memetic algorithm [10], ant colony algorithm [11], particle swarm optimization [12] and so on, have also been developed for solving the PDP and its variant problems.

Differential evolution (DE) is first proposed by Storn and Price [13] in 1995 to solve the complex continuous nonlinear problems. Due to its simplicity, ease of implementation, fast convergence and robustness, DE has been widely applied to various fields [14] including the PDP. Berhan et al. [15] utilized DE to deal with the stochastic VRPSPD for a bus service enterprise. Teoh et al. [16] proposed a DE algorithm with local search to further modify the solutions for the capacitated VRP. Dechampai et al. [17] embedded DE into a two-phase heuristic to solve a complicated variant of the capacitated VRP with pickup and delivery services in poultry industry. Lai and Cao [18] improved DE with different coding methods for VRPSPD. They utilized a novel decimal coding in initialization, employed an integer order criterion in mutation and designed a self-adapting crossover probability. The good performance and successful applications make DE a popular and promising evolutionary algorithm to solve complicated combinatorial optimization problems in real life.

This paper proposes a hybrid DE (HDE) for solving the OMDP. To find good solutions in a very limited time, the search space of OMDP is reduced by decomposing it into two sub-problems, which are PDP and order dispatching problem. Accordingly, the HDE is composed of the DE-based routing phase and the heuristic-based dispatching phase to deal with the two sub-problems respectively. In the DE-based routing phase, a heuristic based on the urgency of the orders is designed to ensure the quality of the initial population. Besides, a discrete mutation operator is presented to enhance the exploration capability. Furthermore, we develop a discrete crossover operator embedded with insertion-based local search to improve the exploitation capability. In the heuristic-based phase, the orders are dispatched in the light of their regret values, which can reflect the long-term cost of dispatching each order. The two phases of HDE collaborate to produce good dispatching schemes with minimum cost and maximum efficiency.

The remainder of this paper is structured as follows. Section II gives the notations and describes the OMDP. Section III introduces the design of the proposed HDE. Computational results and analysis are provided in Section IV. Section V ends this paper with conclusions and future work.

II. Problem Description

With the notations listed in Table I, the OMDP can be described as follows. At time *T*, there are n^{new} new online orders $O^{new} = \{o^{new}(1), o^{new}(2), \ldots, o^{new}(n^{new})\}$ to be dispatched to *m* riders $Q = \{q_1, q_2, ..., q_m\}$. Each rider q_j possesses n_j old orders $O_j^{old} = \{o_j^{old}(1), o_j^{old}(2), \ldots, o_j^{old}(n_j)\} \subseteq O^{old} = \{o^{old}(1), o^{old}(2), \ldots, o^{old}(n^{old})\}$, which have been dispatched previously. Each order $o(i)$ has a due time DT_i that is either appointed by the customers or committed to the customers by the platform. Besides, each order $o(i)$ is associated with a single node i^- or a pair of nodes (i^+, i^-), where i^+ is the pickup node and i^- is the delivery node. To be specific, some of the old orders will not be associated with a pickup node if the meals have been picked up before the current time, while the rest old orders and all new orders contain both pickup node and delivery node. Therefore, it holds that $|P^{old}| <= |D^{old}|$ and $|P^{new}| = |D^{new}|$, where $P^{old} = \{i^+ | o(i) \in O^{old}\}$ and $P^{new} = \{i^+ | o(i) \in O^{new}\}$ is the set of all pickup nodes of old orders and new orders respectively, and $D^{old} = \{i^- | o(i) \in O^{new}\}$ and $D^{new} = \{i^- | o(i) \in O^{new}\}$ is the set of all delivery nodes of old orders and new orders respectively. Let $L = \{l_1, l_2, ..., l_m\}$ be the set of the site nodes that riders locate in at time *T*. Then the OMDP can be defined on a graph $G = (N, A)$, where $N = P^{old} \cup P^{new} \cup D^{old} \cup D^{new} \cup L$ is the set of nodes and A is the set of arcs. Each arc $(i_1, i_2) \in A \quad (i_1, i_2 \in N, i_1 \neq i_2)$ is related to a travel time $t(i_1, i_2)$ and a travel distance $d(i_1, i_2)$. Each rider has an old route $R_{0,j} = \{r_{0,j}(0), r_{0,j}(1), r_{0,j}(2), \ldots, r_{0,j}(n_{0,j})\}$, where $r_{0,j}(0)$ is the site node l_j, $r_{0,j}(k)$ $(k = 1, \ldots, n_{0,j}, r_{0,j}(k) \in P^{old} \cup D^{old})$ is the kth route node and $n_{0,j}$ is the number of old route nodes. If order $o(i) \in O^{new}$ is assigned to rider q_j, a new route $R_{i,j} = \{r_{i,j}(0), r_{i,j}(1), r_{i,j}(2), \ldots, r_{i,j}(n_{i,j})\} (i > 1)$ will be scheduled for q_j, where $r_{i,j}(0)$ is the site node l_j, $r_{i,j}(k)$ $(k = 1, \ldots, n_{i,j}; r_{i,j}(k) \in P^{old} \cup P^{new} \cup D^{old} \cup D^{new})$ is the kth route node of the new route and $n_{i,j}$ is the number of new route nodes. There is a capacity Q which is the maximum weight of meals that can be carried by each rider. The basic constraints are as follows.

