Joint Order Dispatching and Vehicle Repositioning for Dynamic Ridesharing

Zhidan Liu, *Member, IEEE,* Guofeng Ouyang, Bolin Zhang, Bo Du, Chao Chen, *Senior Member, IEEE,* and Kaishun Wu, *Fellow, IEEE*

Abstract—Dynamic ridesharing has gained significant attention in recent years. However, existing ridesharing studies often focus on optimizing order dispatching and vehicle repositioning separately, leading to short-sighted decisions and underutilization of the ridesharing potential. In this paper, we propose a novel joint optimization framework called `JODR`. By coordinating order dispatching and vehicle repositioning, `JODR` enhances ridesharing efficiency while ensuring high-quality service. The core idea of `JODR` is to dispatch ride orders with high demand in specific mobility directions to vehicles with sufficient available capacity, effectively balancing future supply and demand in those directions. To achieve this, we introduce a novel mobility value function that can predict the long-term mobility value of matching an order with its travel direction. By considering orders' directional mobility values, service quality assessments, and available vehicle capacities, `JODR` formulates the order dispatching as a minimum-cost maximum-flow problem to derive the optimal order-vehicle assignments. Furthermore, the value function helps the intelligent repositioning of idle vehicles. Extensive experiments conducted on a large real-world dataset demonstrate the superiority of `JODR` over state-of-the-art methods across various performance metrics. These experimental results validate the effectiveness of `JODR` in improving the ridesharing efficiency and experience.

Index Terms—Dynamic ridesharing, order dispatching, vehicle repositioning, mobility value

1 Introduction

DYNAMIC ridesharing services, offered by platforms like Uber [2] and Didi Chuxing [1], allow multiple riders with similar itineraries and schedules to share a single vehicle to reach their respective destinations. With the wide adoption of dynamic ridesharing, this form of shared mobility has fundamentally transformed urban transportation, offering convenient and efficient alternatives to traditional modes of travel. Moreover, it brings forth various advantages for urban cities, such as alleviating traffic congestion and reducing energy consumption [52].

The fundamental goal of ridesharing systems is to efficiently connect available vehicles (*i.e.*, *supply*) and spatio-temporally distributed orders (*i.e.*, *demand*), so that to maximize the number of served orders while guaranteeing the service quality. If a ridesharing system can maintains supply-demand equilibrium for the long term, the vehicle supply could well serve all orders in time, thereby maximizing the order completion rate. Such an objective is mainly determined by two essential tasks, *i.e.*, *order dispatching* and *vehicle repositioning*, both of which have great influences on the balance of supply and demand [42]. Specifically, order dispatching aims to match orders with suitable vehicles, where each vehicle owns remaining capacity for sharing and the riders already in this vehicle have similar travel route with the matched new order [24], [31]. Intuitively, the decision of each order dispatching directly impacts the spatial distribution of vehicle supply, which in turn affects the outcome of future order dispatching decisions. On the other hand, vehicle repositioning involves relocating idle or underutilized vehicles to the areas where they are more likely to receive future orders. This approach can potentially help reduce riders' waiting time, increase vehicle utilization, and improve overall system efficiency [19]. Therefore, order dispatching and vehicle repositioning are intricately linked in dynamic ridesharing where the orders dynamically appear. Effective coordination between these two tasks plays a pivotal role in guaranteeing a seamless and optimized ridesharing experience for both riders and drivers.

Despite numerous research efforts aimed at enhancing dynamic ridesharing, the majority of these works tend to optimize the two tasks separately. Some existing studies focus on improving order dispatching performance by optimizing specific objectives [4], [17], [24], [32], [43], [44], while others concentrate on developing vehicle repositioning strategies that redistribute vacant vehicles based on perceived supply and demand [19], [26], [37], [53], [57]. However, the former works often overlook the long-term impact of each order dispatching decision and neglect the crucial role of vehicle repositioning in balancing supply and demand. Meanwhile, the latter works primarily consider repositioning idle vehicles and overlook the redistribution of underutilized vehicles. Consequently, previous approaches tend to make short-sighted decisions, failing to unleash the full potential of ridesharing. Although some works [5], [12], [40] con-

- *Zhidan Liu is with Intelligent Transportation Thrust, System Hub, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China, 511453. (E-mail: zhidanliu@hkust-gz.edu.cn)*
- *Guofeng Ouyang and Bolin Zhang are with College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China, 518060. (E-mails: {ouyangguofeng2021, zhangbolin2023}@email.szu.edu.cn)*
- *Bo Du is with Griffith Business School, Griffith University, Queensland, Australia. (E-mail: bo.du@griffith.edu.au)*
- *Chao Chen is with State Key Laboratory of Mechanical Transmission, Chongqing University, Chongqing, China, 400044. (E-mail: cschaochen@cqu.edu.cn)*
- *Kaishun Wu is with DSA Thrust and IoT Thrust, Information Hub, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China, 511453. (E-mail: wuks@hkust-gz.edu.cn)*

sider joint optimization of order dispatching and vehicle repositioning, focusing on ride-hailing services where each vehicle serves one order. However, dynamic ridesharing is much more complex, as vehicles can serve multiple orders simultaneously, making these methods inapplicable.

To this end, we propose JODR, a Joint Order Dispatching and vehicle Repositioning framework. JODR is designed to optimize ridesharing efficiency while ensuring high-quality service. By utilizing a batch-based processing model, JODR collects a set of orders within a sliding time window and then dispatches them to suitable vehicles, effectively coordinating tasks of order dispatching and vehicle repositioning.

The core idea of JODR revolves around dispatching orders with high demand in specific travel directions to these suitable vehicles with sufficient capacity. By doing so, JODR aims to balance future supply and demand in those directions. During the order dispatching process, several factors are considered, including future demand in the order's travel direction, additional detour and waiting time caused by ridesharing, and the remaining capacity of each vehicle. This approach enables implicit repositioning of vehicle supply through deliberate order dispatching. A key element enabling this idea is the novel mobility value function $\mathtt{V}(\cdot)$. To model directional state changes in ridesharing activities, we employ the Markov Decision Process framework. Specifically, we utilize a deep reinforcement learning model's value network [35] to approximate the function $\mathtt{V}(\cdot)$, which predicts the long-term mobility value of matching an order in its travel direction. Furthermore, we formulate the order-vehicle matching problem as a *minimum-cost maximum-flow* problem in network flow [8]. By applying optimization methods, we can derive the optimal order-vehicle assignments. Moreover, leveraging the insight of function $\mathtt{V}(\cdot)$ on identifying valuable travel directions, we enhance supply-demand equilibrium by explicitly repositioning idle vehicles towards directions with anticipated high future demand.

In summary, we make the following key contributions:

- We analyze the limitations of existing ridesharing studies, and formulate the joint optimization problem of order dispatching and vehicle reposition in dynamic ridesharing.
- We propose JODR, an innovative solution that effectively coordinates the two tasks of order dispatching and vehicle repositioning. By achieving a supply-demand equilibrium, JODR enhances ridesharing efficiency while ensuring high-quality service.
- To implement JODR, we introduce a novel mobility value function capable of predicting the potential mobility values for different travel directions. We further model the order dispatching as a minimum-cost maximum-flow problem to determine the optimal order-vehicle assignments.
- We conduct extensive experiments on a large real-world dataset to investigate the efficiency and effectiveness of JODR under various comparison and parameter settings. Experimental results demonstrate that JODR significantly outperforms state-of-the-art methods across a wide range of performance metrics.

The rest of this paper is organized as follows. Section 2 reviews the related works. Section 3 introduces the preliminary of ridesharing and the problem statement. We elaborate and evaluate JODR's design in Section 4 and Section 5, respectively. Finally, Section 6 concludes this paper.

2 Related Work

Unlike traditional static ridesharing, *a.k.a.* carpooling [55], where the information of both riders and shared vehicles is known in advance, dynamic ridesharing has to serve dynamically appearing ride orders by properly arranging shared vehicles [17], [24], [31], [43], [44]. Dynamic ridesharing is more aligned with real-world shared mobility applications, hence is our main focus. In this paper, we will particularly review the related works from two aspects, *i.e.*, *order dispatching* and *vehicle repositioning*, which are the two major operations on determining the efficiency of ridesharing.

Order dispatching. Compared to ride-hailing that generally provides on-demand service for individual orders [39], [42], order dispatching in dynamic ridesharing is more challenging, as it involves multiple orders sharing a single vehicle to travel together and split the cost of the trip [18]. By matching orders with available vehicles, order dispatching aims to optimize the overall system efficiency while satisfying the requirements of all riders in terms of waiting time, detour costs, and the deadlines of arriving at destinations [4], [24], [25], [31], [32], [43], [44]. Existing works on ridesharing order dispatching can be categorized into *real-time* and *batch-based* methods.

