# Order Dispatch in Price-aware Ridesharing

Libin Zheng
Hong Kong University of Science and Technology
Hong Kong, China
lzhengab@cst.ust.hk

Lei Chen
Hong Kong University of Science and Technology
Hong Kong, China
leichen@cse.ust.hk

Jieping Ye
Didi Research Institute, Didi Chuxing
Beijing, China
yejieping@didichuxing.com

## ABSTRACT

With the prevalence of car-hailing applications, ridesharing becomes more and more popular because of its great potential in monetary saving and environmental protection. Order dispatch is the key problem in ridesharing, which has a strong impact on riders' experience and platform's performance. Existing order dispatch research works fail to consider the price of the orders, which can be an important reference because it directly relates to the platform's profit. Our work takes the order price into concern, and formulates a constrained optimization problem, which takes platform's profit as the optimization objective and performs controls on riders' detour distance and waiting time. We prove the problem is NP-hard, thus, we propose approximation methods. We further develop a simulation framework based on real ridesharing order and vehicle data. We conduct experiments with this simulation framework to evaluate the effectiveness and efficiency of the proposed methods.

**PVLDB Reference Format:**
Libin Zheng, Lei Chen and Jieping Ye. Order Dispatch in Price-aware Ridesharing. *PVLDB*, 11 (8): 853-865, 2018.
DOI: https://doi.org/10.14778/3204028.3204030

## 1. INTRODUCTION

To utilize the empty seats in manned cars and reduce the monetary cost of riders, car-hailing service providers, such as Uber[1] and Didi Chuxing[2], have been promoting their ridesharing business in recent years. In contrast to traditional ride-hailing services where one vehicle takes at most one rider at a time, the service provider in ridesharing may assign multiple riding orders to a car at the same time. Ridesharing brings a lot of benefits. For riders, they get travel fee reductions from sharing the ride with others. For car-hailing service providers, ridesharing enables a better utilization of the limited vehicles, which means more profits. Ridesharing also relieves their serving stress during peak ordering hours. Besides, the government encourages such a shared transport for the purpose of reducing carbon dioxide emission. Due to its great significance, ridesharing has drawn increasing attention from the academia.

[1]https://www.uber.com
[2]http://www.xiaojukeji.com/


*Proceedings of the VLDB Endowment,* Vol. 11, No. 8

DOI: https://doi.org/10.14778/3204028.3204030

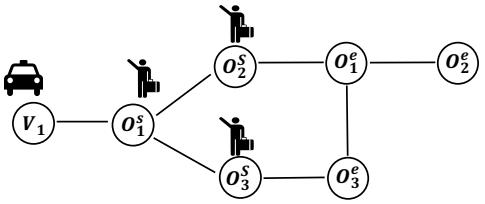



**Figure 1: Example 1.**

The dial-a-ride problem (DARP) [8, 10, 5, 16] has been widely studied in the past years, which is a variant of the vehicle routing problem [19] with time window constraints for pick-up and drop-off. The problem of adopting these methods lies in their setting that all orders are served. This is not practical in real-time ridesharing due to the limited transportation capacity. Besides, integer programming techniques are commonly used to solve the problem, which leads to great time costs. Generally only small instances are tested in their works [9]. There are also research works [1], which use a multi-objective function to trade-off the travel distance and the number of un-served orders. However, it is unclear on how to regularize the penalty term. Actually, all of these works fail to consider the order prices, which can be an important reference to distinguish the orders during the dispatch process.

Online algorithms have also been proposed to address the real-time ridesharing problem. [20] and [15] re-optimize the travel plans of vehicles on arrival of each new order. This online setting conducts optimizations on individual orders and give them immediate responses. However, they do not consider the order prices either.

In typical ridesharing platforms (Uber and Didi Chuxing), the price of a ridesharing order is decided after user's input of destination and before his/her drop of an order. Therefore, the prices of the ridesharing orders are fixed regardless of the dispatch and routing results. This is quite different from non-ridesharing orders whose price may depend on the actual travel length. Ignorance of the price during dispatch of the ridesharing orders may prevent the dispatcher (the server of ridesharing platforms) from finding the profit-optimal solution. For example, when the vehicles are in shortage, the dispatch algorithm decides to serve which orders. The order price would be one important reference as it is directly related to the platform's earning. Previous works on ridesharing order dispatch [10, 1, 20, 15] fail to distinguish the orders according to their prices.

**Example 1.** *In Figure 1, there are three orders $o_1$, $o_2$, $o_3$ and only one vehicle $v_1$. $o_1^s$, $o_2^s$, $o_3^s$ and $o_1^e$, $o_2^e$, $o_3^e$ represent the origins and*

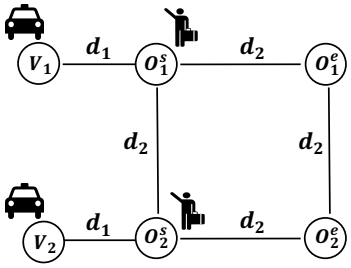



**Figure 2: Example 2.**

*destinations respectively. Circles and edges represent the nodes and paths respectively in the road network. In this example, all edges are of equal travel distance. This example corresponds to peak ordering periods where vehicles are in shortage. There are two possible travel plans for $v_1$: 1) $v_1 o_1^s o_2^s o_1^e o_2^e$; 2) $v_1 o_1^s o_3^s o_3^e o_1^e$. Without considering the order prices, the above two plans show no difference because they own the same amount of travel distance and the same number of served orders. Let us consider the order prices and assume $price(o_2) > price(o_3)$. This is reasonable as $d(o_2^s, o_2^e) > d(o_3^s, o_3^e)$. Note that $d(\cdot, \cdot)$ represents the shortest path distance, and has a strong impact on orders' price in practice. Then, the price-aware solution would be dispatching $o_1$ and $o_2$ to $v_1$.*

**Example 2.** *Figure 2 gives an example which corresponds to the scenario with enough transportation capability to serve the orders. There are two orders $o_1$ & $o_2$ and two vehicles $v_1$ & $v_2$. In this example, we assume $d(v_1, o_1^s) = d(v_2, o_2^s) = d_1 < d(o_1^s, o_2^s) = d(o_1^s, o_1^e) = d(o_2^s, o_2^e) = d(o_1^e, o_2^e) = d_2$. Without consideration of order prices, the dispatch solution with minimum travel distance is to dispatch $o_1$ and $o_2$ to $v_1$ and $v_2$ respectively. However, this may not lead to the profit-optimal solution. Let us take the order prices and vehicle payments into account. The profit of dispatching these two orders is equal to $price(o_1) + price(o_2) - pay(v_1) - pay(v_2)$. The payment to a vehicle is comprised of the base fare (flag-down fare) and the travel fare. The platform would pay one base fare less if employing $v_1$ only. Let the base fare be larger than $cost(d_2 - d_1)$, i.e., the travel cost saving from adopting $v_1 o_1^s o_1^e, v_2 o_2^s o_2^e$ instead of $v_1 o_1^s o_2^s o_2^e o_1^e$. Then, the optimal dispatch solution is to dispatch both $o_1$ and $o_2$ to $v_1$.*

In this work, the goal is to maximize the platform's profit. Given an order dispatch solution, its profit is equal to the sum of order prices minus the payment to the vehicles. Therefore, maximization of the profit naturally requires one to control the payment by taking care of the vehicles' travel distance, which is also the key objective in previous works [20, 15]. However, profit maximization needs to refer to the order price as well, which is ignored in those works. Following the industrial practice [30], we adopt the setting where the dispatcher operates on the batched orders periodically. The dispatcher packs the pended orders and dispatches them to the vehicles such that the profit is maximized. There are constraints to ensure the rider experience, including control of detour distance and pick-up distance. To the best of our knowledge, this is the first time that the order price has been considered in the dispatch process. The exponentially large solution space makes it difficult to solve the problem exactly. We prove that the problem is NP-hard, and thus we propose approximation methods. To evaluate their performance, we develop a simulation framework based on the real ridesharing order and vehicle data from Didi Chuxing. In summary, our contributions are listed as follows:

**Table 1: Symbols and notations**

| Symbol | Description |
|---|---|
| $O = \{o_i\}$ | Set of orders. |
| $V = \{v_j\}$ | Set of vehicles. |
| $o_i^s$, $o_i^e$ | The origin and the destination of order $o_i$. |
| $rev_i$ | The price of order $o_i$. |
| $cost_j$ | The payment to vehicle $v_j$. |
| $P$ | The overall profit. |
| $P_{i,j}$ | The profit of dispatching $o_i$ to $v_j$. |
| $t(a, b)$ | The travel time from $a$ to $b$. |
| $d(a, b)$ | The travel distance from $a$ to $b$. |
| $dr_i$ | The detour ratio of $o_i$. |
| $\theta_{dr}$ | The threshold of detour ratio. |
| $pt_i$ | The pick-up time of order $o_i$. |
| $\theta_{pt}$ | The threshold of pick-up time. |
| $plan_j$ | The travel plan of vehicle $v_j$. |
| $c_j$ | The capacity of vehicle $v_j$. |

- We formulate the problem of Maximizing Profit in Price-aware Ridesharing (MPPR) and prove the problem hardness in Section 2.
- We propose the greedy method and the matching-based order dispatch methods in Section 3 and Section 4 respectively.
- We devise some optimization techniques in Section 5, and devise a real-data based simulation framework in Section 6.
- We conduct experiments to investigate the performance of our proposed algorithms, and compare them to the state-of-art approach in section 7.

In addition to the contributions listed above, we discuss the related work in Section 8 and conclude the paper in Section 9. The important notations and symbols in the rest of this paper are summarized in Table 1 .

# 2. MAXIMIZING PROFIT IN PRICE-AWARE RIDESHARING

In this section, we first explain some concepts and describe the problem of profit maximization (Section 2.1). Then, we show the problem hardness by a reduction from the *traveling salesman problem* [4] (Section 2.2). Finally, we give some preliminary discussions in Section 2.3 before description of approximation methods.

## 2.1 Problem Formulation

**Definition 1** (Time epoch)**.** *A time epoch is a time instant when the dispatcher operates on the batched orders.*

The dispatcher assigns orders at a time epoch, after which it batches new orders until the next time epoch. In each time epoch, the input consists of the order set $O = \{o_i\}$ and the vehicle set $V = \{v_j\}$.

