An Effective Iterated Greedy Algorithm for Online Route Planning Problem

Xing Wang[1], Shengyao Wang[2], Ling Wang[1],
Huanyu Zheng[2], Jinghua Hao[2], Renqing He[2], Zhizhao Sun[2]
[1]*Department of Automation*, *Tsinghua University*, Beijing, China
[2]*Meituan-Dianping Group*, Beijing, China
wang-x17@mails.tsinghua.edu.cn, wangshengyao@meituan.com,
wangling@mail.tsinghua.edu.cn
{zhenghuanyu, haojinghua, herenqing, sunzhizhao}@meituan.com

***Abstract*—Route planning serves as the most fundamental part in modern food ordering and delivery system, which is also a classical combinatorial optimization problem. In this paper, we study an online extension of traditional route planning problem where a near-optimal solution has to be generated in a very short time. A simple and effective iterated greedy algorithm is presented along with problem-specific initialization rules, destruction procedure and construction procedure. We also propose a local search method, consisting of two adjustment operators and two neighborhood search operators. With experimental results, we show that our algorithm outperforms the compared evolutionary algorithms and has the capability of providing high-quality solutions within milliseconds.**

***Index Terms*—Food ordering and delivery, Route planning, Iterated greedy, Evolutionary algorithms**

I. Introduction

A. Background

With the fast development of internet and e-commerce, food ordering and delivery is becoming an indispensable part in modern life. Food ordering platforms such as Grubhub, UberEats and Meituan-Dianping provide millions of customers with convenient food delivery service. For instance, on Meituan-Dianping food delivery platform, there are more than 3.6 million restaurants available and over 24 million daily orders generated and accomplished, according to the statement of Meituan-Dianping Group [1]. An earlier research by Morgan Stanley [2] also shows that online food delivery is expected to grow by 16% annual compound rate in the next 5 years in the US, which indicates bright prospects of food ordering and delivery service.

However, challenge comes together with opportunity. To run a food delivery platform successfully, several key stakeholders have to be considered, which are the customer, the restaurant and the driver. Customers want their orders to be delivered punctually and safely. Restaurants aim to maximize the profit. As for drivers, completing the delivery tasks as many as possible with the least driving cost is their primary goal in order to get decent wages.

The process of food ordering and delivery is shown in Fig. 1. Customers order via their mobile application and then the orders are collected by the food delivery platform. The food delivery platform pushes the order to the restaurants and dispatches the orders to the available drivers at the same time. Meanwhile, feasible routes are also generated by the platform for drivers so that they can complete the pickup and delivery tasks on time. Under this situation, there are two vital problems a dispatching system should consider. On the one hand, the quality of final schemes for dispatching and route planning have to be guaranteed. If an order is not picked up or delivered on time, the food may be not fresh and the customer will wait too long to be satisfied. Consequently, there will be a decreasing possibility that the customer will use the delivery service of the driver and patronize the restaurant again [3]. On the other hand, the process of finding a promising solution has to be quick enough because large amount of orders come into the food delivery platform continuously, which leaves extremely little time for order dispatching and route planning. For instance, at the peak hours of a day, there will be thousands of orders generated every minute and a driver may have to serve more than 10 orders simultaneously, which results in a very large solution space. Therefore, how to find favorable schemes within milliseconds is a challenging and urgent issue for food delivery platforms.

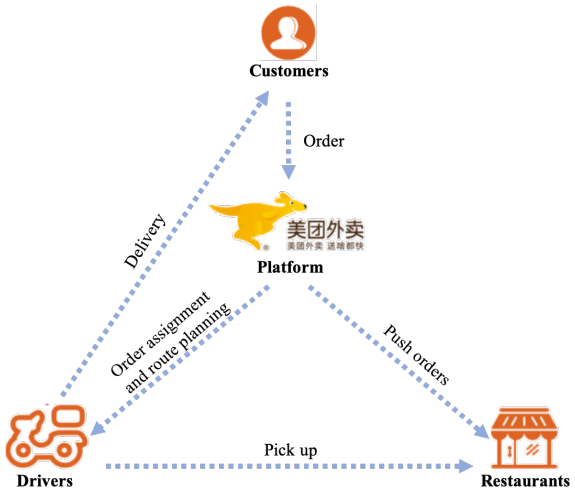

Fig. 1. The food ordering and delivery process.

Generally, route planning serves as the most fundamental

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

part of the food ordering and delivery process, deciding the final route that the driver will execute. Therefore, the performance of the whole dispatching system largely depends on the quality of solutions to the route planning problem. Meanwhile, as an online algorithm, route planning has to be finished within milliseconds so as to be used millions of times a day. However, finding an optimal solution is extremely time-consuming, while a fast heuristic often results in solutions of poor quality. Under this situation, our paper aims to design an effective evolutionary algorithm to provide high-quality routes for drivers, while consuming very short computational time. The target we considered here is to minimize the total cost, including the time delay of orders and the driving distance of the driver, which take care of the feelings of customers and restaurants, and the workload of the driver, respectively.