- For each order, the meal must be picked up before being delivered if it has a pickup node. (precedence constraint)
- Meals cannot be picked up before it has been prepared by the restaurant.
- A new order can only be dispatched to one rider.
- Old orders cannot be reassigned to other riders.
- The total weight of orders that a rider carries cannot exceed Q. (capacity constraint)

In OMDP, custom satisfaction and delivery efficiency are both considered to construct the objective function.

To measure the custom satisfaction, one of the most direct ways is to calculate the tardiness of each order. The tardiness $TA_{i,j}(i'^-)$ of order $o(i')$ in route $R_{i,j}$ $(i = 0, 1, ..., n^{new}, j = 1, 2, ..., m)$ can be computed as follows.

$$WT_{i,j}(k) = max\{0, PT_{i'} - AT_{i,j}(k)\}, r_{i,j}(k) \in P \tag{1}$$

$$AT_{i,j}(k_{i'^-}) = T + \sum_{k=1}^{k_{i'^-}} t(r_{i,j}(k-1), r_{i,j}(k)) + \sum_{\substack{1 \le k < k_{i'^-} \\ r_{i,j}(k) \in P}} WT_{i,j}(k) \tag{2}$$

$$TA_{i,j}(i'^-) = max\{0, AT_{i,j}(k_{i'^-}) - DT_{i'}\} \tag{3}$$

where $WT_{i,j}(k)$ is the waiting time before rider q_j can take away the meal at pickup node $r_{i,j}(k)$, $AT_{i,j}(k)$ is the time when rider q_j arrives at node $r_{i,j}(k)$ and $k_{i'^-}$ is the index of the delivery node of order $o(i')$ in route $R_{i,j}$.

Then, the calculation of total tardiness $TT_{i,j}$ for route $R_{i,j}$ is in (4).

$$TT_{i,j} = \sum_{r_{i,j}(k) \in D} TA_{i,j}(r_{i,j}(k)), \tag{4}$$

The delivery efficiency of each rider q_j can be reflected by the total travel distance $TD_{i,j}$ of route $R_{i,j}$ in (5). The larger the travel distance, the lower the delivery efficiency.

$$TD_{i,j} = \sum_{k=1}^{n_{i,j}} d(r_{i,j}(k-1), r_{i,j}(k)), \tag{5}$$

The route cost $RC_{i,j}$ of $R_{i,j}$ can be calculated as (6).

$$RC_{i,j} = TT_{i,j} + TD_{i,j}, \tag{6}$$

The goal of the DE-based routing phase is to minimize the route cost $RC_{i,j}$. However, from the perspective of riders, they are unwilling to alter their routes too much when dispatching new orders to them. In this case, the variation of the routes is considered in the heuristic-based dispatching phase to meet the riders' demand. Consequently, we define the route cost difference $C_{i,j}$ ($i = 1, 2, ..., n^{new}, j = 1, 2, ..., m$) between the new route and the old route as (7) when dispatching order $o(i)$ to rider q_j.

$$C_{i,j} = RC_{i,j} - RC_{0,j} \tag{7}$$

The goal of the heuristic-based dispatching phase, also the final goal of HDE, is to minimize the total cost TC in (8) after dispatching all the new orders.

$$TC = \sum_{i=1}^{n^{new}} \sum_{j=1}^{m} C_{i,j} X_{i,j} \tag{8}$$

III. The Proposed HDE

To solve the OMDP effectively, the problem is decomposed into two sub-problems. The upper level problem is a matching problem between new orders and riders, while the lower level problem is a PDP. Accordingly, a hybrid differential evolution algorithm is proposed with two phases to deal with each sub-problem respectively. For the lower level problem, a discrete DE is designed as the routing phase to schedule the new routes. For the upper level problem, a dispatching heuristic based on the regret heuristic is designed as the dispatching phase to determine the dispatching scheme. The framework of HDE is illustrated in Fig. 2.

