Two Fast Heuristics for Online Order Dispatching

Qingte Zhou, Huanyu Zheng, Shengyao Wang,
Jinghua Hao, Renqing He, Zhizhao Sun
Meituan-Dianping Group
Beijing, China
{zhouqingte, zhenghuanyu, wangshengyao,
haojinghua, herenqing, sunzhizhao}@meituan.com

Xing Wang, Ling Wang
Department of Automation
Tsinghua University
Beijing, China
wang-x17@mails.tsinghua.edu.cn,
wangling@mail.tsinghua.edu.cn

***Abstract*—Order dispatching, a key part in real-time food delivery system, has received a lot of attention over the last decade. Under constraints of on-time rate and some other practical constraints, our goal is to maximize the total efficiency of the whole system. As an online algorithm, the running time of the algorithm for order dispatching is limited within milliseconds. At the same time, the problem is highly dynamic and the complexity of the solution space is huge. In this paper, we design two fast heuristics for order dispatching in real-time food delivery. We compare our algorithm with two state-of-the-art algorithms. With numerical results, we show our algorithms are faster than algorithms in the literature and with similar or better solution quality.**

***Index Terms*—Real-time food delivery, Order dispatching, Heuristics, Real-world applications**

I. Introduction

A. Background

Food delivery service is a fast growing market all over the world. Take Meituan-Dianping, a Chinese food ordering platform for example. According to 3rd quarter financial statement [1], the revenue was $2.2 billion dollars, with a growth rate of 39.4%. Moreover, there were 2.5 billion transactions, with a growth rate of 38.1%. Under intense competition, designing an efficient and effective on-time delivery system is the key challenge. One of core algorithms in this system is order dispatching. We assign orders to the most efficient driver so as to maximize the overall efficiency of the delivery process.

There are three key stakeholders in food delivery, namely, customers, restaurants and drivers. Customers require a fast service. Restaurants require specialized and stable service. Drivers provide delivery service, and require good delivery experience. For example, the dispatched orders at a time should be in the same direction. Moreover, given hours spent on delivering food, drivers want to maximize their income, in other words, the number of orders they finish. The platform has to dispatch orders considering requirements from all three parties and maximize the efficiency of drivers.

With these challenges in mind, our paper is motivated by an order dispatching system, in which a group of drivers are specialized to a restaurant. The dispatching process is as Fig. 1. A customer orders food from our platform, then our system pushes the order to the restaurant. At the same time, we assign this order to one of our drivers. Note that we also have common drivers who serve for all restaurants, who can be

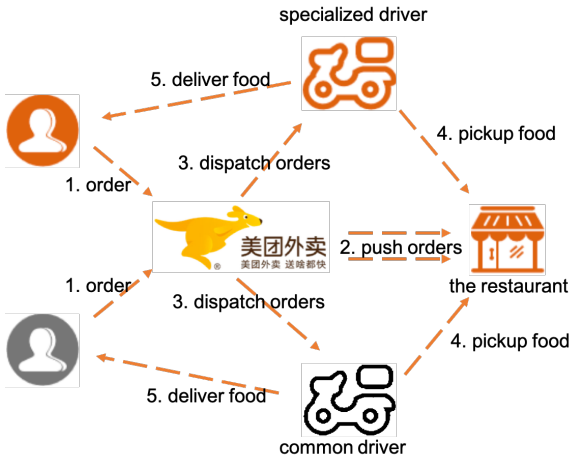


Fig. 1. The order dispatching process.

assigned orders from this restaurant as well. Therefore, there is a two-stage process deciding either assigning the order to a specialized driver or a common driver first, and then finding the best driver to assign. In our setting, specialized drivers are cheaper than common drivers, thus it is preferable to assign the order to a specialized driver. Our purpose is to maximize the percentage of orders that assigned to specialized drivers, under the constraints of serving customers on-time, and some other practical constraints. For example, we have to dispatch the same direction orders at a time. Moreover, drivers have to finish all delivery tasks before pickup tasks next round.

The major difficulty for this problem are two folds. One is that the problem is highly dynamic. Orders are not known before the minute of dispatching. The other one is that the solution space is growing fast with problem scale.

Unlike traditional vehicle routing problems or pickup and delivery problems, orders in our setting are not known beforehand. In this problem, orders are generated in real time. The decision of dispatching orders to drivers now will affect future driver behaviors. Therefore, we have to solve the order dispatching problem dynamically. In this paper, we reschedule drivers' routes in each minute and make decisions based on these routes.