Based on the first-come-first-served principle, real-time solutions assigns each order to an available vehicle right upon receiving the order [17], [24], [25], [29], [30], [31], [32], [43], [46]. For each newly coming order, these methods first search a set of candidate vehicles, and then determine the best vehicle to serve this order based on some objectives [27]. For example, Zheng *et al*. [31], [32] present *T-Share*, which firstly searches the candidate vehicles through grid index and then dispatches this order to the one with the minimum increased travel distance. To improve *T-Share*, Ma *et al*. [30] further incorporates quality-of-service as the constraint for order dispatching. In addition, Liu *et al*. [24], [25] develop a mobility-aware ridesharing system called *mT-Share*. It constructs index structures for orders and vehicles by exploiting both geographical information and travel directions to refine candidate searching, and further optimizes the route planning to improve computation efficiency. Inserting an order's origin and destination into a vehicle's route without changing its current schedule is a key operation for dynamic ridesharing. Tong *et al*. [44] thus propose an insertion operator that uses a greedy strategy for route planning, and further improve the insertion operation by considering both online and predicted orders [43]. In particular, Wang *et al*. [46] have explored a special ridesharing scenario with meeting points. While real-time methods can swiftly respond to each order, they often face limitations in accessing timely information during the decision-making stage. As a result, these methods may overlook closely timed information about orders and vehicles, leading to missed opportunities for making optimal decisions.

Different from the real-time solutions, batch-based methods [4], [6], [59], [60] instead wait for a certain time interval, which decides the batch size, before matching received orders with available vehicles. Alonso-Mora *et al.* [4] propose a two-step approach that firstly groups orders that can be shared, and then assigns each group to an available vehicle using the bipartite matching algorithm by minimizing the total travel distance while serving all orders. In fact, the bipartite matching algorithm has been adopted by various batch-based methods, yet with different objectives, *e.g.*, maximizing the platform's revenue [59], [60], or minimizing total travel distances [6]. Nevertheless, these methods typically involve enumerating all possible groups of orders and validating the feasibility of each order group's matching with an available vehicle. This process incurs high time complexity and significant response delays. The batch size largely determines the computation efficiency [43]. Different from these works, we divide a batch of orders based on their travel directions, which can reduce computation overheads greatly. Moreover, by jointly considering both order assignments and vehicle repositioning, we achieve substantial enhancements in ridesharing performance.

Vehicle repositioning. This task primarily focuses on optimizing the distribution and relocation of idle or underutilized vehicles to areas with high future demand [38]. We classify existing vehicle repositioning works into two categories, namely *explicit* and *implicit* methods. Specifically, explicit methods proactively dispatch idle vehicles across different areas, while implicit methods indirectly redistribute idle or underutilized vehicles through demand-aware order dispatching.

For explicit vehicle repositioning, traditional methods usually focus on recommending cruising routes [23] or popular locations [54] for vacant taxis, where they may find passengers easily. With the wide availability of mobility data, numerous data-driven approaches have been proposed. These works build supply and demand models by using historical data [33], and exploit different techniques, *e.g.*, receding horizon control [34] or combinatorial optimization [51], to dispatch idle vehicles based on their real-time locations and the predictions from the trained supply/demand models.

Recently, many works tend to study the system-level vehicle repositioning for better fleet managements [38]. In particular, the majority of these works exploit deep reinforcement learning [35] to derive model-free vehicle repositioning policies [13], [14], [15], [16], [19], [22], [26], [37], [48], [49], [50], [53], [57], [58]. For example, Lin *et al.* [19] propose a contextual multi-agent reinforcement learning framework to achieve explicit coordination among vehicles. In addition, Liu *et al.* [26] present a context-aware vehicle repositioning model by considering rich traffic contexts, *e.g.*, road connectivity and external factors. Our work differs from these works by incorporating both explicit and implicit vehicle repositioning strategies to a sustainable supply-demand balance over the long term.

There exist much fewer works that have considered implicit vehicle repositioning in ridesharing [3], [11], [20], [21], [24], [25], [43]. For example, Lin *et al.* [21] develop a probabilistic demand-aware framework for order-vehicle assignments and routing, aiming to maximize expected number of served orders given the probability distributions of future demand. In addition, *mT-Share* [24], [25] calculates a probability map from historical order distributions and proposes a probabilistic routing to guide shared vehicles to meet potential riders. Similarly, Al-Abbasi *et al.* [3] incorporate historical travel demand and deep learning models to implicitly dispatch vehicles. Different from them, we achieve fine-grained implicit vehicle repositioning by considering the travel directions of orders.

TABLE 1: Summary of key notations.

Notation	Description
$\mathcal{G}_r =< \mathcal{V}_r, \mathcal{E}_r >$	The directed graph of a road network
$\mathbb{G}$	A set of grids for the road network $\mathcal{G}_r$
r	A ride order
w	A shared vehicle
Δ_{sw}	The size of a sliding time window
$\mathtt{V}(\cdot)$	A value function
$\vec{\mathtt{v}}$	A mobility vector
$\mathtt{C}$	A mobility cluster
$\mathbb{T}_r$	A set of candidate vehicles for order r
$\vec{\mathtt{v}}^g$	The grid-level mobility vector
$\mathcal{G}_f =< \mathcal{V}_f, \mathcal{E}_f >$	The flow network
$U(i,j)$	The capacity of edge (i,j) in flow network
$C(i,j)$	The cost of edge (i,j) in flow network

Furthermore, Ge *et al.* [10] utilize elastic dummy orders to promote or restrain ridesharing through order association dispatching, in hope of achieving the supply-demand equilibrium. Guo *et al.* [11] integrate vehicle routing and order-vehicle assignments to optimize both operation cost and service quality for autonomous mobility-on-demand systems. Zhou *et al.* [56] present a robust optimization based joint order dispatch and repositioning framework, while it is applied for car-sharing rather than ridesharing. Our work differs from these works by jointly considering order dispatching and vehicle repositioning to improve the efficiency and experience of dynamic ridesharing.

3 Problem Statement

In this section, we first present some definitions, and then motivate our work with vivid examples. Finally, we formulate the joint optimization problem of order dispatching and vehicle repositioning in the dynamic ridesharing scenario. Table 1 presents the key notations used in this paper.

3.1 Definitions

Definition 1. (**Road Network**) *A road network is denoted by a directed graph $\mathcal{G}_r =< \mathcal{V}_r, \mathcal{E}_r >$ with a vertex set $\mathcal{V}_r$ and an edge set $\mathcal{E}_r$. Each edge $(u,v) \in \mathcal{E}_r$ is associated with a weight, indicating the travel time between vertex u and v.*

To enhance order dispatching and vehicle repositioning, the platform typically segments the road network into distinct zones. Instead of utilizing complex clustering algorithms to create these zones, we employ a simple yet effective method that organizes the road network into uniform grids as a set $\mathbb{G}$, similar as previous works [32], [43], [44].

Definition 2. (**Ride Order**) *A ride order is represented as $r =< t_r, o_r, d_r, e_r, c_r >$, where vertex $o_r \in \mathcal{V}_r$ and $d_r \in \mathcal{V}_r$ are the origin and destination of the ride order, respectively. In addition, order r is released at time t_r with size c_r, and ought to be fulfilled ahead time e_r by delivering the c_r riders of order r from origin o_r to destination d_r.*

Definition 3. (**Shared Vehicle**) *A shared vehicle is represented as $w =< \ell_w, S_w, C_w >$, where ℓ_w indicates the current*

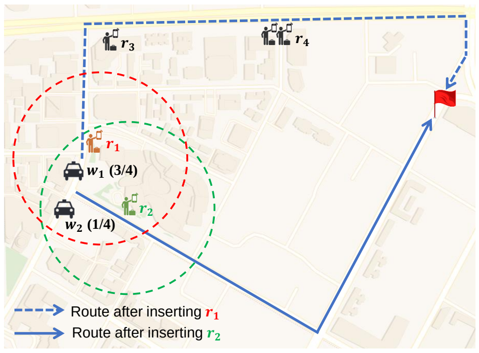


Fig. 1: A motivation example, where dashed circles represent the searching range, and the number a/c alongside each shared vehicle w indicates the number a of riders already on board and the capacity c of vehicle w.

location of vehicle w, while S_w and C_w are the schedule and capacity of vehicle w, respectively.

In practice, a shared vehicle w is guided by the ridesharing platform to serve suitable ride orders, following a well-planned schedule S_w. In addition, we denote $\mathbb{R}_w$ the set of ride orders currently being served by vehicle w.

Definition 4. (**Schedule**) *The schedule of a shared vehicle w is denoted as $S_w = (\ell_w, \ell_1, \cdots, \ell_n)$, which comprises an ordered sequence of origins and destinations of orders in $\mathbb{R}_w$. A schedule is **valid** if (i) $\forall r \in \mathbb{R}_w$, o_r should precede d_r in the schedule; (ii) $\forall r \in \mathbb{R}_w$, vehicle w should deliver riders of order r to destination d_r no later than the deadline e_r; (iii) At any time, the total number of riders does not exceed the capacity C_w of vehicle w, i.e., $\sum_{r\in\mathbb{R}_w} c_r \leq C_w$.*

In real-world, orders usually come in a streaming fashion. Previous studies have demonstrated that the batch-based order dispatching methods generally perform better than these real-time based methods on finding the optimal order-vehicle assignments [4], [47]. Therefore, in this study we adopt the batch-based processing model, which collects a batch of ride orders within a sliding time window of size Δ_{sw} and then dispatches them to available vehicles.