B. Literature Review

This section presents a review of literature related to the online route planning problem (ORPP). The ORPP is most similar to the single vehicle pickup and delivery problem with time windows (SVPDPTW), which is a special case of pickup and delivery problem with time windows (PDPTW) [4]–[6]. SVPDPTW assumes one vehicle serving multiple orders. Each order is associated with a pickup location and delivery location, which have to be visited within a certain time window.

Studies on traditional SVPDPTW are extensive and most of them focus on proposing effective heuristics and meta-heuristics. Bruggen et al. [7]. proposed a two-phase heuristic, which takes a variable depth arc exchange procedure as a neighborhood move. In the first phase a feasible solution is constructed while in the second phase the solution is iteratively improved. Their approach is able to obtain near-optimal solutions most of the time, despite that low-quality or infeasible solution may occur when the algorithm is trapped in a poor local optima. Later, Jih and Hsu [8] designed a hybrid algorithm to solve SVPDPTW under static case and dynamic case, respectively. Their approach tries to combine the advantage of dynamic programming and genetic algorithm. Experimental results show that genetic algorithm converges better with the initial population passed from dynamic programming, which improves the genetic algorithm in approximating near-optimal solutions. Landrieu et al. [9] presented two algorithms based on tabu search to solve the problem. The first is the regular deterministic tabu search, where they designed two classical node interchange procedures for the neighborhood structure, which are a swap operation and an insertion operation. Secondly, they proposed the probabilistic tabu search, which can choose the next-best candidate with probabilities at each iteration. Their computational results show that feasible solutions can be reached in a reasonable amount of time for instances under 40 customers. However, for larger problem sizes, obtaining a feasible solution can possibly take more than one hour. Hosny and Mumford [10] studied the SVPDPTW with genetic algorithms, focusing on investigating effective problem-specific genetic encoding and operators. They used a duplicate gene encoding that guarantees the satisfaction of the precedence constraint. Furthermore, they compared four genetic operators, which are a modified 2-child merge crossover (MX1), a new PDPX crossover operator, a regular random swap mutation and a directed mutation. They concluded from computational results that the directed mutation operator seems to be useful in guiding the search towards feasible solutions while PDPX crossover does not. In the later work of Hosny and Mumford [11], they designed an algorithm based on simulated annealing (SA), adopting intelligent neighborhood moves that are guided by time windows and a hill climbing heuristic similar to the 3-stage SA. Simulation results show that the intelligent neighborhood moves are successful in both SA and hill climbing heuristic, although the improvement is more dramatic in the context of SA. Recent researches mainly focus on different variants of the pickup and delivery problems with time windows, which can be found in the review by Braekers et al. [12].

Traditional evolutionary algorithms such as GA and SA, often take large amount of time to converge to an acceptable solution, which is not applicable to the setting of ORPP. In this paper, we propose a fast iterated greedy (IG) algorithm that can generate high-quality solutions within milliseconds. Compared with other complicated evolutionary algorithms, IG is remarkably simple with mainly two phases, which are called the destruction phase and the construction phase. It is first developed to solve the scheduling problem by Ruiz and Stützle [13]. Later works on solving shop scheduling problems by different kinds of IG can be seen in [14]–[16]. As for problems related to route planning problem, Karabulut and Tasgetiren [17] proposed a variable iterated greedy algorithm with changing neighborhood for the traveling salesman problem with time windows. Computational results confirm that their approach is either competitive or even better than the state-of-the-art methods. In the following section, we elaborate our iterated greedy algorithm which is tailored for solving the online route planning problem. To the best of our knowledge, our paper is the first to employ the iterated greedy algorithm to solve the ORPP. Through computational experiments on different scales of instances and compared with other evolutionary algorithms, we demonstrate that the proposed IG algorithm is efficient and effective for solving the ORPP.

The remainder of the paper is organized as follows. In section II we describe the online route planning problem and illustrate the constraints. The proposed IG algorithm is presented in section III, where we elaborate on the design of initialization, destruction phase, construction phase and local search method. Computational results and comparison of different algorithms are shown in section IV. Finally, section V closes the paper with some conclusions and future research ideas.