Another difficulty is that the solution space is exponentially

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

growing with the number of orders and drivers. We have tens of orders for each restaurants each minute at peak hours. And we also have tens of specialized drivers for a restaurant as well as hundreds of common drivers in the same area. To find an optimal solution is super time consuming. We have to balance between computation time and optimality. As an online algorithm, order dispatching has to be done within milliseconds to meet the need of fast delivery. However, a fast heuristic is usually near sighted, with the loss of optimality. In this paper, we aim to find fast heuristics with better performance and shorter computation time than the state-of-the-art heuristics.

B. Literature Review

As far as we concerned, the most relevant problem to our setting is the pickup and delivery problem with time windows (PDPTW), which is an extension of vehicle routing problem with time windows (VRPTW). In the past few decades, extensive research has been carried on and a large number of approaches have been proposed for traditional PDPTW. They can be roughly classified into three categories: exact algorithms, meta-heuristics-based approaches, and construction heuristics. Generally, exact algorithms [2], [3] establish the integer programming formulation of the problem and find optimal solutions via branch-and-bound-based algorithms. Exact algorithms are usually restricted to small-scale PDPTWs because the computation cost increases rapidly as the problem size grows.

Considering the capability of providing good-quality solutions for larger-scale instances in limited time, researchers turn their attention to heuristics and meta-heuristics. Heuristics are often fast and easy to implement. Lu and Dessouky [4] proposed an insertion-based heuristic to solve the multi-driver pickup and delivery problem with time windows. During the insertion, they considered the insertion cost that not only includes the classical incremental distance but also the cost of the reduction of the time window slack due to the insertions. Hosny and Mumford [5] developed a sequential construction heuristic (SEQ) which tries to gradually modify the current route until no further improvement is possible, and then the feasibility of current route is checked. The procedure ended when all requests have been inserted into the routes. They concluded that the proposed SEQ proves to be the most favorable solution construction method compared to some adapted parallel construction algorithms [6]. Qu and Curtois [7] proposed two iterative heuristic algorithms for the job insertion problem, which is a special case of PDPTW. They developed two methods to select the next job to insert, the first one using a greedy heuristic and the second using a regret heuristic. During insertion, they occasionally choose the second, third or fourth best options instead of simply choosing the best option. They conclude that the greedy heuristic is favorable according to computational results.

Compared with heuristics, meta-heuristics require more computation time but the ability of giving a high quality solution is significantly enhanced. Nanry and Barnes [8] are among the first to present a meta-heuristic for the PDPTW. Their approach is based on a reactive tabu search algorithm that combines several standard neighborhoods. Pankratz [9] proposes a grouping genetic algorithm (GGA) for solving the PDPTW which features a group-oriented genetic encoding in which each gene represents a group of requests instead of a single request. Ropke and Pisinger [10] extended the large neighborhood search (LNS) previously suggested for VRPTW [11] and present an adaptive LNS, which is composed of several removal and insertion heuristics that are used with a frequency corresponding to their historic performance. Nagata and Kobayashi [12] extended a guided ejection search (GES) algorithm to solve the PDPTW. They showed that the perturbation procedure has a great impact on both the solution quality and computation time to which they gave a careful attention. Other hybrid methods [13]–[15] also gives good performance, trying to combine the merits of different meta-heuristics.

For the dynamic version of PDPTW, The standard solution methodology is the use of a rolling time horizon proposed by Psaraftis [16]. Mitrovic-Minic et al. [17] introduces the concept of double-horizon based heuristics for the dynamic PDPTW. Such heuristics may be applied to both routing and scheduling subproblems. Extensive computational experiments have shown that the use of a double-horizon can yield gain in route costs when compared with classical (single) rolling horizon methods. Anker and Hmmerle [18] proposed an optimization algorithm called KoTH-ACO based on the MAX-MIN Ant System. Their KoTH algorithm shows faster convergence and better solution qualities than the MAX-MIN Ant System in benchmark instances. Gianpaolo Ghiani et al. [19] presented a simple but effective policy that every time a new request occurs, allocates it with the goal of minimizing the overall customers inconvenience. The basic idea is to reserve a fraction of the vehicles to service the requests qualified for being satisfied as soon as possible. Guo et al. [20] proposed a two-phase method for robust dynamic multi-objective vehicle routing problem. The experiments show that the proposed method can find more robust routes with less computation cost. A comprehensive review of dynamic PDPTW can be found in [21].

However, the exact algorithms and meta-heuristics-based approaches mentioned above are not suitable for real-time order dispatching due to the following reasons.

i) Online order dispatching problem requires that existing order sequence on every driver can not change when inserting the new order into original route.

ii) New orders to be inserted should not violate capacity, on time, same direction, and delivery before pickup constraints.

iii) The computation time of the algorithm is supposed to be millisecond level because of the scenario where the algorithm is executed.