3.2 Motivation

Despite the extensive research on ridesharing, the majority of previous studies tend to focus on the separate optimization of two crucial tasks: order dispatching and vehicle repositioning. However, this isolated approach fails to achieve a sustainable supply-demand equilibrium in the long run, which is essential for optimizing ridesharing efficiency. Most previous order dispatching methods exhibit a myopic perspective, while vehicle repositioning methods often underutilize the capacities of all shared vehicles.

We explain above arguments using an example as shown in Figure 1. Assume that two orders, *i.e.*, r_1 and r_2, have been sequentially received by the ridesharing platform within the same sliding time window, and they have the same destination, marked by a red flag in Figure 1. Within their searching range, there are two candidate vehicles, *i.e.*, w_1 and w_2, whose route, indicated by the solid blue line, also terminates at the red flag. Both vehicles have the same capacity as 4, while there are already 3 riders in vehicle w_1 and 1 rider in vehicle w_2. To serve order r_1, a vehicle's route has to be rescheduled to the blue dashed line. While serving order r_2 does not alter the vehicle's route, remaining as the solid blue line.

For the real-time order dispatching methods, since order r_1 comes first, it is prioritized. By optimizing some metric (*e.g.*, minimizing the waiting time), r_1 might be assigned to vehicle w_1, and r_2 would be matched with vehicle w_2. While for the batch-based methods, since both orders fall within the same time window, existing methods generally generate some potential order groups (*e.g.*, $\{r_1\}$, $\{r_2\}$, $\{r_1, r_2\}$ in this example), and then validate each group by trying to insert its orders into each vehicle's schedule. Although batch-based methods may find the optimal order-vehicle assignments, they usually incur extensive computations, leading to delayed responses.

Some studies demonstrate that it is necessary to take future travel demand into consideration when dispatching the current orders to vehicles [20], [21], [24], [43]. Figure 1 shows that there are two newly coming orders, *i.e.*, r_3 with 1 rider and r_4 with 2 riders, along the blue dashed line. As a result, if dispatching vehicle w_1 to serve order r_1, as guided by existing methods, vehicle w_1 cannot serve the subsequent orders r_3 and r_4 due to its full load. In this case, the best solution turn to be that we dispatch w_2 serve r_1 and w_1 to serve r_2. As vehicle w_2 has sufficient available capacity, it could successfully serve both orders r_3 and r_4 later. This arrangement maximizes the utilization of shared vehicles. However, it relies heavily on the availability of information regarding vehicles and orders, including both current data and future potential demand.

Hence, it is reasonable to expect the existence of a value function $\mathtt{V}(\cdot)$ that can predict the future demand value of matching an order, thereby facilitating intelligent coordination between order dispatching and vehicle repositioning. Previous approaches have implicitly approximated the function $\mathtt{V}(\cdot)$ by calculating the probability of meeting future orders [11], [20], [21], [23], [24], [25], [43], [54]. However, such probabilistic methods tend to be inefficient. This is because, from a vehicle's perspective, relying solely on the probabilities does not provide sufficient information to determine whether future orders can be shared with the current order and vehicle or not.

Unlike previous works, this paper takes a different approach by considering the value of an order's travel direction. This choice is motivated by several factors. Firstly, urban mobility exhibits distinct origin-destination patterns, as highlighted in studies on human mobility [41]. These patterns indirectly classify travel demand into different mobility directions. Secondly, previous ridesharing research [24], [25] has shown that incorporating mobility information, such as travel directions, is crucial for optimizing order-vehicle matching. Feasible and optimal matches are more likely to occur between orders and vehicles traveling in similar directions. Therefore, a value function $\mathtt{V}(\cdot)$ that can predict the potential mobility values of different travel directions can provide valuable guidance for both tasks. By following the suggestions provided by $\mathtt{V}(\cdot)$, the ridesharing platform can not only dispatch current orders to vehicles but also implicitly adjust the distribution of vehicle supply to better cater to future travel demand.

3.3 Problem Definition

To improve the long-term system efficiency, we consider the joint optimization problem of order dispatching and vehicle reposition (*JODR* for short) in dynamic ridesharing with the assistance of potential value function $\mathtt{V}(\cdot)$.

Definition 5. (**JODR Problem**) *Given a set $\mathbb{W}$ of vehicles and a set $\mathbb{R}$ of ride orders that dynamically arrive within a sliding time window on a road network $\mathcal{G}_r$, by leveraging a value function $\mathtt{V}(\cdot)$, the JODR problem aims to find the optimal matching of ride orders and vehicles, such that the total future value $\mathcal{F}(\mathbb{R},\mathbb{W})$ is maximized, i.e.,*

$$\mathcal{F}(\mathbb{R},\mathbb{W}) = \max_{(r,w)} \sum_{r\in\mathbb{R}} \sum_{w\in\mathbb{W}} \mathtt{V}((r,w)), \tag{1}$$

subject to each optimal matching (r,w) meets the feasibility condition that the schedule S_w for w serving r is valid.

The order dispatching problem in dynamic ridesharing has been proven to be NP-hard in previous studies [6], [43], [44]. Consequently, the *JODR* problem, which involves dispatching both current and future orders, can be even more challenging. Addressing this problem involves tackling several challenges:

Firstly, the batch-based order dispatching model, while capable of finding optimal order-vehicle assignments, often leads to huge computation overheads. Improving computation efficiency is crucial for providing high-quality service.

Secondly, although the idea of value function $\mathtt{V}(\cdot)$ is appealing, instantiating such a function is challenging due to dynamic nature of ridesharing. The multitude of possible travel directions further complicates the design of $\mathtt{V}(\cdot)$.

Thirdly, effectively managing available vehicles with varying remaining capacities to fulfill a batch of orders, each associated with distinct potential demand values, is a complex task. Achieving the optimal solution necessitates comprehensive modeling of both orders and vehicles.

4 Design of JODR

In this section, we first present the overview of JODR design, and then elaborate the design of each module.

4.1 Overview

JODR operates by utilizing real-time information of shared vehicles and ride orders to dispatch and redistribute available vehicles for serving dynamically arriving orders. The main objective of JODR is to provide quality guaranteed ridesharing service while balancing vehicle supply and travel demand. Figure 2 illustrates the framework of JODR, consisting of three major modules, namely *Candidate Matching*, *Mobility Value Function*, and *Value Aware Dispatcher*.

- The *Candidate Matching* module searches a set of candidate vehicles for each order that is received within current sliding time window by leveraging mobility information and geographical locations of both vehicles and orders. A pair (r,w) of ride order r and candidate shared vehicle w is termed as a *matching* that assumes w could serve r. JODR assesses the service quality of matching (r,w) by spuriously inserting r into w's schedule S_w to calculate the

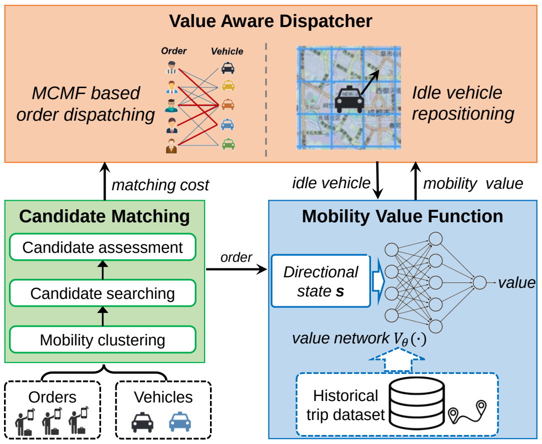


Fig. 2: The framework of JODR.

matching cost measured by the sum of additional detour time and waiting time cased by ridesharing.
- To predict the long-term mobility value of a specific travel direction, JODR leverages a deep reinforcement learning (DRL) model's value network in its *Mobility Value Function* module. In practical usage, this module utilizes mobility information from an order or a shared vehicle to generate the *directional state*. This state is then fed into the value network of the DRL model to produce a value, representing the predicted long-term mobility value associated with that particular travel direction. The value network serves as the value function $\mathtt{V}(\cdot)$ and is derived through offline training on historical ridesharing trip data.
- Given all possible matchings and their respective costs and mobility values, the *Value Aware Dispatcher* module will dispatch each order to the most suitable vehicle over the long run. To this end, JODR formulates the order dispatching as a minimum-cost maximum-flow (*MCMF*) problem, and solves it via optimization methods. Furthermore, this module explicitly redistributes idle vehicles to neighboring grids under the guidance of value function $\mathtt{V}(\cdot)$, anticipating to serve future orders quickly.

4.2 Candidate Matching

Compared with real-time order dispatching, the batch-based solutions involve more orders and vehicles to be processed, incurring much more computations. To speed up the response on orders, JODR refines the set of candidate vehicles for each order and evaluates each possible matching of an order and an available vehicle. To this end, JODR divides a batch of orders and available vehicles into multiple clusters based on their travel directions. We refer to such a cluster as *mobility cluster*, as it is formed by leveraging the mobility information of orders and vehicles. Then, JODR determines the set of candidate vehicles for each order within the same mobility cluster by exploiting their geographic locations. Lastly, JODR investigates possible vehicle schedules to calculate the detour time and waiting time for each matching.