**Definition 2** (Order $o_i$)**.** *An order $o_i$ is a quaternary tuple $< o_i^s, o_i^e, t_i, rev_i >$, where $o_i^s$ and $o_i^e$ represent the starting (pick-up) location and ending (drop-off) location of the order respectively, $t_i$ is the time when $o_i$ is raised, and $rev_i$ is its price (the revenue of serving $o_i$).*

*Remark*: According to the handbook in the mobile application of Didi Chuxing, users pay an upfront fare to take the ridesharing service. Note that no further fare would be charged from users. This

upfront fare is computed by a pricing strategy, which considers the origin/destination, current demand-supply condition, and dynamic discounts (unavailable for non-sharing orders), etc. Therefore, the upfront fare is one of the inputs in the order dispatch process, and we denote them as the order price, $rev_i$.

**Definition 3** (Vehicle $v_j$)**.** *A vehicle $v_j$ is a tuple $< l_j, plan_j, c_j >$, where $l_j$ is the vehicle's current location, $plan_j$ is the travel plan, and $c_j$ is its capacity.*

$plan_j$ is the travel plan comprised by the pick-up and drop-off of the uncompleted orders. For example, in Figure 1, assuming $v_1$ is currently on the path between $o_1^s$ and $o_2^s$, the plan $v_1 o_2^s o_1^e o_2^e$ means that $v_1$ is on its way towards the pick-up of $o_2$, after which it would drop off $o_1$ and then $o_2$. At any time, the number of riders in the vehicle cannot exceed $c_j$.

The function $t(\cdot, \cdot)$ gives the travel time needed between two locations w.r.t. the shortest path. For example, the time needed to finish the aforementioned plan is $t(v_1, o_2^s) + t(o_2^s, o_1^e) + t(o_1^e, o_2^e)$. Based on $t(\cdot, \cdot)$, we define a detour ratio for each order.

**Definition 4** (Detour ratio $dr_{i,j}$)**.** *The detour ratio of $o_i$ w.r.t. $plan_j$, $dr_{i,j}$, is equal to $\frac{t(o_i^s, o_i^e)}{t_j(o_i^s, o_i^e)}$, where $t_j(o_i^s, o_i^e)$ is the time difference between pick-up and drop-off of $o_i$ in $plan_j$.*

For example, in Figure 1, let the plan of $v_1$ be $v_1 o_1^s o_3^s o_3^e o_1^e$. The shortest path to serve order $o_1$ is $o_1^s o_2^s o_1^e$. Assuming the edges are of equal distance, the detour ratio of $o_1$ w.r.t. this plan is:

$$\frac{t(o_1^s, o_3^s) + t(o_3^s, o_3^e) + t(o_3^e, o_1^e)}{t(o_1^s, o_2^s) + t(o_2^s, o_1^e)} = 1.5$$

Detour ratio measures the relative time loss due to ridesharing. Controlling the detour ratios prevents bad riding experiences.

In addition to the detour time, orders' waiting time during pick-up also has a strong impact on their experience.

**Definition 5** (Pick-up time $pt_i$)**.** *The pick-up time of $o_i$ is the time difference between when it is dispatched and when it is picked up.*

For example, suppose we dispatch $o_1$ and $o_3$ to $v_1$ with travel plan $v_1 o_1^s o_3^s o_3^e o_1^e$ in the current time epoch. Then, $pt_3$ is equal to $t(v_1, o_1^s) + t(o_1^s, o_3^s)$.

The profit of the platform is equal to the difference between the price of the dispatched orders and the payment to the drivers.

**Definition 6** (Driver payment)**.** *The payment to a vehicle is $bcost + cost_j$, i.e., the base fee plus the travel fee.*

The base fee, $bcost$, corresponds to the flag-down fee in taxi services. For the travel fee $cost_j$, without loss of generality, we assume that it is linear to its travel time:

$$cost_j = \alpha * t(plan_j)$$

where $t(plan_j)$ is the travel time of $plan_j$, from the first pick-up, before which $v_j$ is empty, to the last drop-off, after which $v_j$ is empty again.

Based on the above concepts, we define the platform profit, which is rarely considered in existing ridesharing works.

**Definition 7** (Platform profit)**.** *The platform profit of a dispatch solution is:*

$$P = \sum_{v_j \in V} \left( \left( \sum_{o_i \in plan_j} rev_i \right) - (bcost + cost_j) \right) \quad (1)$$

Our objective is to maximize $P$, which equals the price of dispatched orders minus the payment to vehicles. The order prices are fixed and known, but which orders would be dispatched are uncertain. The term $\sum_{o_i \in plan_j} rev_i$ is affected by the dispatch result and cannot be omitted. Our objective differs from those proposed by [8, 10, 5, 16] in that they simply minimize the travel distance assuming enough transportation capacity to serve all orders. In practice, not all orders can be dispatched timely. Some of them starve for a long time, and then are cancelled. According to [30], the order cancellation rate in Didi Chuxing is above $20\%$.

Following the industrial practice [30], we opt to optimize the profit locally in each round. This proceeds in a greedy manner, and does not guarantee a globally optimal solution with respect to the whole time span. However, because of the real-time nature of the problem, i.e., orders arrive dynamically and may be cancelled due to large delays, it is infeasible to solve the problem in a static manner. It is hard to optimize in the global scope unless a powerful order prediction technique is offered.

**Definition 8** (Maximizing profit in price-aware ridesharing, MPPR)**.** *Given a new order set $O$ and a vehicle set $V$ of a time epoch, the problem of maximizing profit in price-aware ridesharing is to dispatch the orders to the vehicles and update their travel plans, so that the profit (Equation* (1)*) is maximized, subjecting to the following constraints:*
*I . **Detour constraint.** For any dispatched order, the detour ratio is no larger than $\theta_{dr}$, i.e., $\forall o_i, dr_i \leq \theta_{dr}$;*
*II . **Pick-up constraint.** For any dispatched order, its pick-up time is no larger than $\theta_{pt}$, i.e., $\forall o_i, pt_i \leq \theta_{pt}$;*
*III . **Capacity constraint.** At any time, the number of passengers in a vehicle is no larger than its capacity, i.e., $\forall_t, |\{o_i \; in \; v_j\}| \leq c_j$.*

Constraint I is to protect the rider experience by controlling the detour ratios. Similarly, Constraint II prevents bothering both rider and driver with a large pick-up time. These two constraints correspond to the service constraint and waiting time constraint in [15] respectively.

## 2.2 Problem Hardness

In this section, we prove that the MPPR problem is NP-hard by a reduction from a variant of the *traveling salesman problem* (TSP) [4]. TSP without return (TSP-WR) differs from TSP in that it does not require the salesman to return to the depot. That is, the salesman visits each non-depot node once without the need for returning. TSP-WR is different from the *Hamiltonian path problem* [23], which does not specify depots. Due to the limited space, we only give proof sketches for the theoretical results in this section. Their formal proofs can be found in our technical report[3].

**Lemma 2.1.** *The traveling salesman problem without return (TSP-WR) is NP-hard.*

*Proof sketch.* Given a TSP instance, we reduce it to a TSP-WR instance by duplicating the depot node. We need to guarantee that this duplicate node is the last visited node in TSP-WR solutions. This can be realized by adding another node which is only accessible from the duplicate node. Then, the optimal TSP-WR solution is the one which visits all TSP nodes between the depot and its duplicate with the minimized travel distance. □

We now prove MPPR is NP-hard by a reduction from TSP-WR.

**Theorem 2.2.** *The problem of maximizing profit in price-aware ridesharing is NP-hard.*

[3]http://www.cse.ust.hk/%7Elzhengab/MPPRtech.pdf

*Proof sketch.* Given a TSP-WR instance, we construct a MPPR instance by mapping the depot to the current location of the only available vehicle and the origin of all orders. Besides, each non-depot node in TSP-WR is mapped to the destination of an order. Then, we set critical pick-up time requirements for orders so that they can only be picked up at the vehicle's departure. We need to guarantee that all orders are served in MPPR solutions. This can be realized by setting loose detour requirements, large order prices, and large vehicle capacity. Subsequently, the optimal MPPR solution is the one which visits all the nodes with the minimized travel distance. □

## 2.3 Preliminary of Approximation Methods

The MPPR problem is shown to be NP-hard with a reduction from TSP. However, existing approximation algorithms of TSP cannot be applied to MPPR due to the following reasons. Firstly, in TSP, all tasks are served and the goal is to minimize the overall travel distance. In contrast, in a dispatch round of real-time ridesharing, it is a common case that only part of orders can be served due to the limited transportation capacity. This is the reason why some orders can be immediately dispatched while others need to wait for some moments in reality. Secondly, the algorithms of TSP only target at routing, which is a subproblem in MPPR. MPPR also requires its solutions to dispatch orders to different vehicles, which obviously cannot be handled by TSP solutions. Therefore, we propose approximation methods for MPPR in Section 3 and 4.

A solution of MPPR includes order dispatch and route planning. Route planning estimates the profit of assigning an order to a vehicle, based on which orders are dispatched. Route planning aims to minimize the travel distance, which has been widely studied in existing works [6, 20, 15, 10, 14]. For efficiency concern, we follow the setting in [6, 20, 10, 14] that after receiving a new order, a vehicle keeps its existing orders' permutation unchanged. That is, given an order-vehicle pair, $(o_i, v_j)$, we insert $o_i$'s pick-up/drop-off locations into $v_j$'s plan like [6, 20]. This insertion-based algorithm traverses all possible insertion location pairs in the plan for the new order, and adopts the one with the least incurred travel cost.

The route planning algorithm outputs a plan of pick-up/drop-off for the vehicle, which decides the vehicle's route. We assume that drivers always adopt the shortest path given two locations. The algorithm has a time complexity of $O(\hat{c}^2 q)$, where $\hat{c}$ is the maximum capacity of the vehicles and $O(q)$ is the time cost of a shortest path query. In the following sections, we propose approximation methods for our focused problem, i.e., order dispatch.

# 3. THE GREEDY ALGORITHM

Greedy based algorithms have been demonstrated effective in spatial-related assignment problems [31, 25]. In this section, we propose a Greedy algorithm. It proceeds step-by-step, and in each step it makes the dispatch which brings the maximum immediate profit gain. As shown in Algorithm 1, the Greedy algorithm keeps dispatching orders until there is no feasible order-vehicle pairs.