To the best of our knowledge, there is no work addressing the same problem as ours. In this paper we describe real-time order dispatching problem in detail, with some real-life complexities. We also develop two fast heuristics to provide

a solution with good quality, which is designed by modifying the classical greedy and regret heuristics.

The remainder of this paper is organized as follows. Section II describes online order dispatching problem and elaborate the constraints. The proposed algorithm is then presented in Section III, including modified greedy insertion heuristic and modified regret insertion heuristic. Computational results are reported in Section IV. Section V gives a discussion about the relationship between our problem and evolutionary algorithms. Section VI concludes the paper with analysis and future research.

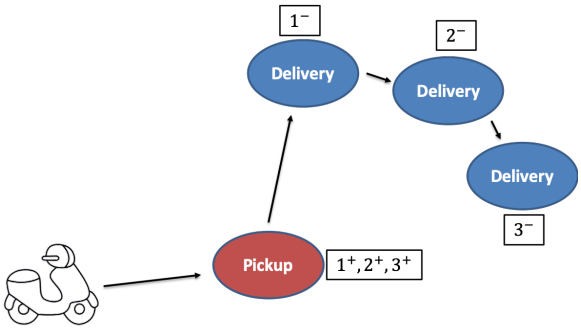
II. Problem Description

This section describes real-time order dispatching problem by elaborating constraints and objective function in detail. In real-world applications, the new orders generate continuously. We gather the new orders generated from last minute with the consideration of order urgency and computation complexity.

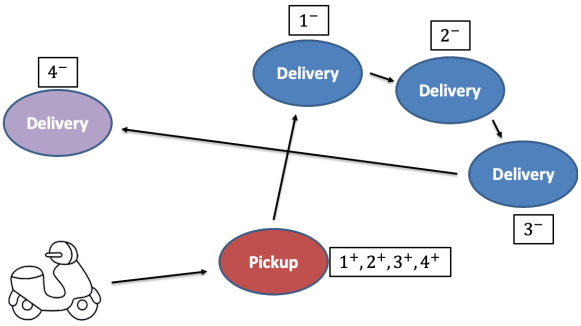
For every minute, the order dispatching problem contains n new orders and m drivers. Let $W = \{1, 2, \ldots, n\}$ be the set of new orders to be dispatched. Each order i has a pickup node and a delivery node, which are denoted as $\{i^+, i^-\}$ respectively. Denote the set of pickup nodes as $V^+ = \{i^+ | i \in W\}$ and the corresponding delivery nodes as $V^- = \{i^- | i \in W\}$. $V = V^+ \bigcup V^-$ represents the nodes that are associated to the new orders. Let $R = \{1, 2, \ldots . m\}$ be the set of drivers. The starting position of driver j is denoted as j^+, and $H = \{j^+ | j \in R\}$ is the set of starting positions of all drivers. Moreover, there is a sequence of dispatched orders, either picked-up already or not. Say that driver j is now working on several orders, which are denoted as Ω_j, and $\Omega = \{\Omega_j | j \in R\}$. Similarly, the set of unvisited pickup and delivery nodes of dispatched orders is denoted as $U = U^+ \bigcup U^-$, where the number of pickup and delivery nodes may not be identical.

Each order $i \in W \bigcup \Omega$ has an earliest pickup time TP_i. A driver cannot pick up the food of order i before TP_i. There is also an estimated time of arrival ETA_i, which is a promise to customers when the order is generated. For every possible node $p, q \in V \bigcup H \bigcup U$, the travel distance is denoted as d_{pq}, the travel time is denoted as t_{pq}, the visiting time of node p is D_p. A heterogeneous fleet of drivers starts from their current position, moving according to the route assigned to them. Every driver has a maximum capacity Q, which limits the maximum quantity of goods the driver can carry at a certain time.

A feasible solution to online order dispatching problem is a feasible matching from orders to drivers. Then we can form a set of feasible routes obtained by inserting the new orders into the original routes of drivers. The sequence of dispatched order nodes that the driver is executing should not be changed. For each assigned order, the pickup and delivery nodes have to be visited exactly once by the same driver. The pickup node has to be visited before the corresponding delivery node. In addition, each feasible route is constructed without violating the following constraints.