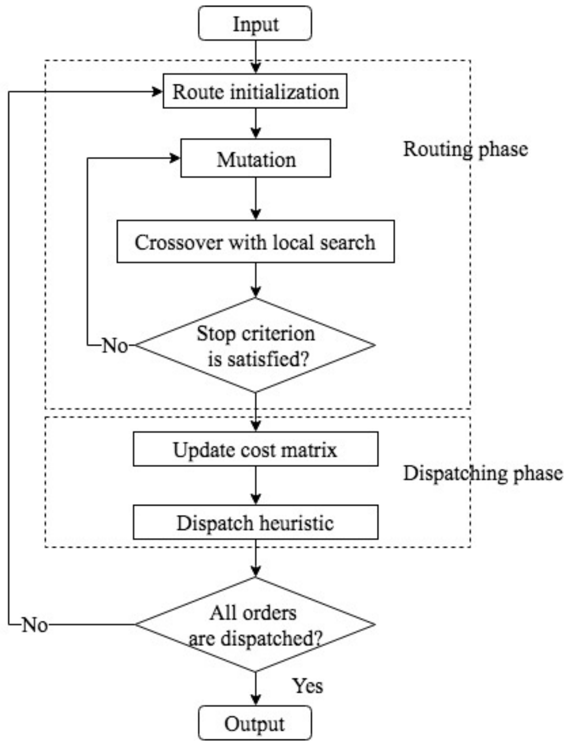


Fig. 2: The framework of HDE.

A. Solution Representation

In HDE, a solution is represented by a permutation sequence with a starting node 0 representing the site node of the rider, pickup nodes and delivery nodes. The length of the sequence depends on the number of the orders that are dispatched to the rider and whether the orders have been picked up or not. Specifically, a pickup node and a delivery node are associated with an order if it has not been picked up, while only a delivery node is associated with an order if it has been picked up. Each node is numbered in an ascending order.

Suppose 3 orders to be served by a rider and the states of the orders are listed in Table II. Since o_1 has not been picked up, the pickup node and delivery node of o_1 are numbered as 1 and 2 respectively. o_2 has been picked up so only a delivery node needs to be numbered, which is 3. Finally, the pickup and delivery nodes are numbered as 4 and 5 respectively according to the state of o_3. A feasible solution must start at 0 and cannot violate the precedence constraint and the capacity constraint. For example, if maximum capacity is 6kg, then [0,3,1,2,4,5] is a feasible solution while [1,0,2,3,4,5] (starting at 1), [0,1,4,2,3,5] (violating capacity constraint) and [0,3,2,1,4,5] (violating precedence constraint) are infeasible solutions.

TABLE II: Data of the example

order	o_1	o_2	o_3
state	to be picked up	picked up	to be picked up
pickup node	1	null	4
delivery node	2	3	5
weight	2kg	3kg	2kg

B. Initialization

In HDE, two methods are used to generate the initial population. To improve the population quality, an individual is generated by a designed heuristic called as urgency-based insertion heuristic (UIH). Besides, the rest individuals are generated randomly to maintain the population diversity. The procedure of UIH is shown in Algorithm 1.

C. Discrete Mutation Operator

Mutation is a critical component in DE because the differentiation information is inherited by the mutant vector. For the traditional DE, the float-point encoding scheme is usually adopted to solve the continuous optimization problems. The mutant vector can be generated via arithmetical calculation on the float-point numbers. However, in HDE, the solutions are encoded in permutation so the traditional way of generating mutant vector in continuous domain cannot be introduced directly. Thus, a discrete mutation operator is designed as follows by extracting the differentiation information from two individuals to generate the mutant individual $V_x = [V_x^1, V_x^2, ..., V_x^N]$, where N is the length of the individual, i.e., the number of the nodes in the permutation sequence.

$$V_x = X_a \oplus F \otimes (X_b - X_c) \tag{9}$$

Algorithm 1 Pseudo-codes of the UIH

```
Require: old orders $O_j^{old} = \{o_j^{old}(1), o_j^{old}(2), \ldots, o_j^{old}(n_j)\}$ of rider $q_j$, new order $o(i)$;
1:  Let $O_j = O_j \cup \{o(i)\} = \{o_j(1), o_j(2), \ldots, o_j(n_j+1)\}$;
2:  Let $\hat{O} = \varnothing$, $\bar{O} = \varnothing$, $\hat{n} = 0$, $\bar{n} = 0$;
3:  for $k = 1 : n_j + 1$ do
4:      if $o_j(k)$ has been picked up then
5:          $\hat{O} = \hat{O} \cup \{o_j(k)\}$, $\hat{n} = \hat{n} + 1$;
6:      else
7:          $\bar{O} = \bar{O} \cup \{o_j(k)\}$, $\bar{n} = \bar{n} + 1$;
8:      end if
9:  end for
10: Sort $\hat{O}$ and $\bar{O}$ in ascending order of due time and obtain $\hat{O}' = \{\hat{o}_j(1), ..., \hat{o}_j(\hat{n})\}$ and $\bar{O}' = \{\bar{o}_j(1), ..., \bar{o}_j(\bar{n})\}$;
11: Let $O_j = \{\hat{o}_j(1), ..., \hat{o}_j(\hat{n}), \bar{o}_j(1), ..., \bar{o}_j(\bar{n})\}$;
12: Encode $O_j$ according to the method in III-A and obtain a permutation sequence $(0, 1, 2, ..., \hat{n} + 2\bar{n})$;
13: Let $newP = (0)$;
14: for $k = 1 : \hat{n} + 2\bar{n}$ do
15:     Insert $k$ into the position in $newP$ with the minimum route cost without violating the constraints;
16: end for
```