Algorithm 1 first puts all feasible dispatch pairs $(o_i, v_j)$ into *pool* and calculates their immediate profit gains (lines $2 \sim 6$). Feasible dispatch pairs are those which satisfy the constraints in Definition 8. The profit gain $P_{i,j}$ is calculated as follows:

$$P_{i,j} = rev_i - \alpha * \Delta t_j(o_i) - bcost * \mathbb{1}(v_j \; is \; empty) \qquad (2)$$

where $\Delta t_j(o_i)$ is the increase in travel time from inserting $o_i$ into $v_j$'s travel plan. $\mathbb{1}(v_j \; is \; empty)$ is an indicator function which judges whether the vehicle is empty. If so, the base fee, *bcost*, needs to be deducted from the earning. After this initialization, the Greedy algorithm jumps to the *while* loop (lines $7 \sim 18$) which

**Algorithm 1** The Greedy algorithm

**Input:** order set $O$, vehicle set $V$
**Output:** updated plans of vehicles.
1: $pool \leftarrow \phi$
2: **for all** $(o_i, v_j) \in O \times V$ **do**
3: **if** $(o_i, v_j)$ is feasible **then**
4: Add pair $(o_i, v_j)$ into $pool$ and calculate $P_{i,j}$
5: **end if**
6: **end for**
7: **while** $pool \neq \phi$ **do**
8: $(o_{i*}, v_{j*}) \leftarrow \arg\max_{(o_i, v_j) \in pool} P_{i,j}$.
9: **if** $P_{i,j} < 0$ **then**
10: **break**.
11: **end if**
12: Dispatch $o_{i*}$ to $v_{j*}$ and update $plan_{j*}$.
13: $\forall v_j$, if $(o_{i*}, v_j) \in pool$, remove $(o_{i*}, v_j)$
14: **for all** $(o_i, v_{j*}) \in pool$ **do**
15: Update $P_{i,j*}$ if it remains feasible.
16: Remove $(o_i, v_{j*})$ from $pool$ otherwise.
17: **end for**
18: **end while**
19: **return** $R$.

greedily dispatches one order at a time. In each execution of the loop, the optimal pair $(o_{i*}, v_{j*})$, which brings the maximum immediate profit gain, is found and removed from the pool (line 8). The algorithm stops if the profit is negative (line $9 \sim 11$), otherwise the corresponding dispatch is conducted (line 12). The remaining candidate pairs of $o_{i*}$ are all removed (line 13). Similarly, due to the update of $plan_{j*}$, the candidate pairs of $v_{j*}$ are re-evaluated in lines $14 \sim 17$.

Note that we consider the case where feasible order-vehicle pairs become infeasible (lines $13 \sim 17$), but do not consider the other direction. This is because it is impossible that infeasible pairs become feasible after a dispatch.

**Lemma 3.1.** *$\forall_{i,j}$, $(o_i, v_j)$ is feasible only if it is feasible before the last dispatch.*

*Proof.* We conduct the proof by contradiction. Suppose after a dispatch of $(o_1, v_1)$, an infeasible dispatch pair $(o_2, v_1)$ becomes feasible. In the following, we show that any feasible insertion of $o_2$ into $plan_1$ would make it a feasible insertion before the dispatch, which contradicts that $(o_2, v_1)$ is originally infeasible.

Without loss of generality, we assume the plan of $v_1$, $plan_1$, as $v_1 o_0^s o_1^s o_1^e o_0^e$, and assume a feasible insertion of $o_2$ as $plan_1'$: $v_1 o_0^s o_1^s o_2^s o_1^e o_2^e o_0^e$. Then, $plan_1'$ respects the three constraints, i.e., detour constraint, pick-up constraint and capacity constraint. We construct a new plan $plan_1''$ by removing $o_1$ from $plan_j'$, which is $v_1 o_0^s o_2^s o_2^e o_0^e$. It is obvious that $plan_j''$ satisfy the capacity constraint. Besides, both $o_0$ and $o_1$ must satisfy the detour constraint and pick-up constraint, because their detour travel time and pick-up time both decrease benefited from removal of $o_1$. This indicates that $plan_j''$ is a valid plan, and therefore $(o_2, v_1)$ should be feasible before the dispatch of $(o_1, v_1)$. This contracts our assumption. Though our proof proceeds by giving an example, it holds for other types of travel plans and insertions. □

Under same conditions (demand-supply condition, ordering period, etc.), it is clear that common ridesharing platforms (Didi Chuxing, Uber) would make a higher price for longer trips. Then, in Algorithm 1, orders with long trips tend to be prioritized and assigned

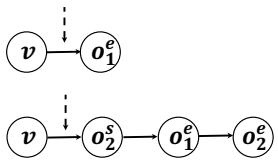



**Figure 3: Example of Lemma 3.2**

in early steps, because it greedily pursues profits. In the following, we show that this prioritization pattern is wise/reasonable with some discussions. Dispatching long trips at first is beneficial for both empty and manned vehicles.

For empty vehicles, assigning them a long trip as the first trip offers them a long initial travel. This brings more opportunities for their travel to be shared with latter trips.

For manned vehicles, without loss of generality, we assume the initial travel plan of a manned vehicle is $vo_1^e$. A new order $o_2$ is firstly added to the plan as shown in Figure 3. Let $D$ and $D'$ denote the travel distance of $vo_1^e$ and $vo_2^s o_1^e o_2^e$ respectively. Besides, let $\Delta D(\cdot)$ and $\Delta D'(\cdot)$ denote the distance increase of inserting a node right after $v$ in the two plans respectively. That is, for another order $o_3$, $\Delta D(o_3^s) = d(v, o_3^s) + d(o_3^s, o_1^e) - d(v, o_1^e)$, and $\Delta D'(o_3^s) = d(v, o_3^s) + d(o_3^s, o_2^s) - d(v, o_2^s)$.

**Lemma 3.2.** $\forall o_1, o_2, o_3, \Delta D(o_3^s) - \Delta D(o_2^s) \leq \Delta D'(o_3^s)$

*Proof.* With some simple deductions, we have:

$$\Delta D(o_3^s) - \Delta D'(o_3^s) = d(o_3^s, o_1^e) - d(o_3^s, o_2^s) + d(v, o_2^s) - d(v, o_1^e).$$

According to triangle inequality, we have $d(o_3^s, o_1^e) - d(o_3^s, o_2^s) \leq d(o_2^s, o_1^e)$. Then,

$$\Delta D(o_3^s) - \Delta D'(o_3^s) \leq d(o_2^s, o_1^e) + d(v, o_2^s) - d(v, o_1^e) = \Delta D(o_2^s)$$

□

Lemma 3.2 states that the new insertion cost of latter orders, $\Delta D'(o_3^s)$, is lower bounded by their original insertion cost minus the insertion cost of $o_2^s$, i.e., $\Delta D(o_3^s) - \Delta D(o_2^s)$. Note that this lower bound is tight. $\Delta D(o_3^s) - \Delta D(o_2^s) = \Delta D'(o_3^s)$ happens when $o_2^s = o_3^s$, i.e., the two orders have same origin. Besides, Lemma 3.2 also holds when $\Delta D'(\cdot)$ refers to the distance increase of insertion between $o_2^s$ and $o_1^e$.

We know that if the first added trip, $o_2$, is a longer trip, their insertion cost, $\Delta D(o_2^s)$, tends to be larger. According to Lemma 3.2, for latter orders, their least possible insertion cost gets smaller.

**Time complexity.** Let $deg_v$ and $deg_o$ denote respectively the average number of feasible matches per vehicle and per order initially. The initialization (lines $2 \sim 6$) costs $\mathcal{O}(|V||O|\hat{c}^2 q)$, where $\mathcal{O}(\hat{c}^2 q)$ is the time cost of route planning as mentioned in Section 2.3, and $|V|$ and $|O|$ represent the number of vehicles and orders respectively. In the *while* loop, dispatch operations in lines $8 \sim 12$ cost $\mathcal{O}(|V| deg_v)$ in total. Update operations in lines $13 \sim 17$ cost $\mathcal{O}(deg_v \hat{c}^2 q)$. The *while* loop executes at most $|O|$ times, leading to an overall complexity of $\mathcal{O}\left((|V| + \hat{c}^2 q) deg_v |O|\right)$. Summing up the initialization cost and loop execution cost, the time complexity of Greedy is $\mathcal{O}\left((\hat{c}^2 q + deg_v)|O||V| + \hat{c}^2 q |O| deg_v\right)$.

# 4. MATCHING-BASED METHODS

In this section, in contrast to greedily assigning orders, we consider matching orders and vehicles in a one-shot manner. We treat the vehicles and orders as nodes in a bipartite graph, and combine them using matching algorithms. More specifically, we first show in Section 4.1 that a dispatch solution can be obtained by using maximum weighted matching algorithms. Then, in section 4.2, we improve the bipartite matching method by properly packing the orders *a priori*.

## 4.1 Bipartite Matching (BM)

In [22], vehicles and orders are treated as nodes of a bigraph, and a maximum weighted matching is computed to dispatch the orders. Similarly, we can obtain a dispatch solution with the graph matching method. To construct the graph, we firstly create $\max\{|O|, |V|\} - \min\{|O|, |V|\}$ dummy nodes to ensure equal number of nodes in the two sides. Then, we obtain a complete bigraph by connecting nodes in one side to all nodes from another side. For each order-vehicle pair, we set their edge weights to $P_{i,j}$ if the dispatch is feasible and $P_{i,j} > 0$, otherwise we set the edge weights to 0.

The maximum weighted matching of the constructed bipartite graph can be found by using combinatorial algorithms such as the Kuhn-Munkres method [21]. A matching solution can be transformed into a dispatch solution by selecting the matched pairs with non-zero edge weights. The disadvantage of this pure matching-based method lies in that it does not consider packing new orders. Two new orders in the same round would not be combined even if their origins/destinations are very close to each other.

Figure 4(a) gives a toy example where there are three orders $(o_1, o_2, o_3)$ and two vehicles $(v_1, v_2)$. $v_1$ and $v_2$ are currently located at the origin of $o_1$ and $o_2$ respectively. The order prices $(rev_i)$, the base fare $(bcost)$, and the charge per unit distance $(\alpha_d)$ in this example are shown in Table 2. All the road segments are of unit length, so the cost of traveling per segment is $\alpha_d = 1$. Accordingly we can construct a bigraph as shown Figure 4(b), where the zero weighted edges and dummy vehicles are omitted for simplicity. The maximum weighted matching on the bigraph indicates the following dispatch: $(o_1, v_1)$, $(o_3, v_2)$. Note that $o_2$ should be matched to a dummy vehicle, which represents it is not dispatched.