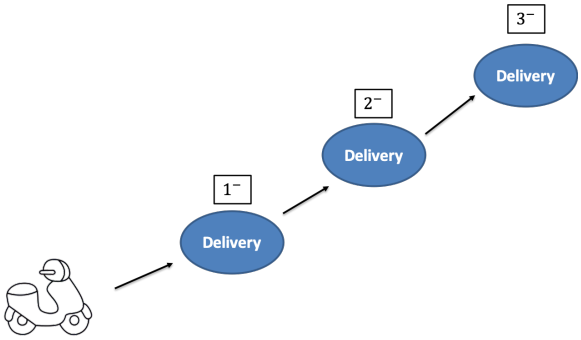


(a) The original route

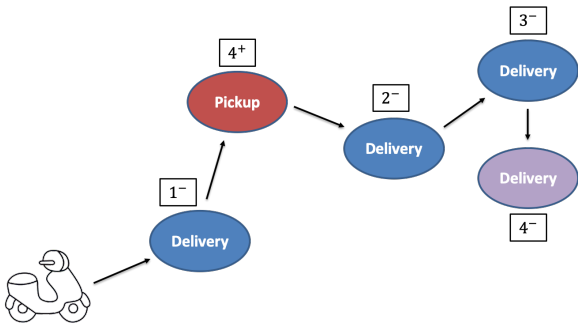


(b) An infeasible new route

Fig. 2. Explanation of the same direction constraint.



(a) The original route



(b) An infeasible new route

Fig. 3. Explanation of the delivery before pickup constraint.

i) On time: Define the overtime OT_i of order i as follows:

$$OT_i = max(0, D_{i^-} - ETA_i) \tag{1}$$

If order i is assigned to driver j and the corresponding pickup node is inserted at position k_i in the route of driver, then the overtime of all orders after position k_i has to be 0.

ii) Same direction: If the new order is near enough, we do not consider this constraint. Otherwise we define set Φ, which is composed of delivery nodes of driver's original route. Note that there may be delivery nodes too close to the restaurant, we delete them from set Φ. Each delivery node i in set Φ forms a vector, say $\vec{i}$ for example, starts from the restaurant and ends at the delivery node. When there is a new order inserting into the route, we calculate $n\vec{e}w$, which also starts from the restaurant and ends at the delivery node. The constraint says that each angle composed by$\vec{i}$ and $n\vec{e}w$ has to be less than a certain angle. Fig. 2 explains the constraint with a graphic example. Fig. 2(a) shows the original route, in which orders 1, 2, 3 are picked up first and delivered one after another. Fig. 2(b) shows the new route where the pickup node and delivery node of new order is inserted into the start and the end of the original route, respectively, which violates this constraint.

iii) Delivery before pickup: If the new order is near enough, we do not consider this constraint. Otherwise, all orders a driver has already picked up should be delivered before picking up a new order. Fig. 3 shows the situation where the constraint is violated. The original route of the driver is to finish deliveries of order 1, 2, 3 sequentially as shown in Fig. 3(a). If the pickup node of a new order 4 is inserted in a position like Fig. 3(b), then the constraint is not satisfied, which means that the driver has to go back to the restaurant again before finishing delivery tasks already assigned.

The objective is to maximize the order dispatched rate (ODR), which is computed by (2):

$$ODR = \frac{DN}{TN} \tag{2}$$

where DN represents the number of new orders dispatched to specialized drivers, and TN represents the number of all new orders. The problem is to dispatch as many new orders to specialized drivers as possible, given constraints mentioned above.

III. Proposed Approach

The online order dispatching is a highly dynamic problem. Every minute, we do the following rescheduling process shown in Fig. 4.

We collect the information of new orders and drivers from the last minute as algorithm input. Then we calculate the dispatch results as output. Finally we push the new orders to corresponding drivers and notice them their new routes.

Two fast heuristics namely modified greedy insertion (MGI) heuristic and modified regret insertion (MRI) heuristic are proposed for the problem described in Section II. These heuristics are based on greedy insertion (GI) heuristic and regret insertion (RI) heuristic [10], respectively. GI and RI are

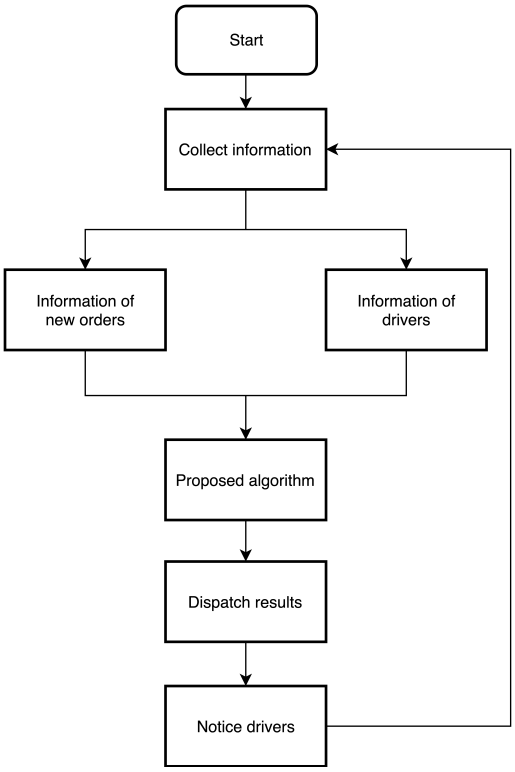


Fig. 4. Online order dispatching process.