1) Mobility clustering: Intuitively, riders with similar travel directions could share a vehicle, while riders who have distinct travel directions should never be considered for ridesharing as they will introduce significant detours.

Therefore, we think that mobility information is essential for finding the most suitable vehicle to serve an order. Based on this intuition, we propose the concept of *mobility vector* that facilitates the division of a batch of orders and available vehicles into mobility clusters.

Definition 6. (**Mobility Vector**) *Mobility vector $\vec{\mathtt{v}}$ is a vector representation of an object's travel direction, pointing from the origin to the destination.*

Therefore, mobility vector $\vec{\mathtt{v}}_r$ of a ride order r is represented using its origin o_r and destination d_r, *i.e.*, $\vec{\mathtt{v}}_r =< o_r, d_r >$. In addition, mobility vector $\vec{\mathtt{v}}_w$ of a vehicle w with riders can be derived by using vehicle w's current location ℓ_w as the origin and the average destination ℓ_{ad}, which is the geometric center of all riders' destinations, *i.e.*, $\vec{\mathtt{v}}_w =< \ell_w, \ell_{ad} >$.

To identify the ride orders that can share a vehicle in terms of travel direction, a straightforward approach is to applying some hierarchical agglomerative clustering algorithms [36] to group orders based on their mobility vectors. By initializing each order as an individual cluster, agglomerative clustering recursively merges pair of clusters in a bottom-up manner until a single cluster remains. This approach, however, is computationally expensive, and inevitably prolongs the response time.

Instead, we devise a simple yet efficient mobility clustering method. We arrange all orders according to their release time, and classify them into mobility clusters in a sequential manner. Specifically, the first incoming order forms the initial mobility cluster, and each subsequent order is either added to an existing mobility cluster or forms a new one by itself. For each mobility cluster $\mathtt{C}_i$, we calculate a *representative vector* $\vec{\mathtt{v}}_{\mathtt{C}_i}$, whose origin and destination are the average values of origins and destinations of all orders belonging to cluster $\mathtt{C}_i$, respectively. When a subsequent order r appears, we calculate the directional similarity between r's mobility vector $\vec{\mathtt{v}}_r$ and the representative vector $\vec{\mathtt{v}}_{\mathtt{C}_i}$ of each mobility cluster $\mathtt{C}_i$. To be specific, we employ *cosine similarity* to calculate the directional similarity α between $\vec{\mathtt{v}}_r$ and $\vec{\mathtt{v}}_{\mathtt{C}_i}$, *i.e.*,

$$\alpha = \frac{\vec{\mathtt{v}}_r \cdot \vec{\mathtt{v}}_{\mathtt{C}_i}}{||\vec{\mathtt{v}}_r|| \times ||\vec{\mathtt{v}}_{\mathtt{C}_i}||}. \tag{2}$$

For each order r, we calculate its directional similarity with all existing mobility clusters, and choose the mobility cluster $\mathtt{C}_*$, which has the maximum similarity α_* with r, for further consideration. If $\alpha_* > \lambda$ where λ is a predefined parameter, order r is considered to be highly similar to the travel direction of orders already present in cluster $\mathtt{C}_*$. In such a case, r is added to $\mathtt{C}_*$; Otherwise, r will form a new mobility cluster on its own. We repeat above operations for all orders in the batch until each order has been assigned to one mobility cluster.

After clustering the ride orders, we then assign the available vehicles to existing mobility clusters based on their own mobility vectors. For each available vehicle w, we also compute its directional similarity with all mobility clusters using cosine similarity, and add w to the cluster having the largest directional similarity with w. Note that we have no similarity constraint on adding vehicles into existing mobility clusters.

2) Candidate searching: For each order r, we determine a set $\mathbb{T}_r$ of candidate vehicles using constraints on both travel direction and geographic location. On the aspect of geographic location, we firstly determine order r's locating grid g_r, and then search candidate vehicles for r from g_r and g_r's neighboring grids, denoted as set $\mathbb{G}_r$. For each grid $g \in \mathbb{G}_r$, we retrieve a list $g.L_w$ of available vehicles, which are currently locating within grid g. On the aspect of travel direction, we firstly determine the mobility cluster $\mathtt{C}_r$ that contains order r, and then pick out the available vehicles in cluster $\mathtt{C}_r$, denoted as $\mathtt{C}_r.L_w$. Then we establish the set $\mathbb{T}_r$ of candidate vehicles for order r as follows:

$$\mathbb{T}_r = \{\cup_{g \in \mathbb{G}_r} g.L_w\} \cap \mathtt{C}_r.L_w. \tag{3}$$

Applying both directional and geographic constraints allows for the early elimination of invalid vehicles from the candidate set, resulting in significant computation cost reductions for the order dispatching task.

3) Candidate assessment: Considering the quality of service, we assess each matching (r, w) with a *matching cost*, which is estimated as the sum of additional detour time and waiting time incurred when vehicle w serves order r. To derive this cost, we employ an insertion algorithm proposed in [24] to calculate the detour time and waiting time for each matching (r, w). Briefly, by spuriously inserting origin o_r and destination d_r of order r into candidate vehicle w's schedule S_w, we enumerate all feasible new schedules that enable w to serve r, and calculate detour and waiting time for each feasible new schedule. Finally, we choose the minimum sum of detour and waiting time as the matching cost for matching (r, w).

4.3 Mobility Value Function

We consider the ridesharing platform as an agent that utilizes information from both orders and vehicles to achieve a supply-demand equilibrium through efficient execution of order dispatching and vehicle repositioning tasks. The decision-making process of the agent can be modeled as a Markov Decision Process (MDP), which provides a robust framework for solving decision problems in uncertain environments. By defining a set of states, actions, rewards, and state transitions, we formulate the MDP problem for both order dispatching and vehicle repositioning in the dynamic ridesharing context.

4.3.1 ***Directional state***

To prune the extremely large spatio-temporal space involved in ridesharing, we discrete the dimensions of both time and geographic locations. Specifically, considering the patterns exhibiting in human mobility across time of the day and day of the week [26], [41], [51], we divide the time of each day into a series of time frames with size of 5 minutes. In addition, we consider the *grid-level mobility vectors* for both orders and shared vehicles, so as to simplify the computational complexity. For example, an object (*e.g.*, an order or a vehicle) travels from grid 4 to gird 10, then its grid-level mobility vector is denoted as $\vec{\mathtt{v}}^g =< 4, 10 >$.

Different from previous works that mainly capture vehicle states [15], we propose *directional state* that encodes

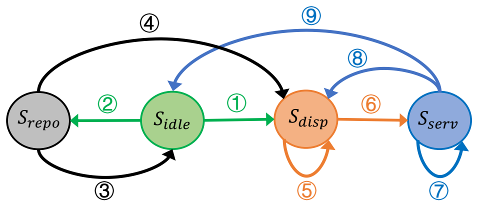


Fig. 3: The state transitions among four types of states.

ridesharing activity information on different travel directions, which are represented by grid-level mobility vectors. Specifically, we represent the directional state using a tuple as $\mathbf{s} =< \vec{\mathrm{v}}^g, tf, dw >$, where $\vec{\mathrm{v}}^g$ is a grid-level mobility vector constructed according to a given object's mobility information, tf is the time frame index, and dw is the day of the week when the state occurs. There exist totally four types of directional states as follows.

- *Dispatching state* indicates that along some given direction an order has just been dispatched with a vehicle. Given the assigned order r, we represent the state's grid-level mobility vector as $\vec{\mathrm{v}}^g =< g_{o_r}, g_{d_r} >$, where g_{o_r} and g_{d_r} are the grids where order r's origin o_r and destination d_r locate, respectively.
- *Serving state* indicates that along some given direction a vehicle w is now delivering on-board riders by following the predefined schedule. Hence, we represent the state's grid-level mobility vector as $\vec{\mathrm{v}}^g =< g_w, g_{ad} >$, where g_w is vehicle w's locating grid and g_{ad} is the grid where the average destination of all orders locates.
- *Idle state* indicates that along some given direction a vehicle w is now vacant, and the grid-level mobility vector of that travel direction is denoted as $\vec{\mathrm{v}}^g =< g_w, g_w >$.
- *Repositioning state* indicates that along some given direction a vehicle w is vacant and has been redistributed by the agent. We represent the travel direction using w's grid-level mobility vector as $\vec{\mathrm{v}}^g =< g_w, g_{tar} >$, where g_{tar} is the target grid determined by the agent.

The four kinds of directional states are denoted as set $\mathcal{S}_{disp}$, $\mathcal{S}_{serv}$, $\mathcal{S}_{idle}$, and $\mathcal{S}_{repo}$, respectively.

4.3.2 Action

Given current state, the agent may accordingly apply some action, either assigning an order to a vehicle or repositioning an idle vehicle. To optimize vehicle redistribution effectively, we restrict the target repositioning areas for a vehicle to include its current grid and the neighboring grids.