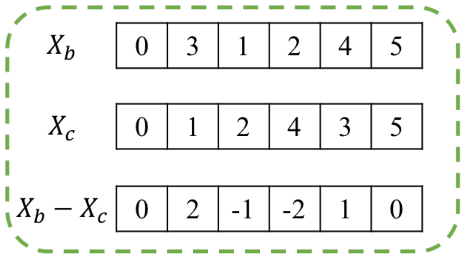
In (9), $X_a = [X_a^1, X_a^2, .., X_a^N]$, $X_b = [X_b^1, X_b^2, .., X_b^N]$ and $X_c = [X_c^1, X_c^2, .., X_c^N]$ are three randomly selected individuals in population (a, b and c are mutually different) and F is a mutation scale factor to control the amplification of the differential variation. The mutation operator is composed of two components. One component is the weighted difference $\Delta_x = [\Delta_x^1, \Delta_x^2, .., \Delta_x^N]$ between two individuals X_b and X_c, which can be calculated as follows.

$$\begin{aligned} &\Delta_x = F \otimes (X_b - X_c) \\ &\Leftrightarrow \Delta_x^h = \begin{cases} X_b^h - X_c^h, \text{if } rand(h) \le F \\ 0, \text{otherwise} \end{cases} \quad h = 1, ..., N \end{aligned} \tag{10}$$

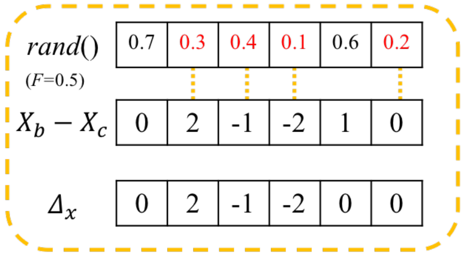
The other component is to produce the mutant individual by adding the target individual and the weighted difference, which can be calculated as follows.

$$\begin{aligned} &V_x = X_a \oplus \Delta_x \\ &\Leftrightarrow V_x^h = (X_a^h + \Delta_x^h + N)\%N, \ h = 1, ..., N \end{aligned} \tag{11}$$

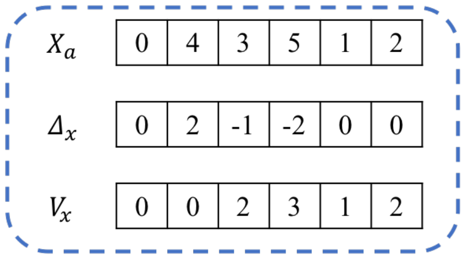
where % denotes the modulus operator to ensure that each node V_x^h ranges from 0 to $N-1$. An example is given in Fig. 3 to illustrate the procedure of mutation.



(a) Calculate the difference



(b) Calculate the weighted difference



(c) Add the weighted difference to the target individual

Fig. 3: An example of the mutation

D. Discrete Crossover Operator

After executing the discrete mutation operator, the mutant individual may be illegal because some nodes are lost or repeated. Therefore, a discrete crossover operator is presented to produce a legal individual by combining the target individual with the mutant individual. Besides, to further improve the solution quality, an insertion-based local search is embedded in the crossover operator. The procedure of the discrete crossover operator is shown in Algorithm 2 and an example is given in Fig. 4. CR is the crossover scale factor to control the amplification of the crossover operator. The larger CR is, the less information will be inherited from the target individual.

E. Dispatching Heuristics

In HDE, two dispatching heuristics are presented for solving the order dispatching sub-problem. To determine the best rider for each order, the costs of dispatching each order to each rider are calculated according to (7) so that a cost matrix CM between all riders and all new orders is generated. Based on the cost matrix, we design two heuristics to dispatch the orders, which are inspired by the insertion heuristics for VRP in [19]. The first is the greedy dispatching heuristic (GDH), which selects the order with minimum cost in the cost matrix and dispatches it to the corresponding rider. Afterwards, the row of the dispatched order in the cost matrix is eliminated and the old route of the selected rider is replaced by the corresponding new route. Accordingly, the new routes of dispatching the remaining orders to this rider should be rescheduled by the proposed DE and then the cost matrix is updated as well.