chosen as state-of-the-art route construction heuristics because they are effective and efficient algorithms, widely used in the literature. The main difference between our MGI and MRI is the way they rank orders when inserting orders to the route. Therefore, we can describe the main process of both algorithms with the following pseudo-codes, which are shown in Algorithm 1.

The inputs are the set of unassigned new orders W and the set of drivers R. For each driver r, an initial route is given according to the sequence of dispatched order nodes that the driver is executing. Let $\Delta f_{w,r}$ denote the lowest incremental insertion cost of inserting order w into route of driver r. The metric of insertion cost is the total time consumption, including travel time and waiting time. Set $\Delta f_{w,r} = \infty$ if order w cannot be inserted into route of driver r. If an order w cannot be inserted into any position of any route due to the violation of constraints described in Section II, we remove it from W. For a given order w, we define $c_w = \min_{r \in R}\{\Delta f_{w,r}\}$, which is the cost of inserting order w at its best position overall. This position is denoted as the minimum cost position.

As for the ranking of unassigned orders, two different criteria are proposed, which will be introduced in subsections III-A and III-B. According to the sequence of sorted order list W_s, we insert the order into its minimum cost position one by one until meeting driver r who has been assigned order in the same iteration. We denote AR as the set of drivers who are assigned orders in each iteration. Then we need to update the

Algorithm 1 Pseudo-codes of MGI and MRI

Require: the set of unassigned orders W and the set of drivers R with original routes;

Calculate $\Delta f_{w,r}$ for each order $w \in W$ and driver $r \in R$ pair;
while $W! = \emptyset$ **do**
  for w in W **do**
    if there is no feasible position for w to insert **then**
      $W = W \backslash w$;
    end if
  end for
  Rank the orders W according to modified greedy criterion or modified regret criterion and get sorted order list W_s;
  Initialize the set of assigned orders $AO = \emptyset$ and the set of assigned drivers $AR = \emptyset$;
  for w in W_s **do**
    Find the driver r^* with lowest cost c_w;
    if $r^* \notin AR$ **then**
      Insert order w into its minimum cost position;
      $AO = \{AO, w\}$;
      $AR = \{AR, r^*\}$;
    else
      break;
    end if
  end for
  $W = W \backslash AO$;
  if $W! = \emptyset$ **then**
    Update $\Delta f_{w,r}$ for each order $w \in W$ and driver $r \in AR$ pair;
  end if
end while

related insertion cost $\Delta f_{w,r}$ for each unassigned order $w \in W$ and driver $r \in AR$ pair. This process continues until all orders have been assigned or no more orders can be inserted into any feasible position.

Observe that in each iteration, instead of dispatching only one order like GI and RI, we dispatch at least one order and usually many orders. This means that our proposed heuristics may use less iterations and less times of insertion cost recalculation. Therefore, we expect our MGI and MRI to be faster heuristics.

A. Modified Greedy Criterion

The greedy insertion heuristic executes a feasible order insertion into a driver's route with the lowest c_w at each iteration. Since the objective of the problem is to maximize order dispatched rate, the greedy insertion heuristic may be myopic and lead to a low quality solution.

We try to improve the performance of greedy insertion heuristic by incorporating information from the number of feasible drivers. We first rank all the orders by its feasible drivers number, the smaller the better. When comparing two orders with the same number of feasible drivers, the one with a lower c_w is better. Here, the number of feasible drivers is used to evaluate an order's current feasibility, which provides a look-ahead guidance. An order with lower feasibility needs to be assigned with higher priority.

B. Modified Regret Criterion

The regret insertion heuristic improves the greedy insertion heuristic through the use of a look-ahead strategy. For each order, let $x_{w,i} \in \{1, \ldots, R\}$ be a variable indicating the driver for which order w has the ith lowest insertion cost. Then, the regret value of order w can be defined as:

$$RV_w = \sum_{i=2}^{k} \left(\Delta f_{w,x_{w,i}} - \Delta f_{w,x_{w,1}} \right) \tag{3}$$

The regret insertion heuristic selects the order with maximum regret value and insert it into its minimum cost position. However, the influence of $\Delta f_{w,x_{w,i}} - \Delta f_{w,x_{w,1}}$ with different i can be different and should not be weighted equally. For example, the difference between the cost of the 2nd driver and the 1st driver can be weighted much higher than the difference between the cost of the 10th driver and the 1st driver. Because an order is much less likely to be assigned to its 10th driver than 2nd driver.