## 4.2 Packing Based Matching (PBM)

In this subsection, we tend to improve the pure matching method (in Section 4.1) by properly packing orders beforehand. Observing that the current practice in typical ridesharing applications (Didi Chuxing) allows at most three riders in a shared travel, packing more than three orders is meaningless. We consider only packing two orders in our algorithm. This is because, we observe from the real data that in a dispatch round, it is uncommon that three pended orders can be packed without violation of the constraints. The reason is that the pended order set in a round is relatively sparse, and its size is around several hundred. Generally the triply shared rides are produced by assigning an order or a pack (of size two) to manned vehicles. Manned vehicles refer to those which have received orders in previous rounds.

The packing based matching algorithm proposed in this section, consists of two steps:

(i) **Order packing.** Construct an order graph, based on which pack the orders.

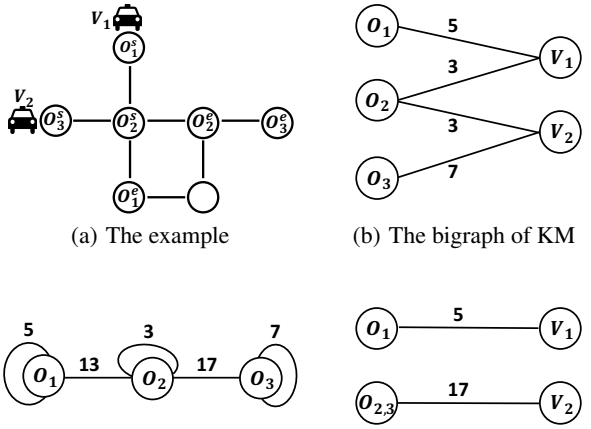
(ii) **Order dispatch.** Construct a bigraph with orders/packs and vehicles, based on which dispatch the orders.

Actually, step (ii) does exactly what we have described in Section 4.1. That is, for each order or order pack, we compute the profit of assigning it to each vehicle, which serves as the edge weight in the bigraph. A dispatch solution is then obtained by computing a maximum weighted matching. In the following, we focus on step (i), explaining how to construct the order graph and pack the orders.

To construct the order graph, we virtually create an empty vehicle at the origin of each order like [1]. Assuming the availability of

**Table 2: Variables of example in Figure 4**

| Variable | $rev_1$ | $rev_2$ | $rev_3$ | $bcost$ | $\alpha_d$ |
|---|---|---|---|---|---|
| Value | 12 | 10 | 15 | 5 | 1 |




(a) The example (b) The bigraph of KM

(c) The order graph in PBM (d) The bigraph with packs in PBM

**Figure 4: An example of matching methods.**

such virtual vehicles, the order packing stage aims to maximize the overall profit of dispatching all orders. Two orders $o_1$ and $o_2$ are connected in the order graph, if they can be combined without violation of constraints $I \sim III$ (in Definition 8). There are four possible plans to combine $o_1$ and $o_2$: $o_1^s o_2^s o_1^e o_2^e$, $o_1^s o_2^s o_2^e o_1^e$, $o_2^s o_1^s o_1^e o_2^e$ and $o_2^s o_1^s o_2^e o_1^e$. For plans starting from $o_1^s$ ($o_2^s$), the virtual vehicle created at $o_1^s$ ($o_2^s$) would be the one to conduct the travel. A plan is valid if it does not violate the constraints. $o_1$ and $o_2$ are connected in the graph if at least one valid plan yields positive profit. The edge weight is set to the maximum among the profits of the valid plans. By traversing all order pairs, we obtain a general graph (not bipartite), where an edge represents a feasible pack and its weight represents the profit of the pack. One easy follow-up is to compute a maximum weighted matching on the graph, which yields a packing solution where the total profit of the packs is maximized. However, this method fails to consider the profits of the un-packed orders. Maximization of the profit of order packs does not guarantee the maximization of profit of all orders. Remember that the order packing step is to achieve the largest amount of profit from dispatching all orders, assuming enough transportation capability. Nevertheless, in the following, we would show that the aforementioned order packing problem can be solved by weighted matching algorithms with a reduction.

**Lemma 4.1.** *The order packing problem can be reduced to the maximum weighted matching problem.*

*Proof.* Given an order graph in the packing stage, we augment the graph by additionally creating a dummy node for each order node. Each dummy node is connected to its corresponding order node with the edge weight equal to the order profit. With respect to this augmented graph, each order packing solution leads to a matching solution, and vice versa.

$\Longrightarrow$: Given an order packing solution, we can construct a matching solution by matching the un-packed orders to their dummy nodes.

$\Longleftarrow$: Given a matching solution, we can obtain a packing solution by only packing nodes which are not matched to dummy nodes. □

Regarding the toy example in Figure 4(a), 4(c) and 4(d) present the corresponding order graph and the bigraph with packs. In Figure 4(c), the dummy nodes are represented by the edges linking nodes to themselves. Numbers on the edges represent the profits of the corresponding packs. The maximum weighted matching of this graph should be $(o_1, o_1)$, $(o_2, o_3)$, indicating packing $o_2$ and $o_3$ only. Subsequently we obtain the order-vehicle bigraph as shown in Figure 4(d), by considering the actual vehicles $v_1$ and $v_2$. The maximum weighted matching on this bigraph is $(o_1, v_1)$, $(o_{2,3}, v_2)$.

**Time complexity.** In the graph construction stage, the order packing and edge calculation costs $\mathcal{O}(|O|^2 \hat{c}^2 q)$ and $\mathcal{O}(|O||V|\hat{c}^2 q)$ respectively. In the order packing stage, we adopt the matching algorithm described by Galil [11], whose time complexity is $\mathcal{O}(|O|^3)$. The time complexity of the Kuhn-Munkres algorithm [21] for weighted bipartite matching is $\mathcal{O}\left((\max\{|V|, |O|\})^3\right) \rightarrow \mathcal{O}(|O|^3 + |V|^3)$. Summing up these costs, the time complexity of the packing based matching method is $\mathcal{O}(\hat{c}^2 q|O|(|O|+|V|)+|O|^3+|V|^3)$.

# 5. OPTIMIZATIONS

Algorithms of real-time ridesharing are required to be rather efficient. In this section, we propose some optimization techniques to improve the dispatch efficiency. Section 5.1 describes a pruning strategy to avoid unnecessary distance queries, and Section 5.2 describes a node clustering method to approximate shortest-path distances with bounded errors.

## 5.1 Vehicle Pruning

When computing the profit of a dispatch $(o_i, v_j)$, $P_{i,j}$, we need to invoke the route planning algorithm (see Section 2.3) to insert $o_i$ into $v_j$'s plan. The planning algorithm returns the optimal insertion, or nothing if all insertions fail to meet the constraints. When real road networks are used, queries of shortest-path distance would be issued frequently during route planning. In our experiments with the Beijing road network, we found that even the state-of-art shortest-path algorithm, Contraction Hierarchy (CH) [12], still takes some milliseconds per query. No matter which dispatch algorithm is used, for each order, we need to traverse all vehicles to evaluate the dispatch feasibility and profits. This leads to a large amount of shortest path queries, whose time cost is unbearable.

Our observation from the real ridesharing data indicates that for an order, due to the waiting-time constraint, only its nearby vehicles can be dispatched. Most vehicles turn out infeasible because they are too far away from the order's origin. To save the route planning cost from these infeasible vehicles, we calculate the geographical distance to prune most of the infeasible vehicles. Given the longitude and latitude of two locations, we can calculate their geographical distance according to the Haversine formula[4].

Suppose the waiting-time constraint indicates a pick-up distance threshold as $d_{pt}$, we have:

**Lemma 5.1.** *An order-vehicle pair can be safely pruned if their geographical distance is larger than $d_{pt}$.*

The proof is straightforward: the geographical distance between two locations is a lower bound of their distance on any road network. The computation of the Haversine formula takes only some microseconds, which is much faster than the shortest path computation.

[4] https://en.wikipedia.org/wiki/Haversine_formula#cite_note-Inman_1835-4

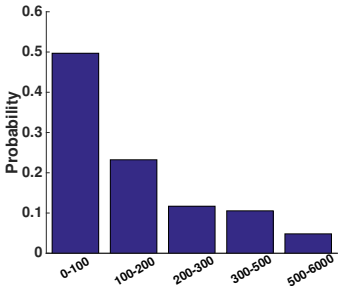



**Figure 5: PMF of network edge length.**

## 5.2 Node Clustering and Distance Approximation

In our experiment, we obtain the city road network from the Open Street Map (OSM)[5]. In the urban area of Beijing city (i.e., within the 5th Ring Road of Beijing), there are 35692 road nodes and 48192 edges. On such a road network, one shortest-path computation with CH [12] costs some milliseconds. The dispatch algorithms would issue a vast amount of distance queries, leading to an unbearable amount of time cost for real-time ridesharing.

The probability mass distribution of the edge length in the road network is shown in Figure 5. It can be observed that most of the edges are less than 500 $m$ (meter), and nearly half of the edges are within 100 $m$. This observation inspires us to cluster the road nodes, and approximates the distance of a node pair using the distance between their cluster centers. In the following, we first describe our node clustering algorithm (in Section 5.2.1), based on which we describe the distance approximation procedure (in Section 5.2.2).

### 5.2.1 *The Node Clustering Algorithm*

Since our purpose is to approximate the distance of two nodes by their centers' distance, the node clustering algorithm should control the distance between nodes and their centers to prevent large approximation errors.

To bound the distance among road nodes inside a cluster, the hierarchical clustering with complete linkage[6], can be an alternative. In complete linkage, the distance between two clusters is defined as the maximum distance between their nodes. Hierarchical clustering greedily combines two clusters with the minimum cluster distance, with space complexity equal to $O(|N|^2)$, where $|N|$ is the size of the node set, $N$. The employed road network in our experiment has 35692 nodes. This disables using hierarchical clustering due to its great memory requirement.

Observing that the network is sparse, we devise a method named Neighbor-Growing Clustering (NGC), which proceeds by operating the edge set. Subsequently its space complexity is $O(|E|)$, where $|E|$ is the size of the edge set. This would not cause a problem of memory. The idea of NGC is to keep picking the least weighted edge and merging its incident nodes, until the edge weight exceeds a given threshold, $\theta_d$. The threshold $\theta_d$ is the maximum allowed distance between a node and its cluster center, and is specified in the input. To make it easier for understanding, we give a running example of NGC here. Its formal algorithmic description can be found in our technical report.