II. Problem Description

The online route planning problem assumes a set of orders $O = \{1, 2, \ldots, n\}$ to be served by one driver. Specifically,

the orders are classified into two categories. The first is the orders that only have delivery locations. That is to say, this kind of orders have already been fetched from the restaurant by the driver. The second one is the ordinary orders that have both pickup locations and delivery locations. Denote the set of the first kind of orders as $O_1 = \{1, \ldots, n_1\}$, the set of the second kind of orders as $O_2 = \{1, \ldots, n_2\}$. Then we have $O = O_1 \bigcup O_2$ and $n = n_1 + n_2$. Let i^+ be the pickup location of an order and i^- be the corresponding delivery location of the order i. The pickup location set of all orders can be denoted as $P = \{i^+ | i \in O_2\}$, and the delivery location set as $D = \{i^- | i \in O\}$. The current location of the driver is denoted as l_0, which is the starting point of every feasible route.

Every location $l \in P \bigcup D \bigcup \{l_0\}$ has a customer demand q_l, such that $q_l > 0$ for pickup locations, $q_l < 0$ for delivery locations, and for a pair of pickup and delivery locations of an order $i \in O_2$, $q_{i^+} + q_{i^-} = 0$. Note that current location of the driver also has a customer demand q_{l_0}, which indicates the initial load of the driver. The maximum capacity of the driver is set to Q, which is a realistic constraint because the total load carried by the driver cannot be infinite in real world. Moreover, each order $i \in O$ is associated with an earliest pickup time PT_i, before which the order cannot be fetched. Also, there is an estimated time of arrival ETA_i, around which the order is promised to be delivered to customers. If the real delivery time of an order is later than ETA_i, then positive time delay will occur, which dissatisfies the customers. For every possible location $k, l \in P \bigcup D \bigcup \{l_0\}$, the travel distance between them is denoted as $d_{k,l}$, and the travel time as $t_{k,l}$.

To construct a feasible solution for the ORPP, there are several requirements that need to be satisfied. First, a feasible route is supposed to start from the current location of the driver and end at one of the delivery locations, while each location has to be visited exactly once. Second, all locations must be served with the earliest pickup time and the estimated time of arrival in consideration as mentioned above. The third constraint is the capacity constraint, which prescribes the upper limit of the load that the driver can carry. Last but significantly, the pickup location has to be visited before the corresponding delivery location, namely the precedence constraint. A typical route is shown in Fig. 2, which contains three orders to be served, including 1 pickup location and 3 delivery locations.

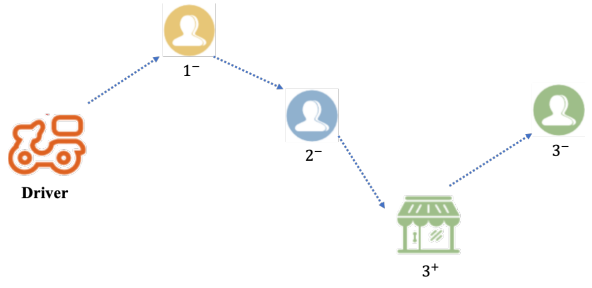

Fig. 2. A typical route for ORPP.

The objective function we considered in this paper is to minimize the total cost (TC) of the route, consisting of two parts that are the time delay (TD) of the route and the distance (DIS) traveled by the driver, respectively. Equation (1) gives the mathematical expression to calculate the objective function,

$$minTC = \sum_{i=1}^{n} max(D_i - ETA_i, 0) + \sum_{j=1}^{n_1+2n_2} d_{j-1,j} \quad (1)$$

where the first item is TD and the second is DIS. D_i represents the real delivery time of order i, which is calculated by the route evaluation module after the route is constructed. $d_{j-1,j}$ represents the distance between the jth point and its previous point in a candidate route.

III. Proposed Approach

In this section, we present our methodology for solving the ORPP by designing an effective iterated greedy algorithm. The outline of the proposed IG algorithm is shown in Fig. 3. It starts from some initial solution and iterates through a main loop which is composed of two kernel procedures called destruction and construction. In the destruction phase, a number of elements of the candidate solution are selected and removed, resulting a temporary partial solution. After that, the removed elements are reinserted into the partial solution with a reinsertion heuristic in the construction phase. When a new complete solution is constructed, an acceptance criterion is applied to decide whether the new solution substitutes the current solution. This process is repeated until some termination criterion is met, such as the maximum running time or the maximum number of iterations. Usually a problem-specific local search procedure is added to the iterated process in order to enhance the quality of the solutions generated by the construction operator, which will accelerate the convergence of the algorithm and save considerable amount of computational time. The steps of every segment of the proposed algorithm will be detailed in the following contents.