In our proposed MRI, we use a decay rate $\gamma \in (0, 1)$ to control the weight of $\Delta f_{w,x_{w,i}} - \Delta f_{w,x_{w,1}}$ for ith driver. And the regret value of order w in MRI can be defined as:

$$RV_w^* = \sum_{i=2}^{k} \left(\Delta f_{w,x_{w,i}} - \Delta f_{w,x_{w,1}} \right) \gamma^{i-2} \tag{4}$$

For order with only one feasible rider to assign, we set $RV_w^* = -\Delta f_{w,x_{w,1}}$.

The parameter k controls the number of drivers that we need to consider in regret insertion heuristic. In our modified regret criterion, we determine the parameter k according to the number of feasible drivers. We also first rank all the orders by its number of feasible drivers, the smaller the better. When comparing two orders with the same number of feasible drivers, the one with higher RV_w^* is better. Generally speaking, we choose the insertion with least feasibility and most regret value if it is not done now. Under this criterion, we not only maintain the ability of looking ahead, but also reduce some redundant calculation.

IV. Experimental Results

In this section, numerical experiments are provided to test the performance of proposed heuristics. First, we give an overall comparison of the four heuristics. Then we compare MGI with GI and MRI with RI in terms of solution quality and computation time, respectively. 20 instances are randomly sampled from real order dispatching problems. We partition the instance set into two sets, '21 to 30', and '31 to 40', according to the number of new orders. The decay rate γ of MRI is set as 0.5.

All experiments are run on a MacBook Pro with 2.2 GHz processors / 16 GB RAM under Mac OS. All the algorithms are coded in Java SE8 using Intellij IDEA.

A. Overall Comparison of Four Heuristics

In this subsection, we give an overall comparison of four heuristics, including MGI, MRI, GI and RI.

Table I demonstrates the average performance of four heuristics on 20 instances. From Table I, we can see that the order dispatched rates (ODR) of MGI, MRI and RI are quite close, which are significantly higher than GI. This means that proposed MGI and MRI obtain very near solutions with RI for all test instances and outperform GI. As for computation time, two proposed heuristics are faster than their original version. The average computation time of MGI and GI is 101.28ms and 115.17ms, which is 12.06% faster. Meanwhile, the average computation time of MRI and RI is 111.11ms and 140.69ms, which is 21.02% faster. From the above analysis, it can be confirmed that the effectivenesses of MRI and MGI is not harmed by our speeding up technique.

Table II shows the average performance of four heuristics on '21 to 30' instances set. It is seen that two proposed heuristics outperform original heuristics in both solution quality and computation time.

Table III compares the average performance of four heuristics on '31 to 40' instances set. We can find that MGI is better than GI in terms of order dispatched rate and time consumption. Compared with RI, the order dispatched rate of MRI is slightly lower but the computation speed is significantly faster.

From the comparison above, we show the effectiveness and efficiency of two proposed heuristics by comparing our results with RI and GI in the literature.

TABLE I
PERFORMANCE OF FOUR HEURISTICS ON ALL INSTANCES

Heuristic	Average ODR (%)	Average time (ms)
MGI	64.65	101.28
MRI	65.47	111.11
GI [10]	60.23	115.17
RI [10]	65.63	140.69

TABLE II
PERFORMANCE OF FOUR HEURISTICS ON INSTANCES WITH 21 TO 30 NEW ORDERS

Heuristic	Average ODR (%)	Average time (ms)
MGI	68.68	94.05
MRI	69.75	93.26
GI [10]	62.99	99.74
RI [10]	69.40	111.87

TABLE III
PERFORMANCE OF FOUR HEURISTICS ON INSTANCES WITH 31 TO 40 NEW ORDERS

Heuristic	Average ODR (%)	Average time (ms)
MGI	61.21	108.51
MRI	61.82	130.59
GI [10]	57.88	128.96
RI [10]	62.42	169.52

B. Comparison between MGI and GI

In this subsection, we compare the proposed MGI with GI instance by instance.

Fig. 5 shows the order dispatched rates of MGI and GI over 20 instances. It illustrates that order dispatched rates of MGI are higher than GI on almost all instances. GI often postpones the insertion of difficult orders, so that most of time GI cannot insert these orders to any feasible position as many of the routes are already full of orders. But with the information of the number of feasible drivers, MGI reduces the greediness of GI, which leads to a better solution in most of the time.