Algorithm 2 Pseudo-codes of crossover

```
Require: mutant individual $V_x = [V_x^1, V_x^2, ..., V_x^N]$, target individual $X_t = [X_t^1, X_t^2, ..., X_t^N]$;
 1: Let $\tilde{V}_x = \varnothing$, $\tilde{n}_x = 0$;
 2: for $k = 1 : N$ do
 3:     if $V_x^k = 0 || V_x^k \in \tilde{V}_x$ then
 4:         continue;
 5:     end if
 6:     if $rand() < CR$ then
 7:         $\tilde{V}_x = \tilde{V}_x \cup \{V_x^k\}$, $\tilde{n}_x = \tilde{n}_x + 1$;
 8:     end if
 9: end for
10: if $\tilde{V}_x = \varnothing$ then
11:     $k = 1 + rand()\%(N-1)$;
12:     $\tilde{V}_x = \tilde{V}_x \cup \{V_x^k\}$;
13:     $\tilde{n}_x = \tilde{n}_x + 1$;
14: else
15:     Move the pickup node to the previous adjacent position of its delivery node if $\tilde{V}_x$ violating the precedence constraint; //repair the crossover vector
16: end if
17: Remove the nodes of $X_t$ that $X_t$ and $\tilde{V}_x$ share;
18: for $k = 1 : \tilde{n}_x$ do
19:     Insert $\tilde{V}_x^k \in \tilde{V}_x$ into the position in $X_t$ with minimum route cost without violating the constraints;
20: end for
```

Repeat the steps and all new orders can be dispatched. The second is the regret dispatching heuristic (RDH). Different from the GHD, the RDH dispatches orders by considering the regret value [19], which is defined as the cost difference of the second-best rider and the best rider in this paper. In RDH, the order with maximum regret value will be dispatched to its best rider. Similar to the GDH, the old route, new routes and the cost matrix will be updated. Then, repeat dispatching the order with maximum regret value and updating until all the new orders are dispatched. The mechanism of RDH incorporates a kind of look ahead information [19] in case the rest orders will be assigned to a rider with relative large value of total cost. The pseudo codes of the two heuristics are shown in algorithm 3 and algorithm 4. In this paper, two hybrid algorithms, i.e., DE-GDH and HDE, are presented by embedding the proposed DE into GDH and RDH, respectively.

IV. Computational Results

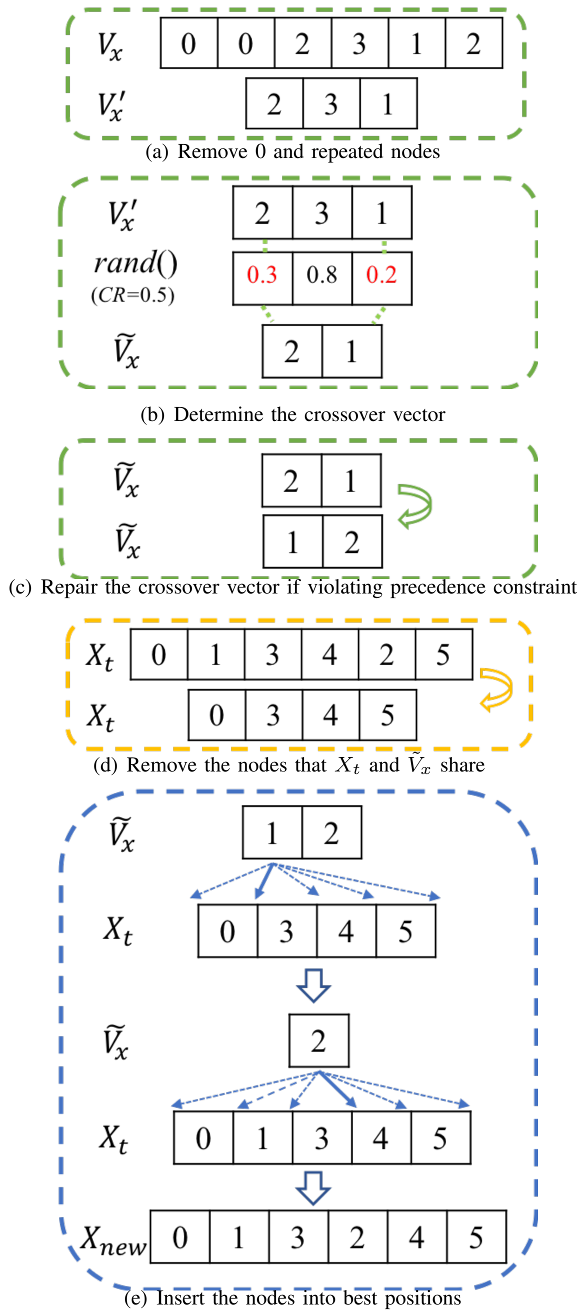
To test the performance of the proposed algorithm, numerical experiments are conducted and the results are shown in this section. The benchmark instances are generated from the real data of Meituan-Dianping platform. The instances are classified into 10 groups by the number of new orders. Each group contains 10 different instances so there are 100 instances in total. Table III gives the ranges of the number of new orders, the number of riders and the number of old orders for each rider. Limited by the article space, the preparation time of each order, the travel distance and travel time between each pair of