A. Initialization

Initialization often plays an important role in evolutionary algorithms, which can provide a promising starting point for the search process. For ORPP, considering the fact that the first kind of orders (that only have delivery locations) is normally more possible to be served before the second kind of orders (that have both pickup and delivery locations), we design a problem-specific insertion heuristic for initialization whose main steps are as follows:

Step 1: Classify the orders into two categories. The first set contains orders that only have delivery locations and the second set contains orders that have both pickup and delivery locations;

Step 2: Sort the orders inside each set according to some proposed sequencing rule and put the first set ahead of the second set;

Step 3: Sequentially take out the orders from the set and insert them into the partial route. For orders in the first set, directly place them in the best position of the route; For orders

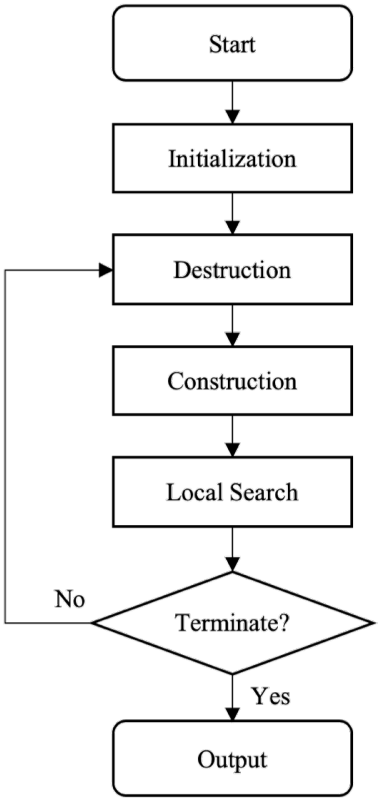

Fig. 3. The outline of the proposed IG algorithm.

in the second set, try all the possible pairs of positions of the pickup location and delivery location, then choose the best pair that does not violate the capacity constraint and precedence constraint. This step is executed iteratively until a complete solution is constructed.

The sequencing rule in step 2 has to be carefully designed, to which the quality of the initial solution is closely related. Therefore, we design four different sequencing rules, aiming at exploring different parts of the solution space.

1) Rule with Earliest-pickup-time (REPT): This sequencing rule sorts orders in an ascending order of their pickup times, which mainly considers that in real world, orders are only ready to be fetched after their corresponding pickup times. Hence, if the driver comes earlier than the earliest pickup time, extra waiting time will occur, which increases the possibility of customer dissatisfaction. In other words, orders that are ready first are supposed to be served first.

2) Rule with Estimated-time-of-arrival (RETA): This sequencing rule takes the deadline of orders as the most important index, which sorts orders according to increasing estimated time of arrival. RETA puts orders with a closer deadline ahead so as to arrange the time spent at each location reasonably and avoid large amount of time delay. To some extent, ETA reflects the urgency of the order and the difficulty of delivering the order punctually. Therefore, RETA is a natural sequencing rule that we will conceive intuitively.

3) Rule with Urgency (RU): Note that RETA only measures the urgency of orders in terms of time. Consequently, there are cases that RETA cannot handle. Consider a simple example in Fig. 4, where there are two orders to be delivered. The ETA of order 1 is earlier than that of order 2. According to RETA, order 1 is definitely more urgent than order 2. However, order 2 is farther to the current location of the driver, which means that it requires longer driving distance and consumes more time on the road. That is to say, order 2 may have bigger risk of generating positive time delay. In order to measure the urgency of orders more comprehensively, we propose RU which takes information of both time and distance of orders into consideration. Specifically, RU sorts orders in a descending order of urgency, which is computed by (2),

$$\delta_i = \frac{d_i}{ETA_i - ct} \tag{2}$$

where δ_i is the urgency of order i, ETA_i is the estimated time of arrival of order i, ct is the current time. For the first kind of orders, d_i is the distance between the delivery location of order i and the current location of the driver, while for the second kind of orders, d_i is the distance between current location of the driver and the pickup location of order i, plus the distance between the pickup location and the delivery location of order i. Therefore RU also distinguishes the orders that are not the same kind.

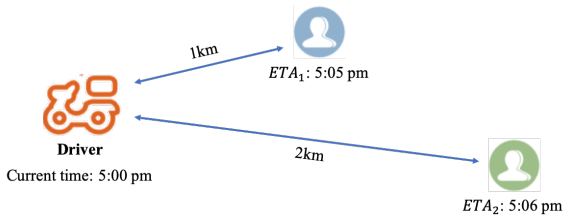

Fig. 4. Explanation of Rule with Urgency.

4) Rule with Hybridization (RH): Recall that we have two kind of orders and the sequencing rules mentioned above do not treat them differently except for RU. Nevertheless, different kinds of orders often require different sequencing rules. For instance, orders of the first kind do not have pickup locations but they are still sorted by the earliest pickup time according to REPT, which is not reasonable enough. Hence, we design RH as a hybrid sequencing rule which combines the REPT and RETA. RH sorts the first kind of orders by increasing estimated time of arrival and sorts the second by increasing earliest pickup time.