As for computation time, Fig. 6 indicates that MGI is faster than GI on 18 of 20 instances. Since MGI may insert more than one order in each iteration, the number of iterations is lower than GI on most instances, which leads to a shorter running time.

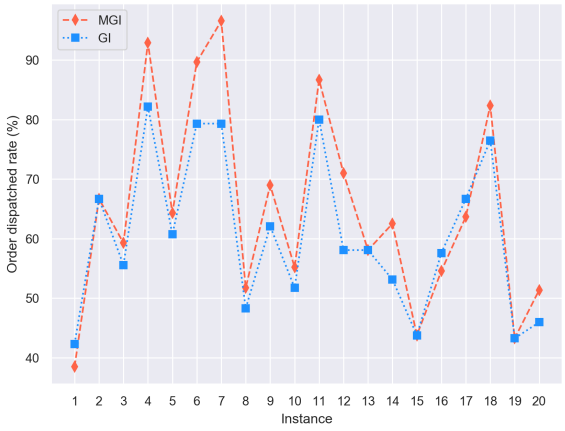


Fig. 5. Order dispatched rate comparison between MGI and GI.

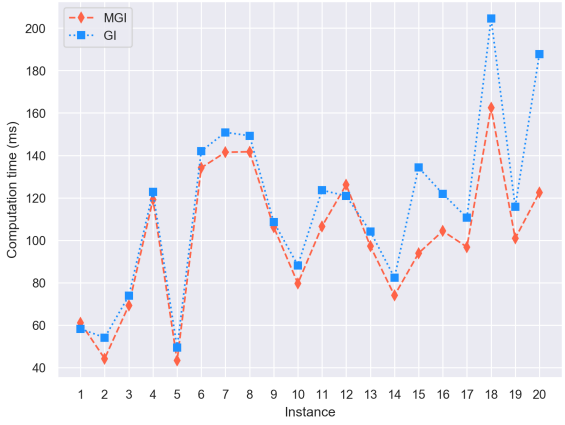


Fig. 6. Computation time comparison between MGI and GI.

C. Comparison between MRI and RI

In this subsection, an instance by instance comparison between MRI and RI is also illustrated.

From Fig. 7, we can see that the order dispatched rates of MRI are higher than RI on 6 of 20 instances and lower on 9 of 20. Indeed, the order dispatched rates are quite close over all instances, which indicates that MRI and RI share similar effectiveness on these instances.

As for efficiency, MRI is faster than RI over all the instances in Fig. 8. We can find that the advantage of MRI over RI on computation time increases on larger instances. With the consideration of the number of feasible drivers, some redundant calculation of RI can be eliminated. Since MRI may insert more than one order in each iteration, the computation time of MRI is also reduced.

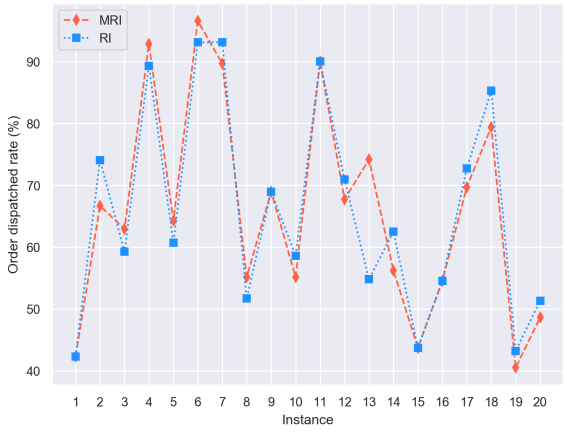


Fig. 7. Order dispatched rate comparison between MRI and RI.

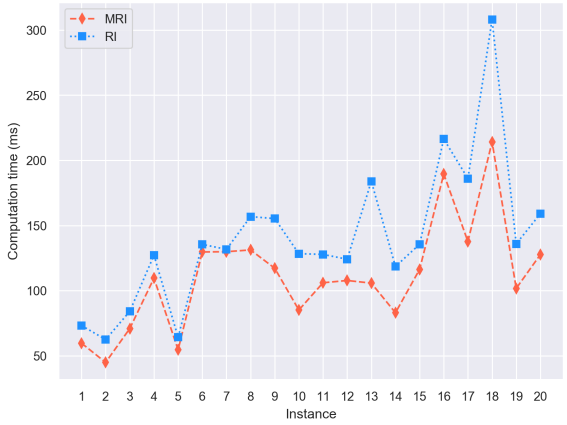


Fig. 8. Computation time comparison between MRI and RI.