[5] http://download.geofabrik.de/

[6] https://en.wikipedia.org/wiki/Hierarchical_clustering

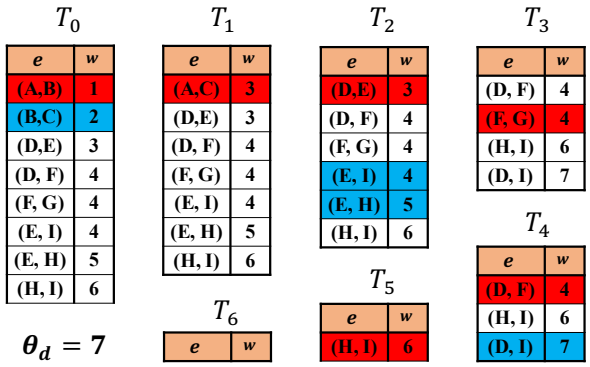



**Figure 6: Example of node clustering.**

We demonstrate the algorithm with the example in Figure 6. There are nine nodes in the network, $A \sim I$. Besides, the threshold $\theta_d$, is set to 7 in the input. Before the algorithm starts, we first remove edges whose lengths are larger than $\theta_d$, because they cannot be used to merge nodes subjecting to $\theta_d$. The survived edges and their lengths are presented in Column '$e$' and '$w$' of Table $T_0$ respectively. Note that length and weight are used interchangeably for this example.

In the first merging operation, the least weighted edge, $(A, B)$, is picked, shaded in red in Table $T_0$. Since $A$ and $B$ are both single nodes, we build a cluster containing these two nodes, with arbitrarily one of them as the center. Suppose $A$ is chosen as the center. Then, we need to grow new edges of $A$ bypassing $B$. Except $(A, B)$, $(B, C)$ is the only incident edge of $B$, shaded in blue. We delete it, and grow a new edge $(A, C)$, with length equal to $w(A, B) + w(B, C)$. The updated edge set thereafter is shown in Table $T_1$, according to which we perform the next merge. $(A, C)$ and $(D, E)$ have the least weight, and we suppose $(A, C)$ is picked. Note that ties are broken arbitrarily. Since $A$ is already the center of $\{A, B\}$, the single node $C$ is merged into $A$. No new edge is grown from $A$ because $C$ has no other edges. In Table $T_2$, $(D, E)$ is the next picked edge, and we suppose $D$ is chosen as the center. $(E, I)$ and $(E, H)$ would be removed, but only $(D, I)$ is grown and added to Table $T_3$. This is because $w(D, H) = w(D, E) + w(E, H) = 8$, is greater than $\theta_d$ $(= 7)$. In the 4th merge, single nodes $F$ and $G$ are clustered with $F$ as the center.

In the 5th merge, the picked edge, $(D, F)$ connects two existing clusters, i.e., $\{D, E\}$ and $\{F, G\}$. To merge two clusters, we try by setting the new center to one of the existing centers, i.e., $D$ and $F$. If $D$ is selected as the new center, its distance to $G$, which is $w(D, F) + w(F, G) = 8$, would be larger than $\theta_d$. We then try $F$. Since $w(F, E) = w(D, E) + w(D, F) = 7$ is no larger than $\theta_d$, $F$ is chosen as the new center. The only existing edge of $D$, $(D, I)$ is removed, but no new edge can be grown within distance of $\theta_d$. Note that the two clusters would not be merged and it would directly jump to the next merge if neither of $D$ and $F$ can be the new center. $H$ and $I$ are merged in the final step. The clustering result is : $\{\mathbf{A}, B, C\}$, $\{D, E, \mathbf{F}, G\}$, $\{\mathbf{H}, I\}$, where centers are marked in bold.

The difference between NGC and hierarchical clustering lies in: $I$. NGC acts on the evolutional edge set while hierarchical clustering acts on the pairwise distances among nodes; $II$. NGC bounds the distance between a node and its center, while hierarchical clustering bounds the pairwise distances among nodes in the same cluster.

The space complexity of NGC is $O(|E|)$. There are at most $|N|$ merges, and the time cost of one merge is $O(|E|)$. Therefore, the time complexity of NGC is $O(|E||N|)$.

### 5.2.2 Shortest-path Distance Approximation

By using NGC to cluster nodes, for a shortest-path query of $(n_1, n_2)$, we can approximate their shortest-path distance with the distance between their corresponding cluster centers. We set $\theta_d = 500\ m$ for NGC. In this way, the resultant number of clusters is 5971, which is small enough, and, as we would show later, the approximation error would not be large. Because the number of centers is small enough, we can compute their pairwise shortest-path distances on the original network *a priori*, and cache these distances in main memory during runtime. Therefore, using centers' distance for approximation is very fast. In addition, the following lemma guarantees that the distance error brought by using centers' distance is bounded.

**Lemma 5.2.**

$$\forall n_1, n_2,\ |d(n_1, n_2) - d(c_1, c_2)| \leq w(e_{n_1,c_1}) + w(e_{n_2,c_2}) \leq 2\theta_d$$

*where $c_1$, $c_2$ are the centers of $n_1$ and $n_2$ respectively, $w(e_{n_1,c_1})$ and $w(e_{n_2,c_2})$ are their distances in the clustering result of NGC.*

*Proof.* We need to show that:

$$-w(e_{n_1,c_1}) - w(e_{n_2,c_2}) \leq d(n_1, n_2) - d(c_1, c_2) \quad \text{and}$$

$$d(n_1, n_2) - d(c_1, c_2) \leq w(e_{n_1,c_1}) + w(e_{n_2,c_2})$$

Since there exists a path $n_1c_1c_2n_2$, we know that $d(n_1, n_2) \leq w(e_{n_1,c_1}) + w(e_{n_2,c_2}) + d(c_1, c_2)$. This proves the second inequality. In the following, we demonstrate the first inequality.

Suppose $d(n_1, n_2) - d(c_1, c_2) < -w(e_{n_1,c_1}) - w(e_{n_2,c_2})$. Then, $d(c_1, c_2) > d(n_1, n_2) + w(e_{n_1,c_1}) + w(e_{n_2,c_2})$. Since $d(c_1, c_2)$ is the shortest-path distance and there is a path $c_1n_1n_2c_2$, we have $d(n_1, n_2) + w(e_{n_1,c_1}) + w(e_{n_2,c_2}) \geq d(c_1, c_2)$. Subsequently,

$$d(c_1, c_2) > d(n_1, n_2) + w(e_{n_1,c_1}) + w(e_{n_2,c_2}) \geq d(c_1, c_2)$$

This makes a contradiction. □

Since we set $\theta = 500\ m$, this guarantees that distance error is at most $1\ km$ (kilometer). If the speed of a vehicle is $60\ km$ per hour, $1\ km$ error corresponds to a time deviation of 1 minute, which does not make big difference to most of the travels. If one insists on using the exact shortest-path distance, we can use $d(c_1, c_2) - w(e_{n_1,c_1}) - w(e_{n_2,c_2})$ as a lower bound of $d(n_1, n_2)$. Then, it can be used to prune the vehicles as stated in Lemma 5.1.

# 6. SIMULATION SETUP

We develop a simulation framework to evaluate the performance of the proposed methods. It is devised based on the real data retrieved from the ridesharing business of Didi Chuxing. The experiments in Section 7 are run under this simulation framework.

**Vehicle simulation.** We retrieve the status change records of the vehicles, where each record indicates that a vehicle goes either online or offline. In our simulation, a vehicle is available from when it goes online to when it goes offline. Note that there may be more than one online record of a vehicle in a day, and each of them indicates an availability time span of the vehicle. Each online record also reports the initial geographical location of the corresponding vehicle. We further map this location to the nearest node in the road network.

The speed of the vehicles is set to $60\ km/h$. An idle vehicle walks randomly along the road network. In contrast, a vehicle with unfinished orders moves according to its plan. In our implementation, if the expected finish time of a busy vehicle exceeds its online time span after a dispatch, it would become unavailable for subsequent dispatch rounds.

**Order simulation.** Each ridesharing order record is associated with an origin, a destination, a price and a request time. The origin and destination are mapped to their nearest nodes in the road network. In our simulation, we create the orders according to their request times. Remember that the dispatcher assigns orders round by round. In each round, an order would be either dispatched or pended to next round. In practice, an order may be cancelled by the requester if not dispatched for a long time. As mentioned before, the percentage of cancelled orders in Didi Chuxing is above 20% [30]. Therefore, it is important to model order cancellation to coincide with the practice.

To simulate orders' cancellation, we first perform equal width bucketing over their waiting times, where the bucket width is set to five seconds. For an order $o_i$ currently with waiting time $wt_i$, its corresponding bucket index is $bidx(o_i) = \lceil \frac{wt_i}{5} \rceil$.

We collect a set of cancelled orders from historical data of Didi Chuxing. Then, for each bucket $b$, we retrieve the number of cancelled orders in this bucket, and denote it as $\#cancel(b)$. After a waiting time of $wt_i$, order $o_i$ would either be cancelled and stop at $b_i$ or keep waiting and move to subsequent buckets. Then, the probability that an order is cancelled in bucket $b$ is equal to: $\#cancel(b)$ divided by the sum of $\#cancel(b)$ and the number of cancelled orders fallen into buckets after $b$. That is,

$$Pr(o_i \text{ cancelled at } bidx(o_i)) = \frac{\#cancel(bidx(o_i))}{\sum_{b \in [bidx(o_i), b_{max}]} \#cancel(b)} \quad (3)$$

For an undispatched order, before we start next round's order dispatch, we first update its waiting time by adding the time duration between two rounds. Then, with respect to its old and new waiting time, we obtain its last and current bucket index. We go through the buckets between these two indices, and check whether it would be cancelled in these buckets according to Equation (3). Only orders which survive from all passed buckets would be kept to next round's dispatch.

For instance, suppose in a dispatch round, order $o_1$, which has already been pended for $17\ s$, is still not dispatched. It has to wait for next round's dispatch, and until then its waiting time would be increased to $32\ s$, assuming the time window length of a round as $15\ s$. Subsequently its passed buckets would be $b_4$, $b_5$ and $b_6$. Let $pr_4$, $pr_5$ and $pr_6$ denote the cancellation probability (Equation (3)) of these buckets. Then, the probability that $o_1$ is not cancelled by next round is:

$$(1 - pr_4) * (1 - pr_5) * (1 - pr_6)$$

**Payment mechanism.** The profit of the platform is equal to the price sum of dispatched orders minus the payment to the drivers. In Didi Chuxing, the price of a ridesharing order is displayed to the requesters after the inputs of origin and destination, and is fixed regardless of the order dispatch results. We obtain directly the order prices from the historical records. The payment to drivers includes the base fare and the charge of travel distances. Following the common practice in China, the base fare is $10\ yuan$ by default. Besides, the payment to vehicles' traveling per kilometer is set to $2\ yuan$.