4.3.3 State transition

A state transition represents the transfer of previous state $\mathbf{s}_{pre}$ to current state $\mathbf{s}_{cur}$ after applying some action. Considering the actions that may affect vehicles' travel directions, there exist nine kinds of transitions among the four state types, as illustrated in Figure 3. For a given concerned travel direction, we describe these sate transitions as follows:

① $\mathcal{S}_{idle} \rightarrow \mathcal{S}_{disp}$: A vehicle traveling along the given direction is dispatched with a new order when it is idle;

② $\mathcal{S}_{idle} \rightarrow \mathcal{S}_{repo}$: If a vehicle has been vacant for a time frame, *i.e.*, 5 minutes, along the given travel direction, it will be redistributed to another direction by setting a target grid;

③ $\mathcal{S}_{repo} \rightarrow \mathcal{S}_{idle}$: A vehicle arrives at the redistributed target grid while there are no suitable orders, then the vehicle will become idle;

④ $\mathcal{S}_{repo} \rightarrow \mathcal{S}_{disp}$: A vehicle is traveling to the repositioning target grid along the given direction, while the agent dispatches a new order to this vehicle;

⑤ $\mathcal{S}_{disp} \rightarrow \mathcal{S}_{disp}$: A vehicle is consecutively dispatched with new orders within a time frame along the given direction;

⑥ $\mathcal{S}_{disp} \rightarrow \mathcal{S}_{serv}$: If no new order is dispatched within a time frame after picking up the previously assigned order, the vehicle will deliver on-board riders along the given direction;

⑦ $\mathcal{S}_{serv} \rightarrow \mathcal{S}_{serv}$: If no new order is dispatched to a vehicle that serves riders within a time frame, this vehicle will keep driving along the given travel direction;

⑧ $\mathcal{S}_{serv} \rightarrow \mathcal{S}_{disp}$: A vehicle is dispatched with a new order when it is serving orders along the given travel direction;

⑨ $\mathcal{S}_{serv} \rightarrow \mathcal{S}_{idle}$: After the vehicle has successfully delivered all riders to their destinations respectively, the vehicle will become idle and still drive along the given direction.

4.3.4 Reward

When a state is changed, it is rewarded accordingly. For the purpose of balancing supply and demand among different travel directions, we only set positive rewards for the following kinds of state transitions, *i.e.*, $\mathcal{S}_{idle} \rightarrow \mathcal{S}_{disp}$, $\mathcal{S}_{serv} \rightarrow \mathcal{S}_{disp}$, and $\mathcal{S}_{disp} \rightarrow \mathcal{S}_{disp}$, which can bring valuable changes to the ridesharing system, while giving zero reward to the rest of state transitions. The intuition behind these reward settings is to make the mobility value function $\mathtt{V}(\cdot)$ encourage proactive order dispatching and thus capture the potential long-term values of all grid-level mobility vectors.

In this paper, we employ *value network* of a deep reinforcement learning model [35] to serve as the mobility value function $\mathtt{V}(\cdot)$, which computes the long-term mobility value, *i.e.*, expected discounted reward, in some travel direction at any given time. By leveraging massive historical trip data that contains rich information of ridesharing orders and trips, we replay the ridesharing activities and collect all state transitions, denoted as $\mathcal{T}$. Later, we use these state transitions to train the value network. Specifically, we denote $\mathcal{T}_p$ as the set of state transitions receiving positive rewards, and $\mathcal{T}_z = \mathcal{T} \backslash \mathcal{T}_p$ is used to denote the set of state transitions with zero reward. For each state transition $\mathbf{s}_{pre} \rightarrow \mathbf{s}_{cur}$ in set $\mathcal{T}$, we update value network using the one-step temporal-difference (TD) method as:

$$\begin{cases} \mathtt{V}_\theta(\mathbf{s}_{cur}) = 1 + \gamma \mathtt{V}_\theta(\mathbf{s}_{pre}) & \forall\, (\mathbf{s}_{pre} \rightarrow \mathbf{s}_{cur}) \in \mathcal{T}_p, \\ \mathtt{V}_\theta(\mathbf{s}_{cur}) = 0 + \gamma \mathtt{V}_\theta(\mathbf{s}_{pre}) & \forall\, (\mathbf{s}_{pre} \rightarrow \mathbf{s}_{cur}) \in \mathcal{T}_z, \end{cases} \tag{4}$$

where γ is the discounted factor and θ represents the model parameters. Accordingly, we obtain the TD error as:

$$\delta = \begin{cases} 1 + \gamma \mathtt{V}_\theta(\mathbf{s}_{pre}) - \mathtt{V}_\theta(\mathbf{s}_{cur}) & \forall\, (\mathbf{s}_{pre} \rightarrow \mathbf{s}_{cur}) \in \mathcal{T}_p \\ \gamma \mathtt{V}_\theta(\mathbf{s}_{pre}) - \mathtt{V}_\theta(\mathbf{s}_{cur}) & \forall\, (\mathbf{s}_{pre} \rightarrow \mathbf{s}_{cur}) \in \mathcal{T}_z \end{cases} \tag{5}$$

Algorithm 1: Value network training

```
1  Input: transition set $\mathcal{T}$, discounted factor $\gamma$;
2  Initialize value network $\mathtt{V}_\theta$ with random parameter $\theta$;
3  Initialize target value network $\widehat{\mathtt{V}}_\theta$ with same parameter
     of $\mathtt{V}_\theta$;
4  for epi = 1 to max-episodes do
5      Sample a batch of samples ($\mathbf{s}_{pre}$, $\mathbf{s}_{cur}$, reward) from
         $\mathcal{T}$;
6      Estimate mobility value $\mathtt{V}_\theta(\mathbf{s}_{pre})$;
7      Estimate expected mobility value reward +
         $\gamma\mathtt{V}_\theta(\mathbf{s}_{cur})$;
8      Compute TD-error $\delta$ using Equation (5);
9      Update Q-network parameters using gradient
         descent to minimize the loss in Equation (6);
10     if time to update the target value network then
11         Update parameters of $\widehat{\mathtt{V}}_\theta$ to match parameters of
             $\mathtt{V}_\theta$;
```

Based on above one-step TD method, we update the value network $\mathtt{V}_\theta(\cdot)$ of the neural network approximation using the bootstrapping form of existing Deep Q-learning methods [35] by minimizing TD squared error for all state transitions, *i.e.*,

$$\min_\theta \mathcal{L}(\mathcal{T};\theta) = \sum_{(\mathbf{s}_{pre}\rightarrow\mathbf{s}_{cur})\in\mathcal{T}_p} (1+\gamma\mathtt{V}_\theta(\mathbf{s}_{pre}) - \mathtt{V}_\theta(\mathbf{s}_{cur})) + \sum_{(\mathbf{s}_{pre}\rightarrow\mathbf{s}_{cur})\in\mathcal{T}_z} (\gamma\mathtt{V}_\theta(\mathbf{s}_{pre}) - \mathtt{V}_\theta(\mathbf{s}_{cur})) \tag{6}$$

Algorithm 1 presents the whole training process for a value network $\mathtt{V}_\theta(\cdot)$. We adopt Double Q-learning [45] to resolve the overestimation problem and obtain better training stability. The training involves sampling transitions, estimating mobility values, and updating network parameters θ iteratively. A target network is periodically aligned to stabilize training and thereby enhance the network's ability to predict mobility values.

After model training, we treat the learned $\mathtt{V}_\theta(\cdot)$ as the mobility value function. By inputting a directional state s, function $\mathtt{V}_\theta(\mathbf{s})$ returns the long-term mobility value along that travel direction, which is employed for order dispatching and vehicle repositioning in the ridesharing context.

4.4 Value Aware Dispatcher

The dispatcher intelligently matches orders with suitable vehicles and redistributes idle vehicles to achieve a sustainable supply-demand equilibrium. Both of these tasks are facilitated by the utilization of mobility value function $\mathtt{V}_\theta(\cdot)$.

4.4.1 MCMF based order dispatching

Traditional ride-hailing works [39], [42] usually formulate the order dispatching as a bipartite matching problem, and attempt to maximize the utility by leveraging Kuhn-Munkras (KM) algorithm [9]. However, the KM algorithm can only match one order to one vehicle at a time, and as a result is not suitable for the order dispatching in batch-based ridesharing, where multiple orders within the same batch may still be served by one vehicle.