V. Discussion

Evolutionary algorithms have shown to be powerful for global optimization of a wide range of problems both in research and real-world applications. The major reason for why we do not use evolutionary algorithms in this paper are two folds. One is that the computation time is supposed to be millisecond level, but evolutionary computation is relatively time consuming. The other one is that the problem we studied is a highly dynamic and highly constrained problem. The region of feasibility is very limited, compared to the classical PDPTW. When using evolutionary algorithms, it is hard to find a feasible solution using evolutionary operators.

The solutions derived by proposed heuristics still have some space to be improved. Evolutionary algorithms are good choices. When the computation time is not the bottleneck, we can start from initial solutions by proposed heuristics and improve the solutions by designing a proper evolutionary algorithm.

VI. Conclusion And Future Works

In this paper, we study online order dispatching problem considering some realistic constraints. We present two fast heuristics, namely modified greedy insertion (MGI) and modified regret insertion (MRI), based on greedy insertion (GI) heuristic and regret insertion (RI) heuristic, respectively. The heuristics are tested on a series of instances that are randomly sampled from real order dispatching problems. The results of these experiments show that the solution quality of MGI and MRI are consistently improved with respect to GI and are very close to RI. As for computation time, MGI is faster than GI and MRI is also faster than RI, which demonstrates the efficiency of proposed heuristics.

In the future, we will take more dynamic nature of online order dispatching problem into consideration. With the help of machine learning, we are able to make some predictions for the near future. Taking these predictions into the process of decision making may lead to a better performance from a period view.

Acknowledgements

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] (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] Q. Lu and M. Dessouky, "An exact algorithm for the multiple vehicle pickup and delivery problem," *Transportation Science*, vol. 38, no. 4, pp. 503–514, 2004.

[3] R. Baldacci, E. Bartolini, and A. Mingozzi, "An exact algorithm for the pickup and delivery problem with time windows," *Operations Research*, vol. 59, no. 2, pp. 414–426, 2011.

[4] 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.

[5] M. I. Hosny and C. L. Mumford, "New solution construction heuristics for the multiple vehicle pickup and delivery problem with time windows," in *MIC2009, Metaheuristic International Conference*, 2009.

[6] J.-Y. Potvin and J.-M. Rousseau, "A parallel route building algorithm for the vehicle routing and scheduling problem with time windows," *European Journal of Operational Research*, vol. 66, no. 3, pp. 331–340, 1993.

[7] Y. Qu and T. Curtois, "Job insertion for the pickup and delivery problem with time windows," *Lecture Notes in Management Science*, vol. 9, pp. 26–32, 2017.

[8] W. P. Nanry and J. W. Barnes, "Solving the pickup and delivery problem with time windows using reactive tabu search," *Transportation Research Part B: Methodological*, vol. 34, no. 2, pp. 107–121, 2000.

[9] 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.

[10] 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.

[11] P. Shaw, "A new local search algorithm providing high quality solutions to vehicle routing problems," *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*, 1997.

[12] Y. Nagata and S. Kobayashi, "Guided ejection search for the pickup and delivery problem with time windows," in *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer, 2010, pp. 202–213.

[13] S. Naccache, J.-F. Côté, and L. C. Coelho, "The multi-pickup and delivery problem with time windows," *European Journal of Operational Research*, vol. 269, no. 1, pp. 353–362, 2018.

[14] R. Bent and P. Van Hentenryck, "A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows," *Computers & Operations Research*, vol. 33, no. 4, pp. 875–893, 2006.

[15] C. S. Sartori and L. S. Buriol, "A matheuristic approach to the pickup and delivery problem with time windows," in *International Conference on Computational Logistics*. Springer, 2018, pp. 253–267.

[16] H. N. Psaraftis, "Dynamic vehicle routing problems," *Vehicle Routing: Methods and Studies*, vol. 16, pp. 223–248, 1988.

[17] S. Mitrovic-Minic, R. Krishnamurti, G. Laporte *et al.*, "Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows," *Transportation Research Part B: Methodological*, vol. 38, no. 8, pp. 669–685, 2004.

[18] M. Ankerl and A. Hämmerle, "Applying ant colony optimisation to dynamic pickup and delivery," in *International Conference on Computer Aided Systems Theory*. Springer, 2009, pp. 721–728.

[19] G. Ghiani, E. Manni, and A. Romano, "A dispatching policy for the dynamic and stochastic pickup and delivery problem," in *International Conference on Applied Physics, System Science and Computers*. Springer, 2017, pp. 303–309.

[20] 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.

[21] G. Berbeglia, J.-F. Cordeau, and G. Laporte, "Dynamic pickup and delivery problems," *European Journal of Operational Research*, vol. 202, no. 1, pp. 8–15, 2010.