The performance of the proposed rules is compared in section IV, and the first two rules with the best performance are chosen to be used in the initialization procedure, which are RETA and RU. In other words, the insertion heuristic proposed above is executed twice with two different sequencing rules in initialization, and the solution with smaller objective function is selected as the initial solution, with which the iterative process of our IG algorithm starts.

B. Destruction, Construction and Acceptance Criterion

As mentioned before, after the initialization completes, the initial solution enters the destruction phase, where d elements are removed randomly according to Ruiz and Stützle [13]. However, in ORPP, a feasible solution must contain delivery locations for the first kind of orders and both pickup and delivery locations for the second kind of orders. Removing the locations in the incumbent route may result in infeasible partial solution that cannot be evaluated. Therefore, we propose an order-based random destruction operator for ORPP. In the destruction phase, we randomly choose d orders in the current solution and remove the location points related to those orders from the incumbent. This creates two partial sequences where the first is called π_R, containing the removed location points and the second is denoted as π_D, which is composed of the remaining elements in the original solution after destruction.

In the construction phase, all location points in π_R is reinserted into the partial solution π_D greedily. Firstly, π_R is classified and sorted in the way similar to step 1 and step 2 in the initialization phase. The only difference is that the sequencing rule used here is RETA, which is the best one among REPT, RETA, RU, and RH according to the comparison in section IV. During insertion, a candidate pair of location points or a single delivery location point is extracted sequentially from π_R and then tested in each feasible position of π_D. The best position with the minimal objective function is selected where the candidate elements are reinserted. After that, the candidate elements are removed from π_R. This process is repeated until π_R is empty.

At each iteration, a new solution is constructed after the destruction, construction and local search steps. Whether to accept the new solution as the incumbent solution or not has to be decided. A simple and frequently used way is to accept the new solution only when it is better than the incumbent solution. However, such criterion is often too greedy and myopic, which traps the searching process in some local optima. In order to avoid the stagnation of the evolution process, we accept the new solution with the probability of 1 if it is better than the incumbent and the probability of 0.5 if it is worse. This acceptance criterion gives an opportunity to evolve for those solutions that are currently worse but may be potential in the future while maintaining the priority of solutions that are better than the incumbent.

C. Local Search Method

In order to further improve the performance of the general algorithm, local search methods are often added. For ORPP, we design two problem-specific adjustment operators as well as two neighborhood search operators to enhance the quality of solutions.

The first adjustment operator is called backward adjustment, where the delivery location with the largest time ahead of its ETA in the route is moved backward and relocated at the best position. The process is executed for ls iterations, which indicates the depth of adjustment. Correspondingly, the second adjustment operator is forward adjustment, which moves forward the delivery location with the largest time delay and inserts it in the best position. These two adjustment operators aim at balancing the priority of some abnormal orders in order to avoid too much delay or advance of delivery. The pseudo-code of backward adjustment and forward adjustment is shown in Algorithm 1.

Algorithm 1 Pseudo-code of backward adjustment and forward adjustment

Require: the set of orders O of current solution π and the depth of adjustment ls;
Calculate $\Delta_i = D_i - ETA_i$ for each order $i \in O$;
%Backward adjustment
Let $cnt = 0$, $rank = 1$;
while $cnt < ls$ **do**
  Find the order $i^\star$ with the $rank$th smallest $\Delta_{i^\star}$;
  if $\Delta_{i^\star} < 0$ **then**
    Move the delivery location of order $i^\star$ backward to the best position and denote the new solution as π_{new};
    cnt++;
    if $TC(\pi_{new}) < TC(\pi)$ **then**
      $\pi = \pi_{new}$;
      Recalculate Δ_i for each order $i \in O$;
      $rank = 1$;
    else
      $rank$++;
    end if
  else
    break;
  end if
end while
%Forward adjustment
Let $cnt = 0$, $rank = 1$;
while $cnt < ls$ **do**
  Find the order $i^\star$ with the $rank$th largest $\Delta_{i^\star}$;
  if $\Delta_{i^\star} > 0$ **then**
    Move the delivery location of order $i^\star$ forward to the best position and denote the new solution as π_{new};
    cnt++;
    if $TC(\pi_{new}) < TC(\pi)$ **then**
      $\pi = \pi_{new}$;
      Recalculate Δ_i for each order $i \in O$;
      $rank = 1$;
    else
      $rank$++;
    end if
  else
    break;
  end if
end while

As for neighborhood search operators, we make use of the insertion neighborhood and the swap neighborhood that are commonly used in combinatorial problems. In the insertion neighborhood, each location point in the solution is extracted

and inserted into every other possible position. If a smaller objective function is obtained with a different position, then the location point is relocated and the process is continued with another different location point. The process terminates when all the location points in the solution have been tested in all possible positions without improvements. The swap neighborhood selects each location point in the solution sequentially, and tries to swap the selected location point with all the other location points if feasible. The acceptance and termination criterion are same as the insertion neighborhood stated above. The overall local search procedure starts with the backward adjustment, then the forward adjustment, followed by the insertion neighborhood search and swap neighborhood search.