Algorithm 3 Pseudo-codes of GDH

```
Require: The set of new orders $O^{new}$, the set of riders $Q$ and the old route $R_{0,j}$ of each rider $q_j$;
 1: for $i = 1 : n^{new}$ do
 2:     for $j = 1 : m$ do
 3:         Schedule the new route $R_{i,j}$ of dispatching order $o^{new}(i)$ to rider $q_j$ using the proposed DE;
 4:         Calculate the cost $C_{i,j}$ according to $R_{i,j}$;
 5:     end for
 6: end for
 7: Let $N = n^{new}$;
 8: Let $I = \{1, 2, ..., n^{new}\}$;
 9: while $N > 0$ do
10:     Find the minimum cost $C_{i',j'}$ in $CM$;
11:     Dispatch order $o(i')$ to rider $q_{j'}$;
12:     Delete the $i'$th row of $CM$;
13:     Let $N = N - 1$;
14:     Let $I = I \backslash \{i'\}$;
15:     Let $R_{0,j'} = R_{i',j'}$;
16:     Reschedule the route of $q_{j'}$ using the proposed DE and obtain the new route $R_{i,j'}$, $\forall i \in I$;
17:     Update cost $C_{i,j'}$ according to $R_{i,j'}$, $\forall i \in I$;
18: end while
```

Algorithm 4 Pseudo-codes of RDH

```
Require: The set of new orders $O^{new}$, the set of riders $Q$ and the old route $R_{0,j}$ of each rider $q_j$;
 1: for $i = 1 : n^{new}$ do
 2:     for $j = 1 : m$ do
 3:         Schedule the new route $R_{i,j}$ of dispatching order $o^{new}(i)$ to rider $q_j$ by using the proposed DE;
 4:         Calculate the cost $C_{i,j}$ according to $R_{i,j}$;
 5:     end for
 6: end for
 7: Let $N = n^{new}$;
 8: Let $I = \{1, 2, ..., n^{new}\}$;
 9: while $N > 0$ do
10:     for each $i \in I$ do
11:         Find the minimum cost $C_{i,j'}$ and the second minimum cost $C_{i,j''}$;
12:         Let $RV_i = C_{i,j''} - C_{i,j'}$;
13:     end for
14:     Find the maximum regret value $RV_{i'}$;
15:     Dispatch order $o(i')$ to rider $q_{j'}$;
16:     Delete the $i'$th row of $CM$;
17:     Let $N = N - 1$;
18:     Let $I = I \backslash \{i'\}$;
19:     Let $R_{0,j'} = R_{i',j'}$;
20:     Reschedule the route of $q_{j'}$ using the proposed DE and obtain the new route $R_{i,j'}$, $\forall i \in I$;
21:     Update cost $C_{i,j'}$ according to $R_{i,j'}$, $\forall i \in I$;
22: end while
```



(a) Remove 0 and repeated nodes

(b) Determine the crossover vector

(c) Repair the crossover vector if violating precedence constraint

(d) Remove the nodes that X_t and $\tilde{V}_x$ share

(e) Insert the nodes into best positions

Fig. 4: An example of the crossover

nodes, the maximum capacity of each rider and other needed data are not given in this paper but known in the instances.