# 7. EXPERIMENT

In this section, we run experiments to evaluate the proposed methods. We first describe our experimental settings in Section

**Table 3: Experimental settings**

| Variables | Values |
|---|---|
| $\theta_{dr}$ | 1.2, 1.5, **1.8**, 2.1, 2.4 |
| $\theta_{pt}$ | 300, 600, **900**, 1200 |
| $t_{rnd}$ | 5, 10, **15**, 20 |
| $bcost$ | 7, **10**, 13 |

7.1. Then, we validate the effectiveness of the order cancellation model and the optimization techniques in Section 7.2.1 and Section 7.2.2 respectively. Besides, we report the evaluation results of our proposed methods in Section 7.2.3. Finally, we compare them with the online method in Section 7.2.4.

## 7.1 Experimental Setting

The simulation experiment is run in the urban area of Beijing, i.e., the region bounded by the 5th Ring Road, with longitude from $116.1996^\circ$ to $116.5457^\circ$ and latitude from $39.7558^\circ$ to $40.0229^\circ$. Only vehicles and orders inside this area are considered. The data is retrieved from the time period between $7:00\ am$ and $7:30\ am$ in a normal weekday. Note that $7:00 \sim 7:30\ am$ is one of the peak ordering periods in a day. There are totally 4907 orders in this period, and 2.73 new orders come per second on average. The road network is obtained from Open Street Map[7].

**Control variables.** Remember that the proposed order dispatch algorithms are round-based, and each round is associated with a time window to batch the new orders. Let $t_{rnd}$ denote the time window of one round, and we vary $t_{rnd}$ from 5 $s$ to 20 $s$ to examine its effects. Besides, in Definition 8, we have three optimization constraints, regarding the detour ratio $\theta_{dr}$ (Definition 4), pick-up time $\theta_{pt}$ (Definition 5) and the base fare $bcost$ (Definition 6). The detour ratio $\theta_{dr}$ and pick-up time $\theta_{pt}$ are varied in $\{1.2, 1.5, 1.8, 2.1, 2.4\}$ and $\{300, 600, 900, 1200\}$ (in seconds) respectively. Besides, we vary the base fare $bcost$ from 7 to 13. Finally, to coincide with the current ridesharing practice of Didi Chuxing, the vehicle capacity is fixed at 3. Table 3 summarizes the experimental setting of the control variables, where the default values are marked in bold.

**Comparison among proposed methods.** We examine the effectiveness and efficiency of the proposed methods, including Greedy (in Section 3), BM (bipartite matching in Section 4.1) and PBM (packing based matching in Section 4.2). The amount of attained profit and the running time per round are reported.

**Comparison with the state-of-art method.** For online methods [20, 15], they immediately dispatch an order on its arrival. Their difference lies in two aspects. Firstly, in terms of searching insertions on a vehicle's travel plan, [15] enumerates all possible re-orderings while [20] respects the original ordering of existing orders. For example, suppose the travel plan of a vehicle $v$ is $vo_1^e o_2^e$. To add a new order $o_3$, let us assume the optimal new plan is $vo_3^s o_2^e o_1^e o_3^e$, which would be found by [15]. In contrast, the new plan obtained by [20] may be $vo_1^e o_3^s o_2^e o_3^e$ to respect that $o_1^e$ is originally ahead of $o_2^e$. [15] opts to pursue a better dispatch result with more time complexities. However, by using kinetic trees to store the plans, [15] shows that the dispatch of an order can finish within dozens of milliseconds.

Secondly, in terms of the optimization objective, when dispatching an order to a vehicle, [20] minimizes the additional incurred travel distance, while [15] minimizes the travel distance of the augmented plan. For example, let the travel distance of the plan of $v_1$ be $D_1$, and it would be increased to $D_1'$ if dispatched with a new order $o_1$. Then, the additional incurred travel distance refers to $D_1' - D_1$, and the travel distance of the augmented plan refers to $D_1'$. Suppose there is another vehicle $v_2$ with travel distance $D_2$ and $D_2'$ corresponding to before and after insertion of $o_1$ respectively. Besides, let $D_1' < D_2'$ and $D_2' - D_2 < D_1' - D_1$. Then, [20] would dispatch $o_1$ to $v_2$ while [15] would dispatch $o_1$ to $v_1$.

To evaluate [15] under our simulation, we offer the streaming order requests according to their creation times. To adapt [15] to solve our problem, on arrival of $o_i$, [15] would need to find the optimal vehicle which yields the maximum profit. That is,

$$\arg\max_{v_j}\ rev_i - \alpha_d * \Delta d(plan_j)$$

where $\Delta d(plan_j)$ is the distance increase of $v_j$'s plan from serving the new order $o_i$, and $\alpha_d$ is the charge per unit distance. Since $rev_i$ is fixed, to maximize the profit is to minimize $\Delta d(plan_j)$, i.e., the additional incurred distance, which is exactly the objective in [20].

Therefore, we build an online baseline to minimize the additional incurred travel distance like [20]. However, the kinetic-tree-based order insertion method of [15], which enumerates re-orderings, is adopted to make the dispatch more effective. In other words, we adapt the order insertion method in [15] to optimize the objective in [20].

Besides, in [15], an order would be notified about its dispatch result right after its arrival. If there is no feasible dispatch, the riders can re-submit the requests. Therefore, in our experiment, we let an order request again after some seconds if it cannot be dispatched. An order would keep requesting until it is dispatched or its number of requests exceeds the a pre-defined limit.

Note that we do not compare with existing works of DARP, because they [8, 10, 5, 16] assume all orders are served, which cannot be true in real-time ridesharing under constraints of waiting time and detour distance. For works without such an assumption [1], it is unclear on how to regularize the penalty term in its objective function.

**Table 4: Pruning success rate**

| $\theta_{pt}$ (in seconds) | 300 | 600 | 900 | 1200 |
|---|---|---|---|---|
| Pruning success rate | 80.35% | 57.03% | 33.43% | 32.48% |

## 7.2 Experimental results

### 7.2.1 Effectiveness of order cancellation model

We first examine the order cancellation model presented in Section 6. We obtain the bucketing results of the collected cancelled orders (from real data). Then, we create a set of simulated orders, whose size equals that of the real cancelled order set. The simulated orders are then cancelled according to Equation (3) as time goes by. Note that we do not dispatch these orders as we are only interested in their cancellation time distribution. Finally, we calculate the number of cancelled simulated orders of each bucket.

The cancellation bucket indices of the real and simulated orders form two samples. We examine whether these two samples follow the same distribution using the *two-sample Kolmogorov-Smirnov test*[8]. The null hypothesis is that the two samples are from the same continuous distribution. The calculated $p\text{-}value$ is equal to 0.9872, which is larger than 0.05. Therefore, it does not reject the null hypothesis at the 5% significance level.

### 7.2.2 Effectiveness of Optimization techniques

**Effectiveness of vehicle pruning.** As presented in Section 5.1, we prune infeasible vehicles for orders by comparing their geographical distance to $\theta_{pt} * speed$. On our machine, it only takes

[7] https://www.openstreetmap.org

[8] https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test

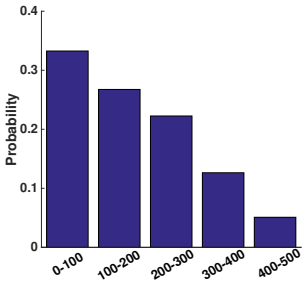



**Figure 7: PMF of distances between nodes and their centers.**

**Table 5: Profit evaluation time cost per round.**

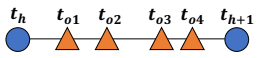
| $t_{rnd}$ | 5 $s$ | 10 $s$ | 15 $s$ | 20 $s$ |
|---|---|---|---|---|
| Time cost | 0.950 $s$ | 1.317 $s$ | 1.071 $s$ | 1.538 $s$ |

some microseconds to compute the geographical distance using the Haversine formula. The pruning success rate under different $\theta_{pt}$'s is shown in Table 4. When $\theta_{pt} \leq 600$ $s$, the pruning success rate is larger than 50%, which means that more than half of the vehicles can be successfully pruned for each order. Even when $\theta_{pt} = 1200$ $s$, the pruning success rate is still around $\frac{1}{3}$.

**Effectiveness of distance approximation.** In Section 5.2, we propose a node clustering algorithm so that the vast amount of shortest-path queries can be answered in real-time using the cluster centers. We set $\theta_d = 500$ $m$ for NGC to cluster the nodes, which leads to 5971 clusters. We show in Figure 7 the probability mass distribution of the distances between road nodes in the original network and their centers in the clustering result. It can be found that more than half of the nodes are within 200 $m$ away from their centers, though the threshold is set to 500 $m$. We further calculate the average distance between nodes and their centers, which is 175.6 $m$. Therefore, according to Lemma 5.2, the mean error of the proposed distance approximation technique is no larger than $2 * 175.6\ m = 351.2\ m$, which is quite small for a normal vehicle travel.

### 7.2.3 *Comparison among the proposed methods*

We report the comparison results of our methods in this section, including the achieved profits (*yuan*) and the running times (seconds). We report the running times which exclude the cost of evaluating the profits of the order-vehicle pairs (i.e., $P_{ij}$'s). This is because we do these computations on arrival of each order. For example, let $t_h$ and $t_{h+1}$ be two adjacent time epochs when the dispatcher functions, as shown in the figure below. For each dynamically arrived order ($o_1 \sim o_4$), its feasible vehicles and the corresponding profits are evaluated at its arrival time ($t_{o1} \sim t_{o4}$).




In our experiment, the profit evaluation time cost per order is 0.0139 seconds. As mentioned before, the number of new orders coming per second is 2.73, so there is no problem in evaluating the profits when orders arrive. Note that our experimental period, $7:00 \sim 7:30$ $am$, is already the ordering peak in a day.

If we do not conduct the evaluation on orders' arrival as described above, we report the corresponding profit evaluation time cost per round in Table 5, which are roughly same for the proposed algorithms.