Different from prior works, we model the order-vehicle matching in ridesharing as a *minimum-cost maximum-flow* (MCMF) problem to maximize the capacity utilization of all shared vehicles. The MCMF problem is a classic optimization and decision problem in the graph theory, which aims to find the most cost-efficient way to transport flow through a network. Typically, the flow network can be represented as a directed graph $\mathcal{G}_f =< \mathcal{V}_f, \mathcal{E}_f >$ with a source vertex $s \in \mathcal{V}_f$ and a sink vertex $k \in \mathcal{V}_f$, and each edge $(i,j) \in \mathcal{E}_f$ is associated with a capacity and a cost. Formally, we formulate order dispatching problem in ridesharing as an MCMF problem as follows:

- **Vertices and edges.** Except the source s and sink k, the remaining vertices in the set $\mathcal{V}_f$ can be classified into two categories, *i.e.*, *order vertices* and *vehicle vertices*, which encompass all candidate vehicles for the orders of current batch. There exists a directed edge from source s to each order vertex, and these edges are denoted as source edges $\mathcal{E}_f^s$. Similarly, there exists a directed edge from each candidate vehicle vertex to the sink vertex k, and such edges are represented as sink edges $\mathcal{E}_f^k$. For each order-vehicle matching (r,w), there exists a directed edge from r's corresponding vertex to w's corresponding vertex. The set of edges between order vertices and vehicle vertices is denoted as $\mathcal{E}_f^{rw}$. In summary, given a batch of orders $\mathbb{R}$ and their candidate vehicle set $\mathbb{T}$, the vertices and edges in the flow network $\mathcal{G}_f$ are defined as:

$$\begin{aligned} \mathcal{E}_f^s &= \{(\mathrm{s}, r) | r \in \mathbb{R}\}, \\ \mathcal{E}_f^k &= \{(w,k) \mid w \in \mathbb{T}\}, \\ \mathcal{E}_f^{rw} &= \{(r,w) \mid r \in \mathbb{R};\, w \in \mathbb{T}\}, \\ \mathcal{E}_f &= \mathcal{E}_f^s \cup \mathcal{E}_f^k \cup \mathcal{E}_f^{rw}, \\ \mathcal{V}_f &= \mathbb{R} \cup \mathbb{T} \cup \{s\} \cup \{k\}. \end{aligned} \tag{7}$$

- **Edge capacity.** The capacity $U(i,j)$ of each edge $(i,j) \in \mathcal{E}_f$ represents the maximum flow that can pass through. In the ridesharing context, we set the capacity values for edges in $\mathcal{E}_f^s$ as $U(i,j) = 1$. This is because each order can only be matched once. In addition, the capacity values for edges in $\mathcal{E}_f^k$ are determined based on the respective candidate vehicle's remaining seat amount, implying the number of orders the vehicle can be matched with. Lastly, the capacity values for edges $\mathcal{E}_f^{rw}$ are set to 1, indicating that each order can be matched with only one vehicle. Therefore, the edge capacity in $\mathcal{G}_f$ is summarized as:

$$U(i,j) = \begin{cases} 1 & \forall e = (s,r) \in \mathcal{E}_f^s, \\ 1 & \forall e = (r,w) \in \mathcal{E}_f^{rw}, \\ w.rc & \forall e = (w,k) \in \mathcal{E}_f^k, \end{cases} \tag{8}$$

where $w.rc$ represents the remaining capacity of candidate vehicle w for serving more orders.

- **Edge cost.** The cost $C(i,j)$ of each edge $(i,j) \in \mathcal{E}_f$ represents the cost incurred per unit flow passing through that edge. To redistribute vehicles more efficiently and provide high-quality service, we consider four factors in calculating edge costs, including the long-term mobility value of order r's travel direction, remaining capacity of candidate vehicle w, detour

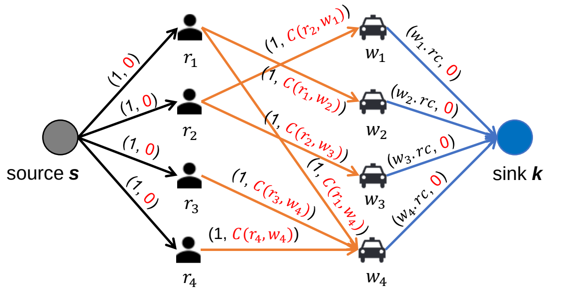


Fig. 4: Modeling order-vehicle matching as the MCMF problem, where (a, b) associated with each edge represents that the edge has capacity a and cost b, and $w.rc$ indicates the remaining capacity of vehicle w.

time and waiting time that are incurred when vehicle w is dispatched to serve order r. Therefore, the edge cost $C(i, j)$ is given as follows:

$$C(i, j) = -1 \times \mathtt{V}_\theta(\mathbf{s}_{cur}) \times (\Phi_c + \Psi_{wd}), \tag{9}$$

where $\mathtt{V}_\theta(\mathbf{s}_{cur})$ represents the long-term mobility value along order r's travel direction, and Φ_c is vehicle w's remaining capacity rate that is set as:

$$\Phi_c = \begin{cases} 0 & \text{if } 0 \leq \eta_w < \frac{1}{4}, \\ 0.8 & \text{if } \frac{1}{4} \leq \eta_w < \frac{1}{2}, \\ 0.9 & \text{if } \frac{1}{2} \leq \eta_w < \frac{3}{4}, \\ 1.0 & \text{if } \eta_w \geq \frac{3}{4}, \end{cases} \tag{10}$$

where $\eta_w = \frac{w.rc}{C_w}$ is a rate between remaining capacity and original capacity for vehicle w. The purpose of setting Φ_c in this way is to effectively redistribute vehicles that have more empty seats towards mobility directions with higher demand.

Meanwhile, we use Ψ_{wd} to measure vehicle w's service quality for order r. To make Ψ_{wd} fall within the range of $(0, 1)$ and accurately reflect the differences on service quality among various candidate vehicles, we propose a ranking-based method to calculate Ψ_{wd} for each edge $(r, w) \in \mathcal{E}_f^{rw}$. Note that, we employ the matching cost, *i.e.*, the sum of waiting time and detour time, as the measure for evaluating service quality. With the derived waiting time and detour time calculated in Section 4.2 for each matching (r, w), we compute the matching costs for all candidate vehicles in $\mathbb{T}_r$. Then, we sort candidate vehicles into a list according to their matching costs in an ascending order, and define a function $Rank(w)$ that can return vehicle w's rank in the sorted list. Finally, we compute Ψ_{wd} for each edge (r, w) as:

$$\Psi_{wd} = \frac{1}{Rank(w) \times \sum_{j=1}^{|\mathbb{T}_r|} \frac{1}{j}}. \tag{11}$$

Figure 4 shows a sample modeling of order-vehicle matching in ridesharing for four orders and four candidate vehicles within a sliding time window. Based on above definitions and modeling, we transform the order-vehicle matching into the MCMF problem with its objective as:

$$\min \sum_{(i,j) \in \mathcal{E}_f} C(i, j) \cdot flow(i, j), \tag{12}$$

where $flow(i, j)$ represents the flow passing on edge $(i, j) \in \mathcal{E}_f$. This objective is subject to the following constraints:

- *Non-negativity:* The flow on each edge (i, j) must be non-negative, *i.e.*,
$$flow(i, j) \geq 0, \quad \forall (i, j) \in \mathcal{E}_f. \tag{13}$$
- *Capacity constraint.* The flow on each edge $(i, j) \in \mathcal{E}_f$ must not exceed its edge capacity, *i.e.*,
$$flow(i, j) \leq U(i, j), \ \forall (i, j) \in \mathcal{E}_f, \tag{14}$$
where vertex i and j are neither source s nor sink k.
- *Flow conservation.* For each vertex i (except source s and sink k), its total inflow must equal total outflow, *i.e.*,
$$\sum_{(*,i) \in \mathcal{E}_f} flow(*, i) = \sum_{(i,*) \in \mathcal{E}_f} flow(i, *), \ \forall i \in \mathcal{V}_f \backslash \{s, k\}. \tag{15}$$
This constraint should also be applied to the source s and sink k, such that the flow into source s must be equal to the flow out of sink k.

To solve this MCMF problem, we make use of the network simplex algorithm [7], [8], which has been specifically designed to leverage the characteristics of flow network $\mathcal{G}_f$. By maintaining a spanning tree of edges, this algorithm iteratively adjusts the flow within the network, ensuring both feasibility and optimality at each step, to minimize the total cost [8].

4.4.2 Idle vehicle repositioning

In order to address the issue of vehicles remaining vacant for extended periods, ridesharing platforms need to proactively redistribute these idle vehicles to areas with anticipated high future demand. To achieve this, `JODR` explicitly redistributes vehicles that have been vacant for a time frame, *i.e.*, 5 minutes. To minimize repositioning costs, the target repositioning areas for a vehicle are limited to its current grid and neighboring grids.

To identify the travel directions with high future demand, `JODR` utilizes the mobility value function $\mathtt{V}_\theta(\cdot)$. For an idle vehicle w located in grid g_w, the possible target grids are denoted as set $\mathbb{G}_w$. For each grid $g_i \in \mathbb{G}_w$, a dummy order r_i is created, originating from grid g_w and terminating at grid g_i. Considering the dummy matching (r_i, w), we define the directional state $\mathbf{s}_i =< \vec{v}^g, tf, dw >$, where $\vec{v}^g =< g_w, g_i >$ is the grid-level mobility vector, tf and dw denote the current time frame index and day of the week. The function $\mathtt{V}_\theta(\mathbf{s}_i)$ is then employed to estimate the value of the dummy matching. Similarly, values are calculated for all dummy orders targeting different grids in set $\mathbb{G}_w$. The grid with the highest value may be chosen as the repositioning destination for idle vehicle w.