To the best of our knowledge, there are few published works addressing the OMDP under our scenario. In this case, the improved differential evolution (IDE) proposed in [18] is adopted as comparative algorithm due to the similarity in the problem characteristics. To make fair comparison, the IDE shares the same way of initialization and is combined with both of the dispatching heuristics to implement the order dispatching, which produces two variants of IDE, i.e., IDE-GDH and IDE-RDH. For DE-GDH and HDE, the population

TABLE III: The scale of the instances

Group	Range		
	n^{new}	n_j	m
1	[1,10]	[0,4]	[63,200]
2	[11,20]	[0,3]	[318,407]
3	[21,30]	[0,4]	[313,413]
4	[31,40]	[0,5]	[382,411]
5	[41,50]	[0,5]	[328,410]
6	[51,60]	[0,5]	[347,407]
7	[61,70]	[0,5]	[348,409]
8	[71,80]	[0,5]	[361,400]
9	[81,90]	[0,5]	[382,402]
10	[91,100]	[0,5]	[380,408]

size, mutation scale factor and crossover scale factor are set as 20, 0.5 and 0.5, respectively. The parameters of IDE are set according to [18]. The terminal criterion of the the lower level sub-problem relates to the number of route nodes $n_{i,j}$, which is set as 0.5×$n_{i,j}$ milliseconds CPU time. Each algorithm stops when all the new orders have been dispatched. Besides, each algorithm is run $TS = 30$ times independently on each instance. All the algorithms are coded in Java and the experiments are run on a MacBook Pro @ 2.2 GHz processors / 16 GB RAM in Mac OS. The average relative percentage deviation ($aRPD$) and standard deviation (SD) of the normalized TC (nTC) are adopted as the indicators to evaluate the performances of the algorithms, which are defined as follows.

$$aRPD = \sum_{i=1}^{TS}(alg_i - opt)/opt/TS \times 100 \tag{12}$$

where alg_i is the TC obtained by a certain algorithm in ith run and opt is the best TC obtained by the four algorithms in all independent runs on a certain instance. It is clearly that a smaller value of $aRPD$ indicates better performance of the corresponding algorithm.

$$nTC_i = (alg_i - min)/(max - min) \tag{13}$$

$$SD = \sqrt{\frac{1}{TS}\sum_{i=1}^{TS}(nTC_i - \frac{1}{TS}\sum_{j=1}^{TS}(nTC_j))^2} \tag{14}$$

where nTC_i is the nTC of a certain algorithm in ith run and min and max is the minimum TC and maximum TC obtained by the four algorithms in all independent runs on a certain instance. It is clearly that a smaller value of SD indicates a better stability.

The $aRPD$s and SDs on the instances from the same group are averaged and listed in Table IV and Table V respectively. In Table IV, the results show that the average $aRPD$s of HDE are smaller that those of the other three algorithms on all instances. Therefore, it can be concluded that HDE performs better than the other algorithms. Besides, in Table V, it can be seen that the average SDs of HDE are smaller than those of the other algorithms on a majority of instances, which reveals better stability of HDE.

TABLE IV: The results on $aRPD$

Group	Algorithm			
	IDE-GDH	**IDE-RDH**	**DE-GDH**	**HDE**
1	4.4441	2.4297	1.6234	**0.0851**
2	9.3658	1.9551	7.1207	**0.2902**
3	6.0511	3.1223	5.3036	**1.1767**
4	7.5903	3.5221	5.7566	**1.3512**
5	16.1437	4.4737	10.5985	**0.4272**
6	9.2301	2.1849	6.5424	**1.2771**
7	12.7024	3.3795	9.6139	**0.5379**
8	13.0374	3.1608	11.6333	**0.7053**
9	19.0265	3.4714	15.6612	**0.6576**
10	19.2685	3.2873	17.2687	**0.7326**
Average	11.6860	3.0987	9.1122	**0.7241**

TABLE V: The results on SD

Group	Algorithm			
	IDE-GDH	**IDE-RDH**	**DE-GDH**	**HDE**
1	0.1224	0.0948	0.0752	**0.0318**
2	0.1859	0.1071	**0.0026**	0.0087
3	0.1098	0.0786	0.0180	**0.0013**
4	0.1021	0.0697	**0.0131**	0.0175
5	0.0730	0.0601	0.0216	**0.0124**
6	0.0802	0.0867	**0.0360**	0.0558
7	0.0723	0.0583	0.0495	**0.0282**
8	0.0739	0.0792	0.0310	**0.0272**
9	0.0624	0.0308	0.0310	**0.0227**
10	0.0590	0.0388	0.0327	**0.0161**
Average	0.0941	0.0704	0.0311	**0.0222**

V. Conclusions and Future Works

In this paper, an online meal delivery problem is addressed. To reduce the search space, the problem is decomposed into two sub-problems: the order dispatching problem and the pickup and delivery problem. A hybrid differential evolution algorithm with two phases is proposed to solve the two sub-problem respectively. Computational results demonstrate the effectiveness of the proposed algorithm. The superiority of our algorithm mainly owes to the following aspects.

- Utilization of the heuristic to produce a population with good quality by considering the urgency of the orders.
- Design of the discrete mutation operator to enhance the exploration ability.
- Design of the discrete crossover operator embedded with an insertion-based local search to enhance the exploitation ability.
- Design of the dispatching heuristic based on the regret value to produce good dispatching schemes.