**Effect of $\mathbf{t_{rnd}}$.** The experimental results of varying $t_{rnd}$ are shown in Figure 8(a) and 8(b). In terms of profit, PBM performs the best overall. Its profit ranks second when $t_{rnd} = 5$, and ranks top in all other cases. The reason why BM is better when $t_{rnd} = 5$ is that fewer orders are batched in the input. This makes it hard for PBM to obtain good shared rides.

However, from Figure 8(b) we can observe that the average running times of BM and PBM are around 5 $s$ and 6 $s$ respectively. Note that the length of time window in a round is the maximum allowed running time for the dispatch algorithms. It is inappropriate to adopt BM and PBM when $t_{rnd} \leq 5$, as their running times can exceed the time limit. Greedy is efficient throughout all values of $t_{rnd}$. Therefore, when $t_{rnd} \leq 5$, efficient algorithms like Greedy would be the choice.

Another interesting finding is that PBM is actually slightly more efficient than BM, though it additionally has an order packing step. The reason may be that after packing orders, the number of nodes in the order-vehicle graph becomes smaller, which accelerates the matching computation. Besides, the process of packing orders is in fact efficient because the order graph is sparse.

**Effect of $\theta_{\mathbf{pt}}$.** The experimental results of varying $\theta_{pt}$ are shown in Figure 8(c) and 8(d). In Figure 8(c), Greedy and PBM achieve the highest profit when $\theta_{pt} \leq 600$ and $\theta_{pt} \geq 900$ respectively. Besides, their profits drop after $\theta_{pt} = 600$ and $\theta_{pt} = 900$ respectively. The reason is that a larger amount of waiting time means a larger driving distance during pick-up, which leads to more payments to vehicles.

The running times of both BM and PBM increase when $\theta_{pt}$ increases. This is because when $\theta_{pt}$ is large, the number of non-zero edges in the order-vehicle graph becomes large, which leads to the increased time cost of running the matching algorithm.

**Effect of $\theta_{\mathbf{dr}}$.** Figure 9(a) and 9(b) show the experimental results of varying $\theta_{dr}$. In terms of profit, PBM performs best when $\theta_{dr}$ is large (1.8 and 2.1), while Greedy and BM are better when it is small (1.2 and 1.5). This is because a large $\theta_{dr}$ corresponds to a relaxed detour constraint, which brings more space to pack the orders. In this case, the packing-based algorithm, PBM, behaves better.

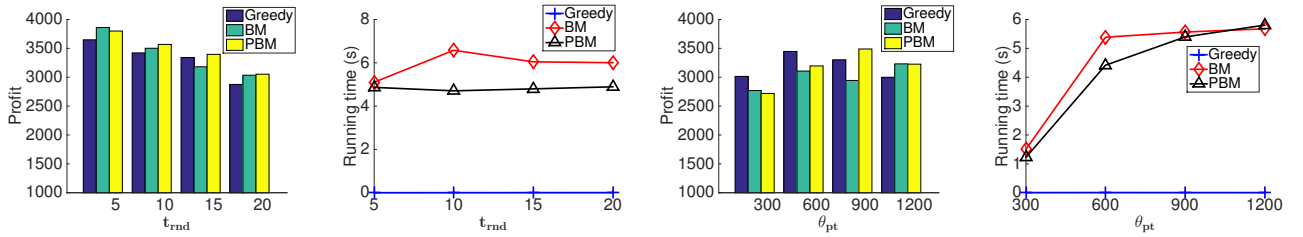
It can also be found that the profit does not always improve when $\theta_{dr}$ is increased. As shown in Figure 9(a), BM peaks at $\theta_{dr} = 1.5$, and Greedy & PBM peak at $\theta_{dr} = 1.8$. The profits of all methods go down when $\theta_{dr} = 2.1$. The reason is that a large $\theta_{dr}$ may result in combination of orders which not really share a long ride. The incurred large amount of detour distance leads to large payment to the vehicles. Subsequently, the achieved profit gets smaller.

The running times of all three methods are stable. They do not change significantly over different $\theta_{dr}$'s.

**Effect of *bcost*.** The base fare, $bcost$, is varied from 7 to 13. In Figure 9(c), PBM and Greedy achieve the highest profit when $bcost = 13$ and $bcost = 7$ respectively. The reason may be that when $bcost$ is large, the benefit of packing orders gets larger, because it employs less vehicles. Therefore, PBM behaves better. For the running times (in Figure 9(d)), there is no significant variation over different $bcost$'s.

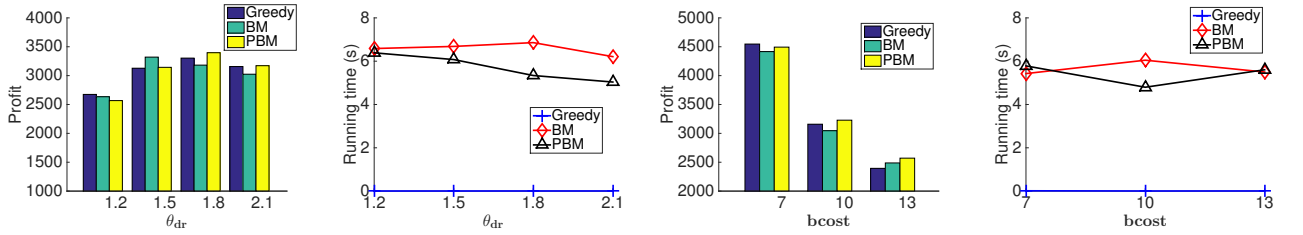
**Comparison to Optimal solution.** We propose approximation methods because the MPPR problem is NP-hard. It remains a question how far are these approximation solutions away from the optimum. Due to the great computational complexity of finding OPT (the optimal dispatch solution) on the large-scale real data, we opt to compare our methods with OPT in a small scenario.

To run a small-scale experiment, we randomly pick 8 orders and 8 vehicles from the historical vehicles and orders in the Dongcheng District of Beijing, and compare the achieved profits of our methods and OPT. As shown in Table 6, the profits are all 0 when




(a) Profits of various $t_{rnd}$'s (b) Running times of various $t_{rnd}$'s (c) Profits of various $\theta_{pt}$'s (d) Running times of various $\theta_{pt}$'s

**Figure 8: Effect of $t_{rnd}$ and $\theta_{pt}$.**




(a) Profits of various $\theta_{dr}$'s (b) Running times of various $\theta_{dr}$'s (c) Profits of various $bcost$'s (d) Running times of various $bcost$'s

**Figure 9: Effect of $\theta_{dr}$ and $bcost$.**

**Table 6: Comparison with OPT.**

| $bcost$ | Profits | | | |
|---|---|---|---|---|
| | Greedy | KM | PBM | OPT |
| 3 | 34.78 | 13.47 | 15.06 | 44.71 |
| 5 | 24.16 | 8.91 | 17.48 | 24.16 |
| 7 | 13.89 | 8.13 | 17.72 | 20.61 |
| 10 | 0 | 0 | 0 | 0 |

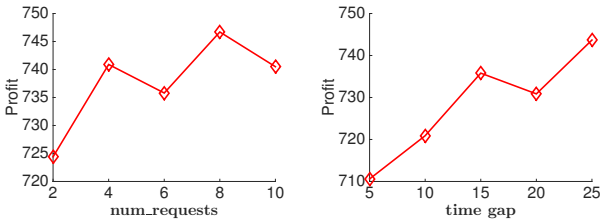
**Table 7: Result summary.**

| Setting | $t_{rnd}$ | | $\theta_{pt}$ | |
|---|---|---|---|---|
| | $\leq 5$ | $> 5$ | $< 900$ | $\geq 900$ |
| Preferred | Greedy | PBM | Greedy | PBM |

| Setting | $\theta_{dr}$ | | $bcost$ | |
|---|---|---|---|---|
| | $< 1.8$ | $\geq 1.8$ | $< 10$ | $\geq 10$ |
| Preferred | KM | PBM | Greedy | PBM |

$bcost = 10$, which indicates that no orders can be dispatched. This is because in the input, the vehicles are sparse and the order prices are not large compared with the base fee ($bcost = 10$ by default). Thus, we set lower base fares (from 3 to 7) and present the results in Table 6. It can be found that the profits of our proposed methods are at the same scale as that of OPT. Besides, even for the worst algorithm KM, its profit is at least around $\frac{1}{3}$ of that of OPT. For the running times, our proposed methods can finish within some milliseconds, while the brute-force OPT takes thousands of seconds to finish.

**Summary of the experimental results.** Overall, in terms of the achieved profit, PBM and Greedy perform more robustly than BM. We summarize the comparison results among our methods in Table 7. Note that when $t_{rnd} \leq 5$ $s$, only Greedy can satisfy the time requirement. When $t_{rnd} > 5$ $s$ (6 $s$), PBM (BM) can finish the dispatch safely within the time window.




(a) Varying number of requests (b) Varying the time gap

**Figure 10: Results of online method [15].**

Besides, through comparison with OPT in a small scenario, we show that the achieved profits of our methods are competitive.

### 7.2.4 *Comparison with Online Method*

In Section 7.1, we discuss how to adapt the online dispatch method [15] to maximize the profit. We now conduct evaluation of this approach under the simulation. We vary the maximum allowed number of repeated requests of an order from 2 to 10 to see the effect. Besides, we also examine the effect of the time gap between two adjacent requests of an order, by varying it from 5 $s$ to 25 $s$.

The results are shown in Figure 10(a) and 10(b). In Figure 10(a), the achieved profit generally goes up when the maximum number of requests per order is increased. However, even with 10 requests per order, the profit is still less than 750. This is far less than the profits of our round-based methods. Similarly, when we increase the time gap between two adjacent requests, the achieved profit generally increases, but remains below 750.

To explain the profit gap between our round-based methods and the online method [15], we calculate the amount of profit per order in results of PBM and [15], which are 11.16 and 4.90 $yuan$ respectively. This indicates that [15] makes distinctly worse dispatches compared to PBM. Its short-sight view of the orders may account for this. That is, it optimizes the current order without consideration of subsequent orders. This prevents it from making good

**Table 8: A toy example**

| $P_{i,j}$ | $v_1$ | $v_2$ |
|---|---|---|
| $o_1$ | 4 | 5 |
| $o_2$ | 2 | 10 |

dispatches from a global view. We give a simple example to help understand this. Suppose there are two manned vehicles $v_1$ and $v_2$, both with one remaining seat. Two orders $o_1$ and $o_2$ come one after another, and the dispatch profits are shown in Table 8. Since $o_1$ comes ahead of $o_2$, [15] would make the dispatch $(o_1, v_2)$ and $(o_2, v_1)$, whose profit is only half of that of $\{(o_1, v_1), (o_2, v_2)\}$.