IV. Experimental Results

In this section we evaluate the performance of our proposed methods through numerical experiments. First, the experimental settings are described. Then we compare four sequencing rules that are proposed for the initialization procedure to find the most efficient two rules to be used ultimately. Then we compare our proposed IG algorithm with some classical and effective algorithms in SVPDPTW, such as the Genetic algorithm (GA) and the Tabu Search (TS) algorithm, which are adapted to the ORPP because previous research on ORPP is scarce. Moreover, we test the performance of IG and compared algorithms under different settings of termination criterion to show the robustness of our algorithm.

A. Experimental Settings

We generated 1500 instances from the real food ordering and delivery platform in China, which are equally partitioned into three sets, the small scale (less than 10), the medium scale (10 to 20) and the large scale (larger than 20), according to the number of total pickup and delivery points.

The criteria we use to evaluate the performance of the algorithm are the total cost (TC) defined in section II and the average relative percentage deviation (ARPD), calculated by (3),

$$ARPD = \frac{\sum_{ins=1}^{num} \frac{Alg_{ins} - Best_{ins}}{Best_{ins}}}{num} \times 100 \qquad (3)$$

where ins represents some instance in the test set, num is the total number of instances, Alg_{ins} is the solution generated by a certain algorithm and $Best_{ins}$ is the best solution among all compared algorithms for the instance.

During the comparison, the initialization methods that use different sequencing rules are denoted by the name of their corresponding sequencing rules, which is REPT, RETA, RU and RH, respectively. The destruction parameter of our IG algorithm is set to $\frac{n}{2}$, where n is the total number of the pickup and delivery points, to balance the exploration and exploitation for different instances. The depth of local search ls is set to 5.

As for the compared evolutionary algorithms, the TS algorithm is implemented according to Landrieu et al. [9]. The tabu list size is set to 5. GA is implemented with the MX1 crossover operator and the random swap mutation operator [11]. The population size, crossover rate and mutation rate are set to 10, 1.0 and 0.1, respectively.

Each instance is run 30 times for every algorithm. Finally, considering the characteristic of ORPP, we set the maximum elapsed CPU time as the termination criterion, which is set to $2n$ milliseconds equally. All the experiments are run on a MacBook Pro with 2.2 GHz processors and 16 GB of RAM under Mac OS. Moreover, all the algorithms are coded in Java using Eclipse.

B. Comparison of Initialization Methods

To compare different sequencing rules in initialization and select the best two rules, we run four initialization methods on all the 1500 instances. Table I shows the average performance of initialization methods. As can be seen from the results, the CPU time consumed by each method is similar while RETA defeats the other three methods in terms of TC and ARPD. RU is the second best with a very close TC to RETA and an acceptable ARPD which is 1.24% larger than that of RETA. Note that REPT and RH have a very large time delay compared with RETA and RU. This is probably due to the reason that RETA and RU attach more weight on delivering orders on time, on which REPT and RH consider less. In addition, RH is slightly better than REPT in terms of time delay, which demonstrates the importance of considering the urgency of order while sequencing. Table II details the performance of four initialization methods under different instance sets, from which similar conclusions can be obtained. From the results presented above, it is obviously reasonable to select RETA and RU as the best two rules to be used in the initialization of our proposed IG algorithm.

TABLE I
THE AVERAGE PERFORMANCE OF INITIALIZATION

Initialization	TC	TD	ARPD (%)	CPU Elapse (ms)
REPT	10.96	6.61	24.17	0.50
RETA	9.84	5.69	**10.59**	0.48
RU	9.86	5.64	**11.83**	0.52
RH	10.84	6.39	22.58	0.52

C. Comparison of our IG Algorithm and other Evolutionary Algorithms

In this subsection, we compare the proposed IG algorithm with the adapted evolutionary algorithms, including Genetic algorithm and Tabu Search algorithm. The comparative results are listed in table III. As can be observed, our IG algorithm outperforms the other two algorithms on all the three scales of instances, given the same CPU time of $2n$ milliseconds. Moreover, the advantage of IG is enlarging with the increase of the number of the points of instances. For small scale instances, the ARPDs of TS and GA are 1.71% and 1.08% larger than IG, while for large scale instances, the differences increase to 12.83% and 12.88%, respectively. This is because the solution space will enlarge in an exponential way with