However, such an approach may blindly dispatch all idle vehicles within a grid to the same destination grid, which potentially disrupt the supply-demand equilibrium. To mitigate this issue, we propose a probabilistic method. Specifically, we utilize the *Softmax* function to transform the mobility values into a probability distribution, which is defined as:

$$p(g_i) = \frac{e^{\mathtt{V}_\theta(\mathbf{s}_i)}}{\sum_{g_j \in \mathbb{G}_w} e^{\mathtt{V}_\theta(\mathbf{s}_j)}}, \tag{16}$$

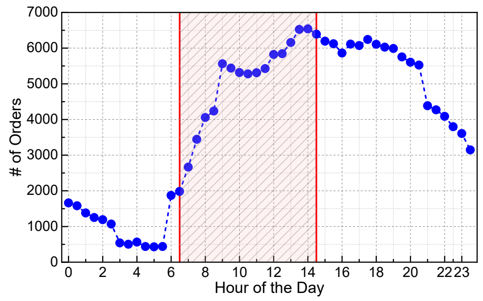


Fig. 5: Statistic on the number of orders across the day.

where $p(g_i)$ represents the repositioning probability to candidate grid g_i, $\mathbf{s}_i$ and $\mathbf{s}_j$ represent the states for repositioning to grid g_i and g_j, respectively. In addition, the denominator in Equation (16) is the sum of exponential mobility values for all grids in set $\mathbb{G}_w$. Based on this distribution, we sample destination grids for idle vehicle w. This probabilistic approach ensures that travel directions with higher mobility values have a higher likelihood of being selected, thereby aligning the repositioning strategies with underlying mobility dynamics.

5 Evaluation

In this section, we evaluate the performance of `JODR` using a large real-world trajectory dataset.

5.1 Experimental Setup

Dataset. We conduct data-driven experiments using a large-scale anonymized trajectory dataset publicly released by the Didi GAIA Initiative[1]. This dataset comprises a total of 7065907 ridesharing transactions collected in November 2016 within the downtown area of Chengdu city, China. Each transaction entry includes a transaction ID, vehicle ID, and ride order details. The ride order information consists of the release time, pick-up location, and drop-off location, represented by latitude and longitude coordinates. To provide insights into the data, we analyze the average hourly order volume, as depicted in Figure 5. The analysis reveals a gradual increase in the number of orders between 6:30AM and 2:30PM, highlighting the growing demand during this period. This finding emphasizes the importance of effective order dispatching and vehicle repositioning during these hours. Consequently, we focus our experiments on the data falling within this specific period.

Since the dataset only includes records of served ride orders, unserved orders are not captured. To address this limitation and enrich experimental data, we generate synthetic ride orders using the method outlined in previous work [28]. To simulate practical ride-hailing service transactions, this method is designed to learn the distribution of real-world orders on the road network over time of the day and the mobility patterns of ride-hailing vehicles. For more details on synthetic order generation, please refer to [28]. Furthermore, we keep the data of last week for testing, and the remaining data are used for model training.

We have downloaded the road information of Chengdu city from OpenStreetMap[2], and model the road network as a graph $\mathcal{G}_r(\mathcal{V}_r, \mathcal{E}_r)$, which consists of 21440 vertices and 466330 edges. Additionally, we divide the road network into 72 grids, with a uniform size about $1.16\,km \times 1.16\,km$.

1. https://outreach.didichuxing.com/research/opendata/.
2. http://www.openstreetmap.org/.

Compared methods. We compare `JODR` with the following state-of-the-art methods.

- *pGreedyDP* [44]. It utilizes grids to index both orders and vehicles, and proposes a greedy insertion strategy. Each order is sequentially assigned to the vehicle, whose new route has the minimum increased travel time.
- *mT-Share* [24]. It employs geographical information and travel directions to index orders and vehicles, and leverages these indexes to filter out irrelevant candidate vehicles. It also sequentially dispatches each order to the vehicle with the minimum increased cost. Additionally, it develops a probabilistic routing scheme to guide vehicles to meet predicted future orders along specific routes.
- *Prohpet* [43]. It also constructs the grid index for orders and vehicles, and proposes a new insertion operator that can handle both online orders and predicted future orders. The insertion based order assignments is solved through a dynamic programming algorithm.
- *PNAS* [4]. It periodically plans assignments for the batch of orders collected within a sliding time window, and dynamically generates optimal routes for available vehicles. It initiates with a greedy assignment and iteratively refines the assignment through constrained optimization.

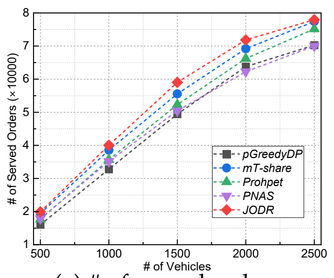
pGreedyDP, *mT-Share*, and *Prohpet* are real-time solutions, while *PNAS* is a batch-based solution. Since *Prohpet* has already considered the future orders and *PNAS* has its own idle vehicle repositioning mechanism, we thus enhance *pGreedyDP* and *mT-Share* by incorporating an additional idle vehicle repositioning component, which dispatches each idle vehicle to one random neighboring grid or remains this vehicle cruising within its current grid.

Evaluation metrics. We evaluate all the methods using the following performance metrics.

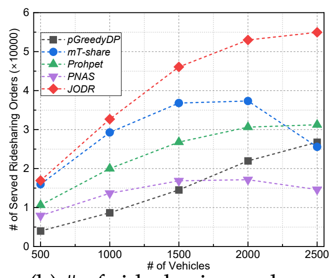
- *Number of served orders* represents the number of orders that have been successfully fulfilled.
- *Number of served ridesharing orders* captures the number of orders that share a vehicle with other orders, aiming to assess vehicle utilization in ridesharing.
- *Number of idle vehicle repositioning* quantifies the repositioning times vehicles have been redistributed while being idle within a time frame, *i.e.*, 5 minutes.
- *Average detour time* represents the mean additional travel time when compared to no ridesharing for all served orders.
- *Average waiting time* is calculated as the average time difference between the pick-up time of orders and their respective release time.
- *Average response time* is the average processing time for dispatching an order to the suitable vehicle.
- *Frequency of seat occupancy* $SO_i = K$ indicates that during the entire dispatching process of all vehicles, there were totally K instances where i passengers were grouped together in the same vehicle.

TABLE 2: The major parameter settings, where the default value of each parameter is marked in **bold**.

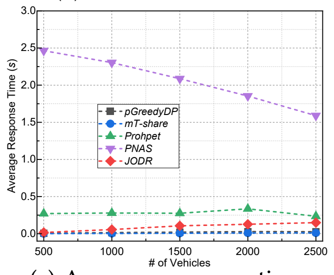
Parameter	Value
# of vehicles	500, 1000, 1500, **2000**, 2500
Vehicle capacity	3, **4**, 6, 8, 10
Flexible factor ρ	1.1, 1.2, **1.3**, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0
Threshold λ	0.867, 0.707, **0.500**, 0.259,
Sliding window size Δ_{sw}	2, **5**, 10, 20
Average riders per order c_r	**1.0**, 1.5, 2.0, 2.5, 3.0



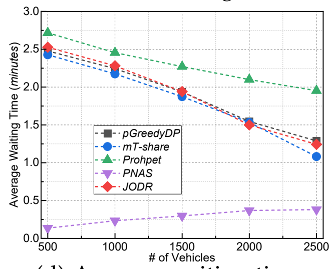
(a) # of served orders



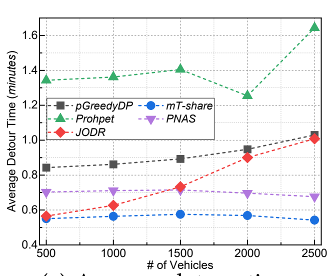
(b) # of ridesharing orders



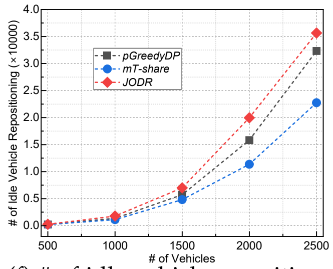
(c) Average response time



(d) Average waiting time



(e) Average detour time



(f) # of idle vehicle repositioning

Fig. 6: Performance comparison among different methods by varying the number of shared vehicles.

Implementation. We implement JODR and all the compared methods using Python. In line with prior research, each order r's origin and destination are pre-mapped to the nearest vertex in graph $\mathcal{G}_r$. The delivery deadline e_r for each order r is determined using a flexible factor ρ, which quantifies the additional travel cost riders are willing to tolerate compared to the shortest path. The shared vehicles are initialized at random vertices within graph $\mathcal{G}_r$, and when delivering riders, they are obligated to adhere strictly to the scheduled pick-up and drop-off sequence and the planned routes. Similar as previous works [24], [25], [31], [44], we assume a uniform driving speed of 30 km/h for all vehicles. Additionally, candidate vehicles of each order r are only searched within several grids, including the grid g_r where order r locates and g_r's neighboring grids. The major parameter settings are listed in Table 2.