In future work, more complexities will be considered in the formulation of PDP under our online meal delivery scenario. Besides, it is interesting to develop more efficient and effective dispatching methods by improving the regret dispatching heuristic or introducing the machine learning.

Acknowledgment

This research is supported by the National Science Fund for Distinguished Young Scholars of China [No. 61525304], the National Natural Science Foundation of China [No. 61873328] and Meituan-Dianping Group.

References

[1] M.-D. Group. (2019, Nov.) Announcement of the Results for the Three Months Ended September 30, 2019. [Online]. Available: https://www1.hkexnews.hk/listedco/listconews/sehk/2019/1121/2019112100554.pdf.

[2] G. Berbeglia, J. F. Cordeau, I. Gribkovskaia, and G. Laporteet, Static pickup and delivery problems: a classification scheme and survey," *Top*, vol. 50, no.1, pp. 1-31, 2007.

[3] K. S. Ruland and E. Y. Rodin, "The pickup and delivery problem: faces and branch-and-cut algorithm," *Computers & Mathematics with Applications*, vol. 33, no. 12, pp. 1-13, 1997.

[4] Y. Dumas, J. Desrosiers, and F. Soumis, "The pickup and delivery problem with time windows," *European Journal of Operational Research*, vol. 54, no. 1, pp. 7-22, 1991.

[5] S. Ropke and J. F. Cordeau, "Branch and cut and price for the pickup and delivery problem with time windows," *Transportation Science*, vol. 43, no. 3, pp. 267-286, 2009.

[6] Q. Lu and M. M. Dessouky, "A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows," *European Journal of Operational Research*, vol. 175, no. 2, pp. 672-687, 2006.

[7] M. Şahin, G. Çavuşlar, T. Öncan, G. Şahin, and D. T. Aksu, "An efficient heuristic for the multi-vehicle one-to-one pickup and delivery problem with split loads," *Transportation Research Part C: Emerging Technologies*, vol. 27, pp. 169-188, 2013.

[8] G. Pankratz, "A grouping genetic algorithm for the pickup and delivery problem with time windows," *Or Spectrum*, vol. 27, no. 1, pp. 21-41, 2005.

[9] T. J. Ai and V. Kachitvichyanukul, "A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery," *Computers & Operations Research*, vol. 36, no. 5, pp. 1693-1702, 2009.

[10] Z. Zhu, J. Xiao, S. He, Z. Ji, and Y. Sun, "A multi-objective memetic algorithm based on locality-sensitive hashing for one-to-many-to-one dynamic pickup-and-delivery problem," *Information Sciences*, vol. 329, pp. 73-89, 2016.

[11] B. Çatay, "A new saving-based ant algorithm for the vehicle routing problem with simultaneous pickup and delivery," *Expert Systems with Applications*, vol. 37, no. 10, pp. 6809-6817, 2010.

[12] Y.-N. Guo, J. Cheng, S. Luo, and D. Gong, "Robust dynamic multi-objective vehicle routing optimization method," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 15, no.6, pp. 1891-1903, 2017.

[13] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341-359, 1997.

[14] L. Wang, Q. -K. Pan, P. N. Suganthan, W. -H. Wang, and Y.-M. Wang, "A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems," *Computers & Operations Research*, vol. 37, no. 3, pp. 509-520, 2010.

[15] E. Berhan, P. Krömer, D. Kitaw, A. Abraham, and V. Snášel, "Solving stochastic vehicle routing problem with real simultaneous pickup and delivery using differential evolution," *Innovations in Bio-inspired Computing and Applications*, pp. 187-200, 2014.

[16] B. E. Teoh, S. G. Ponnambalam, and G. Kanagaraj, "Differential evolution algorithm with local search for capacitated vehicle routing problem," *International Journal of Bio-Inspired Computation*, vol. 7, no. 5, pp. 321-342, 2015.

[17] D. Dechampai, L. Tanwanichkul, K. Sethanan, and R. Pitakaso, "A differential evolution algorithm for the capacitated VRP with flexibility of mixing pickup and delivery services and the maximum duration of a route in poultry industry," *Journal of Intelligent Manufacturing*, vol. 28, no.6, pp. 1357-1376, 2017.

[18] M. Lai and E. Cao, "An improved differential evolution algorithm for vehicle routing problem with simultaneous pickups and deliveries and time windows," *Engineering Applications of Artificial Intelligence*, vol. 23, no. 2, pp. 188-195, 2010.

[19] S. Ropke and D. Pisinger, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows," *Transportation Science*, vol. 40, no.4, pp. 455-472, 2006.