TABLE II
THE PERFORMANCE OF INITIALIZATION METHODS ON DIFFERENT INSTANCE SETS

Instance Set	Initialization	TC	TD	DIS	ARPD (%)	CPU Elapse (ms)
Small	REPT	5.92	2.60	3.32	16.78	0.10
	RETA	5.51	2.29	3.22	**5.41**	0.12
	RU	5.60	2.38	3.22	**5.57**	0.10
	RH	5.97	2.61	3.36	15.13	0.11
Medium	REPT	8.29	4.47	3.81	22.59	0.20
	RETA	7.36	3.80	3.56	**8.53**	0.19
	RU	7.54	4.02	3.52	**9.49**	0.22
	RH	8.24	4.50	3.74	21.89	0.20
Large	REPT	18.67	12.36	6.30	32.71	1.20
	RETA	16.34	10.61	5.73	**14.41**	1.28
	RU	16.59	10.75	5.84	**17.36**	1.12
	RH	18.50	12.22	6.27	30.03	1.20

TABLE III
THE PERFORMANCE OF IG, TS AND GA WITH $2n$ MS

Instance Set	Algorithm	TC	TD	DIS	ARPD (%)	1% Worst
Small	IG	**4.83**	1.68	3.15	**0.47**	**28.08**
	TS	4.98	1.81	3.17	2.18	30.48
	GA	4.90	1.72	3.18	1.55	29.24
Medium	IG	**5.88**	2.45	3.43	**0.52**	**35.08**
	TS	6.23	2.78	3.45	5.10	37.56
	GA	6.04	2.60	3.44	2.96	36.11
Large	IG	**12.53**	7.28	5.25	**2.38**	**93.43**
	TS	13.75	8.29	5.47	15.21	97.54
	GA	13.73	8.25	5.48	15.26	97.68

the increase of problem scale in combinatorial problems. However, the ARPD of IG is slightly changed compared with the drastic increase of TS and GA, which proves the excellent performance of IG under different problem scales. Furthermore, we calculate the average total cost of the worst 1% solutions for every algorithm, which is shown in the column named '1% Worst', to show how bad an algorithm can be under some extreme instances. The result shows that IG still provides satisfying solutions under those worst cases. Note that GA has similar performance as IG in small scale instances, but keeps worsening as the problem scale grows. This is possibly because very little computational time is provided for GA to evolve and converge.

Moreover, it is of our interest to shorten or lengthen the termination criterion, to fully validate the superiority of the proposed IG algorithm. On the one hand, we shorten the termination criterion to 10 ms to see how algorithms will perform under extremely little computational time. On the other hand, we lengthen the termination criterion to 100 ms to fully understand the capability of algorithms because evolutionary algorithms normally need a relatively long time to converge to a near-optimal solution. The results are shown in table IV and V, respectively. From Table IV, we can see that the performance of three algorithms all becomes worse in this situation but IG still dominates the other two. Note that for large scale instances, the performance of GA declines very fast if little computational time is provided, while IG and TS are barely influenced. Additionally, as can be seen from table V, GA benefits the most from longer computational time, resulting in great improvement of the quality of solutions, although still worse than IG. This indicates that our IG algorithm is robust enough to be used under different situations. GA is probably more suitable for offline problems and TS requires better design of operators to improve its performance. With the analysis above, we conclude that our proposed IG algorithm is more suitable and effective for solving the ORPP, compared with other evolutionary algorithms.

V. CONCLUSION AND FUTURE WORKS

In this article, we design an iterated greedy (IG) algorithm for the online route planning problem (ORPP). Firstly, four initialization methods are proposed with different order sequencing rules, among which the best two rules are selected to be used for generating the initial solution of our algorithm. Secondly, the order-based destruction operators and the greedy construction operators are designed according to the characteristics of ORPP. Moreover, two problem-specific adjustment operators and two neighborhood search operators are designed for the local search procedure. Experimental results on 1500 instances from real world show that our IG algorithm outperforms the compared evolutionary algorithms, suggesting that IG is more suitable and efficient for solving the ORPP. Further experiments have also proved that IG is robust enough to provide high-quality solutions under different settings of termination criterion.