# 8. RELATED WORK

**The car-hailing problem.** Under the round-based setting, order dispatch in unshared car-hailing degenerates to the bipartite matching problem [17], which can be solved in polynomial time. In contrast, car-hailing under the online setting has drawn increasing attention [26, 24, 29, 18], which is also named as the Online Maximum Weighted Bipartite Matching (OMWBM) problem. The goal is to maximize the utility sum of the matching, where the utilities can be defined according to the distances between vehicles and orders. [24, 29] and [18, 26] devise randomized algorithms with approximation ratios under the adversarial model and the random order model respectively. In the adversarial model, the approximation solutions are compared with the optimal solution in the worst case of tasks' arriving orders. In contrast, in the random order model, they are compared with the optimum under the average case of arriving orders. In addition to matching workers with tasks, Tong et al. [27] also utilize the guidance on workers' movement to further optimize the online task assignment.

**The dial-a-ride problem.** The typical formulation of a dial-a-ride problem (DARP) is to design travel routes for a given set of vehicles to pick up and drop off a set of customers w.r.t. their time requirements. It can treated as a constrained vehicle routing problem [19], and is NP-hard as well. The main difference between DARP and real-time ridesharing lies in that DARP commonly assumes all orders can be served. This cannot be true in real-time ridesharing due to the limited transportation capacity. The common objective in DARP is to minimize the overall travel distance. Integer programming formulations are widely adopted to find the optimal solution [8, 16]. However, only very small instances can be solved optimally within bearable amount of time costs. Therefore, heuristics have been proposed to find approximation solutions [10, 5]. These heuristics proceed in two stages. In the first stage, they cluster the orders and dispatch the clusters to vehicles. Then, in the second stage, they improve the dispatch by swapping the orders among vehicles. As summarized in [9], however, only small instances with up to hundreds of vehicles and orders have been tested on these algorithms.

The dynamic dial-a-ride problem [3, 7, 14] processes the orders in a stream. This is similar to the online ridesharing [20, 15]. However, [3, 7, 14] allow the dispatcher to exchange the dispatched orders among vehicles. This operation is clearly not supported in real-time ridesharing. In the current practice, an executed dispatch keeps intact unless the order is cancelled by the rider.

**Ridesharing.** [20, 15] study the real-time ridesharing under the online setting. In this setting, the streaming orders are processed by the dispatcher one by one, in the first-come-first-serve manner. A careful discussion of these two works is presented in Section 7.1. Another related online ridesharing work is [2], which also targets at maximizing the platform's profit, jointly considering orders' price and drivers' cost. In [2], on arrival of an order, each vehicle computes the profit of platform if it serves this order. The profits are reported to the platform as a bid, and the largest one would be picked. The key difference between [2] and our work is that the order price is upfront in our setting, and serves as an input of the dispatch process. In contrast, in [2], the price of an order is affected by the final dispatch result. A rider would pay less for a longer detour. One way to adapt [2] to solve our problem is to fix the order price, making it irrelevant to the detours. Then, the method in [2] would degenerate to minimizing the travel cost increase of drivers. This is similar to [20, 15], comparisons with which have been carried out in our experiments.

Alonsomora et al. [1] study the real-time ridesharing problem in the round-based setting. They optimize an objective function which linearly combines the the travel time delays (caused by sharing the ride), and the number of un-served orders. They construct a RTV graph which considers all potential packings of the orders, and solve the problem by integer programming methods. The problem is too complex to be solved optimally in real time, and the solution in [1] is to set timeouts to trade optimality for tractability. However, since the two penalty terms, time delays and the number of un-served orders, have different measures, it is unclear how to set the regularization terms. Cheng et al. [6] study the ridesharing problem with social affinity concerns. Their goal is to let riders with high social affinity take the same vehicle, to better their ride experience.

Gidofalvi et al. [13] propose efficient order packing algorithms for the real-time ridesharing problem, targeting at large-scale inputs. Their goal is to maximize riders' overall savings. They pack all order requests, which indicates that all orders would be served in their setting. In [22], each driver has its own travel plan, and would accept an order request if their shared travel distance is large enough. They formulate the problem as the maximum weighted bipartite graph matching problem, and propose techniques to accelerate the edge weight calculation process. [22] differs from our work in that their drivers have pre-defined travels and thus they do not consider packing orders. Wang et al. [28] study the activity-based ridesharing problem. Assuming the availability of users' daily activities, they allow more than one destinations for the activities as longs as they are functionally similar.

# 9. CONCLUSION

In this paper, we study the order dispatch problem in price-aware ridesharing. We formulate the problem of profit maximization with constraints on detour ratios and waiting times. To the best of our knowledge, this is the first work that has taken order prices into concern during order dispatch. We show that the problem is NP-hard, thus, we propose approximation methods, including the Greedy method, the bigraph matching method, and the packing based matching method. We develop a simulation framework using real ridesharing data, and evaluate the effectiveness and efficiency of the proposed methods with this simulation. Through comparison with the state-of-art method and the optimal solution, we show that our methods are competitive and effective.

# 10. ACKNOWLEDGMENT

The work is partially supported by the Hong Kong RGC GRF Project 16207617, National Grand Fundamental Research 973 Program of China under Grant 2014CB340303, the National Science Foundation of China (NSFC) under Grant No. 61729201, Science and Technology Planning Project of Guangdong Province, China, No. 2015B010110006, and Microsoft Research Asia Collaborative Research Grant. Data retrieved from Didi Chuxing.

## 11. REFERENCES


[1] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proc. of the NAS*, page 201611675, 2017.

[2] M. Asghari, D. Deng, C. Shahabi, U. Demiryurek, and Y. Li. Price-aware real-time ride-sharing at scale: an auction-based approach. In *Proc. of the SIGSPATIAL GIS*, page 3. ACM, 2016.

[3] A. Attanasio, J.-F. Cordeau, G. Ghiani, and G. Laporte. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, 30(3):377–387, 2004.

[4] M. Bellmore and G. L. Nemhauser. The traveling salesman problem: a survey. *Operations Research*, 16(3):538–558, 1968.

[5] R. W. Calvo and A. Colorni. An effective and fast heuristic for the dial-a-ride problem. *4OR - A Quarterly Journal of Operations Research*, 5(1):61–73, 2007.

[6] P. Cheng, H. Xin, and L. Chen. Utility-aware ridesharing on road networks. In *Proc. of the SIGMOD*, pages 1197–1210, 2017.

[7] A. Colorni and G. Righini. Modeling and optimizing dynamic dial-a-ride problems. *International transactions in operational research*, 8(2):155–166, 2001.

[8] J.-F. Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586, 2006.

[9] J.-F. Cordeau and G. Laporte. The dial-a-ride problem (darp): Variants, modeling issues and algorithms. *4OR - A Quarterly Journal of Operations Research*, 1(2):89–101, 2003.

[10] J.-F. Cordeau and G. Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579–594, 2003.

[11] Z. Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Computing Surveys*, 18(1):23–38, 1986.

[12] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. *Experimental Algorithms*, pages 319–333, 2008.

[13] G. Gidofalvi, T. B. Pedersen, T. Risch, and E. Zeitler. Highly scalable trip grouping for large-scale collective transportation systems. In *Proc. of the EDBT*, pages 678–689, 2008.

[14] M. E. Horn. Fleet scheduling and dispatching for demand-responsive passenger services. *Transportation Research Part C: Emerging Technologies*, 10(1):35–63, 2002.

[15] Y. Huang, F. Bastani, R. Jin, and X. S. Wang. Large scale real-time ridesharing with service guarantee on road networks. *PVLDB*, 7(14):2017–2028, 2014.

[16] L. M. Hvattum, A. Løkketangen, and G. Laporte. A branch-and-regret heuristic for stochastic and dynamic vehicle routing problems. *Networks*, 49(4):330–340, 2007.

[17] M.-Y. Kao. Weighted bipartite matching. *Encyclopedia of Algorithms*, pages 1020–1020, 2008.

[18] T. Kesselheim, K. Radke, A. Tönnis, and B. Vöcking. An optimal online algorithm for weighted bipartite matching and extensions to combinatorial auctions. In *European Symposium on Algorithms*, pages 589–600, 2013.

[19] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European journal of operational research*, 59(3):345–358, 1992.

[20] S. Ma, Y. Zheng, and O. Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *Proc. of the ICDE*, pages 410–421, 2013.

[21] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.

[22] T. Na, G. Li, T. Zhao, J. Feng, H. Ma, and Z. Gong. An efficient ride-sharing framework for maximizing shared route. *IEEE Transactions on Knowledge & Data Engineering*, PP(99):1–1, 2017.

[23] F. Rubin. A search procedure for hamilton paths and circuits. *Journal of the ACM (JACM)*, 21(4):576–580, 1974.

[24] H.-F. Ting and X. Xiang. Near optimal algorithms for online maximum edge-weighted b-matching and two-sided vertex-weighted b-matching. *Theoretical Computer Science*, 607:247–256, 2015.

[25] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu. Online minimum matching in real-time spatial data: experiments and analysis. *PVLDB*, 9(12):1053–1064, 2016.

[26] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen. Online mobile micro-task allocation in spatial crowdsourcing. In *Proc. of the ICDE*, pages 49–60, 2016.

[27] Y. Tong, L. Wang, Z. Zhou, B. Ding, L. Chen, J. Ye, and K. Xu. Flexible online task assignment in real-time spatial data. *PVLDB*, 10(11):1334–1345, 2017.

[28] Y. Wang, R. Kutadinata, and S. Winter. Activity-based ridesharing: increasing flexibility by time geography. In *Proc. of the SIGSPATIAL GIS*, page 1. ACM, 2016.

[29] Y. Wang and S. C.-w. Wong. Two-sided online bipartite matching and vertex cover: Beating the greedy algorithm. In *Proc. of the ICALP*, pages 1070–1081, 2015.

[30] L. Zhang, T. Hu, Y. Min, G. Wu, J. Zhang, P. Feng, P. Gong, and J. Ye. A taxi order dispatch model based on comnatorial optimization. In *Proc of the SIGKDD*, pages 2151–2159, 2017.

[31] L. Zheng and L. Chen. Maximizing acceptance in rejection-aware spatial crowdsourcing. *IEEE Transactions on Knowledge and Data Engineering*, 2017.