Our future work will mainly focus on two aspects, which are the extension of problems and approaches, respectively. On the extension of problems, it will be interesting to extend the single vehicle situation to the multiple case, which will

TABLE IV
THE PERFORMANCE OF IG, TS AND GA WITH 10 MS

Instance Set	Algorithm	TC	TD	DIS	ARPD (%)	1% Worst
Small	IG	**4.85**	1.71	3.14	**0.53**	**29.34**
	TS	4.98	1.81	3.17	2.18	30.48
	GA	4.92	1.77	3.15	1.88	30.01
Medium	IG	**5.92**	2.50	3.42	**0.92**	**35.35**
	TS	6.23	2.78	3.45	5.08	37.56
	GA	6.10	2.64	3.46	3.16	36.74
Large	IG	**12.83**	7.54	5.30	**3.01**	**95.67**
	TS	13.84	8.37	5.47	14.95	97.54
	GA	15.18	9.51	5.67	31.82	100.61

TABLE V
THE PERFORMANCE OF IG, TS AND GA WITH 100 MS

Instance Set	Algorithm	TC	TD	DIS	ARPD (%)	1% Worst
Small	IG	**4.82**	1.67	3.15	**0.45**	**28.01**
	TS	4.98	1.81	3.17	2.18	30.46
	GA	4.87	1.70	3.17	1.39	29.23
Medium	IG	**5.87**	2.44	3.43	**0.36**	**34.15**
	TS	6.23	2.78	3.45	5.10	37.55
	GA	5.98	2.54	3.44	2.43	35.23
Large	IG	**12.38**	7.15	5.23	**2.34**	**92.22**
	TS	13.75	8.29	5.47	15.57	97.54
	GA	12.74	7.42	5.32	5.64	95.62

complicate the problem with the complexity of order assignment. Moreover, considering more realistic constraints, such as introducing the uncertainty and dynamism, is also promising [18]. On the extension of approaches, we will further investigate other methods, including heuristics, meta-heuristics, or emerging methods such as reinforcement learning, attempting to bring some insights for the methodology of online route planning problem.

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, Jan.) Meituan to invest rmb11 billion to support merchant development. [Online]. Available: https://www.prnewswire.com/news-releases/meituan-to-invest-rmb11-billion-to-support-merchant-development-300782802.html

[2] M. Stanley. (2017, Jun.) Is online food delivery about to get 'amazoned'? [Online]. Available: https://www.morganstanley.com/ideas/online-food-delivery-market-expands/

[3] Maze. (2016, Jul.) Not everything delivers: Saying no to delivery. [Online]. Available: http://nrn.com/operations/not-everything-delivers-saying-no-delivery/

[4] 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.

[5] Ropke, Stefan and Cordeau, Jean-François, "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] S. Ropke, J.-F. Cordeau, and G. Laporte, "Models and branch-and-cut algorithms for pickup and delivery problems with time windows," *Networks: An International Journal*, vol. 49, no. 4, pp. 258–272, 2007.

[7] L. Van der Bruggen, J. K. Lenstra, and P. Schuur, "Variable-depth search for the single-vehicle pickup and delivery problem with time windows," *Transportation Science*, vol. 27, no. 3, pp. 298–311, 1993.

[8] W.-R. Jih and J. Y.-J. Hsu, "Dynamic vehicle routing using hybrid genetic algorithms," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, vol. 1. IEEE, 1999, pp. 453–458.

[9] A. Landrieu, Y. Mati, and Z. Binder, "A tabu search heuristic for the single vehicle pickup and delivery problem with time windows," *Journal of Intelligent Manufacturing*, vol. 12, no. 5-6, pp. 497–508, 2001.

[10] M. I. Hosny and C. L. Mumford, "Single vehicle pickup and delivery with time windows: made to measure genetic encoding and operators," in *Proceedings of the 9th annual conference companion on Genetic and evolutionary computation*. ACM, 2007, pp. 2489–2496.

[11] M. I. Hosny and C. L. Mumford, "The single vehicle pickup and delivery problem with time windows: intelligent operators for heuristic and metaheuristic algorithms," *Journal of Heuristics*, vol. 16, no. 3, pp. 417–439, 2010.

[12] K. Braekers, K. Ramaekers, and I. Van Nieuwenhuyse, "The vehicle routing problem: State of the art classification and review," *Computers & Industrial Engineering*, vol. 99, pp. 300–313, 2016.

[13] R. Ruiz and T. Stützle, "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *European Journal of Operational Research*, vol. 177, no. 3, pp. 2033–2049, 2007.

[14] R. Ruiz and T. Stützle, "An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives," *European Journal of Operational Research*, vol. 187, no. 3, pp. 1143–1159, 2008.

[15] M. Pranzo and D. Pacciarelli, "An iterated greedy metaheuristic for the blocking job shop scheduling problem," *Journal of Heuristics*, vol. 22, no. 4, pp. 587–611, 2016.

[16] R. Ruiz, Q.-K. Pan, and B. Naderi, "Iterated greedy methods for the distributed permutation flowshop scheduling problem," *Omega*, vol. 83, pp. 213–222, 2019.

[17] K. Karabulut and M. F. Tasgetiren, "A variable iterated greedy algorithm for the traveling salesman problem with time windows," *Information Sciences*, vol. 279, pp. 383–395, 2014.

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