All experiments are conducted on a server equipped with an Intel Core i9-12900K CPU@3.20GHz and 32GB of RAM. To accelerate route planning, we pre-compute the travel costs between any two vertices in graph $\mathcal{G}_r$ and cache

TABLE 3: Performance comparison on the frequency of seat occupancy by varying the number of shared vehicles. The best results for ridesharing cases are marked in **bold**.

# of vehicles	Method	Seat occupancy			
		SO_4	SO_3	SO_2	SO_1
500	*pGreedyDP*	4	86	1958	13989
	mT-Share	66	1088	9063	9228
	Prohpet	80	839	5634	11117
	PNAS	8	262	4392	13200
	JODR	**96**	**1394**	**9485**	8956
1000	*pGreedyDP*	5	180	4246	28371
	mT-Share	84	1721	16511	20251
	Prohpet	148	1471	10565	23499
	PNAS	11	387	7492	27261
	JODR	**167**	**2547**	**17985**	19340
1500	*pGreedyDP*	9	342	7102	42003
	mT-Share	81	1789	20504	33187
	Prohpet	167	1855	13966	36300
	PNAS	12	410	9114	40911
	JODR	**206**	**3515**	**24516**	30789
2000	*pGreedyDP*	13	549	10805	52664
	mT-Share	42	1304	20313	47538
	Prohpet	**169**	1947	15672	48425
	PNAS	4	331	9090	52831
	JODR	158	**3829**	**27060**	40785
2500	*pGreedyDP*	18	696	13041	56494
	mT-Share	11	525	13358	63516
	Prohpet	**165**	1858	15677	57449
	PNAS	6	224	7581	62180
	JODR	110	**3608**	**27592**	46639

the results in memory for quick retrieval across all methods. Each experimental configuration is executed 6 times, and only the average results are reported in this section.

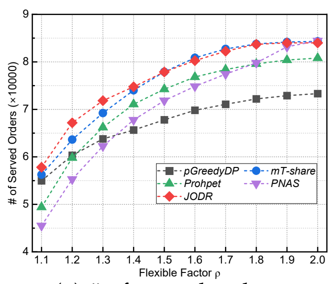
5.2 Performance Comparison

We compare the performance of all methods by varying the number of vehicles from 500 to 2500, with a step as 500.

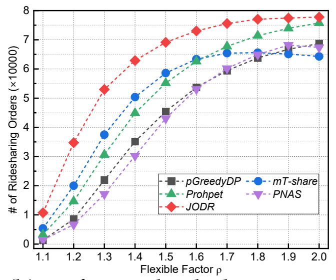
Figure 6(a) shows that all methods can serve more orders with an increase in the number of vehicles. Across different vehicle quantities, JODR consistently serves the most orders, attributing to the implicit vehicle repositioning during order dispatching. This is because JODR's implicit repositioning allows vehicles with sufficient capacity cater to orders associated with higher future demand in certain travel directions, thereby preemptively balancing the supply and demand across different travel directions. It is particularly crucial when the number of vehicles is limited. Taking the case of 1500 vehicles as an example, JODR can serve 12.89% and 17.01% more orders compared to *Prohpet* and *PNAS*, the two state-of-the-art works.

To validate whether our design facilitates more orders participating in the ridesharing service, we report the number of served ridesharing orders in Figure 6(b). It shows that JODR surpasses all the other methods across different settings. Compared to the four methods, JODR increases the number of served ridesharing orders by an average of 125.71%, 122.91%, 121.98%, 116.34% and 143.06%, respectively, under the five vehicle quantity settings. Remarkably, compared to *PNAS*, JODR serves 113.80% more ridesharing orders with only 500 vehicles, and can be up to 275.93% more with 2500 vehicles.

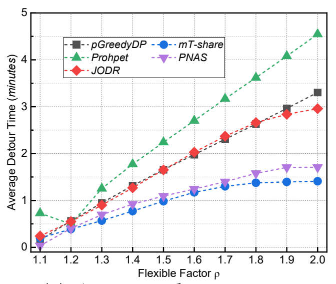
To better understand the ridesharing performances of different methods, we conduct an in-depth exploration on the vehicle schedules, and present detailed statistics on the frequency of seat occupancy in Table 3. Generally, two or



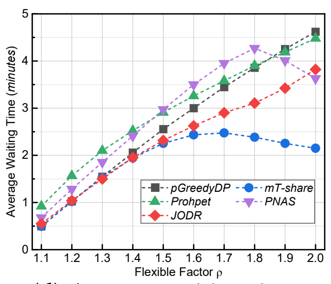
(a) # of served orders



(b) # of served ridesharing orders

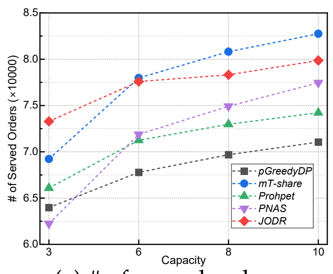


(c) Average detour time

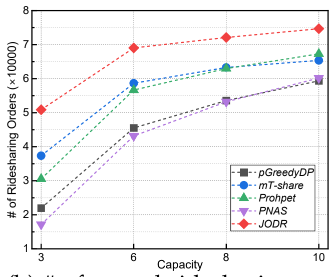


(d) Average waiting time

Fig. 7: Impact of flexible factor ρ on the performance.



(a) # of served orders



(b) # of served ridesharing orders

Fig. 8: Impact of vehicle capacity on the performance.

more orders sharing a vehicle can be viewed as a ridesharing case. Table 3 shows that the case where one vehicle serves one order is the most common scenario, while JODR demonstrates exceptional capability for matching multiple orders with one vehicle. Achieving full utilization of a vehicle's seats can be challenging. For instance, when four orders are accommodated in one single vehicle with a capacity of four, we observe that JODR consistently achieves the highest success rate in such scenarios. Among the 15 potential ridesharing cases, such as those with a seat occupancy of at least 2, JODR has emerged as the top performer, winning the first place in 13 instances.

Figure 6(c) reports the average response time of all methods. As a batch-based method, *PNAS* has the largest response time as it needs to iteratively optimize the arrangements for a batch of orders. While our JODR, as another batch-based solution, is on average 95.12% faster than *PNAS* on dispatching orders, which confirms the effectiveness of mobility clusters by grouping orders based on their travel directions. As *pGreedyDP* and *mT-Share* simply assign incoming orders to suitable vehicles, without considering future demand, they can respond orders rapidly. Another real-time method *Prohpet*, which considers the predicted future orders, takes a bit longer response time, even on average 66.82% slower than JODR.

We compare the waiting time of all methods in Figure 6(d). In general, more vehicles potentially allow each method to find a nearby vehicle to serve each order, and thus the waiting time can be reduced. *PNAS* shows the shortest waiting time, while *Prohpet* has the largest one. The other three methods, *i.e.*, *pGreedyDP*, *mT-Share*, and JODR, have close waiting time, with a difference < 0.2 minutes.

Figure 6(e) shows the performance comparison on average detour time. *mT-Share* matches vehicles with orders traveling on similar directions, and thus derives the fewest detour time. *PNAS* prefers to match orders with vehicles that have short travel distances and few ridesharing orders, thereby holding the second place. Furthermore, their detour time is less affected by the vehicle quantity. In contrary, *Prohpet* considers future orders in the route planning, resulting in the largest detour time. As a batch-based method, JODR has a moderate detour time among the five methods. Its detour time slightly increases with more vehicles, as it continuously redistributes vehicles during order dispatching. From Figure 6(e), we find that JODR reduces detour time by up to 57.92% compared to *Prohpet*, having the largest gap with 500 vehicles. Compared to *pGreedyDP*, JODR reduces detour time by 32.95%, while serving 323.76% more ridesharing orders with 500 vehicles.

Finally, Figure 6(f) presents the times of idle vehicle repositioning for *pGreedyDP*, *mT-Share*, and JODR. We see that JODR is more proactive on explicit repositioning than the other two methods, and thus has more idle vehicle repositioning actions, which are important for providing high-quality service.

5.3 Detailed Evaluation

Impact of flexible factor ρ. We perform experiments to investigate the impact of flexible factor ρ, while keeping other settings as the default values. As shown in Figure 7(a), as factor ρ increases, the number of served orders for each method improves accordingly. This is because all methods have more flexibility on order-vehicle matching when riders can tolerate more detour costs. We find that JODR outperforms the other methods in the settings where ρ is small, especially when ρ falls within $[1.1, 1.4]$. On average, JODR can offer an improvement of 10.83% than the four methods across this range. At the specific value $\rho = 1.1$, our JODR exhibits a remarkable performance increase of 27.05% when compared to *PNAS*, indicating that JODR still works well even riders have extremely rigorous requirements on the detour costs. Figure 7(b) shows that on the aspect of total served ridesharing orders, JODR significantly exceeds the quantities observed in all other compared methods across all values of ρ. In particular, JODR exhibits average improvement of 418.76%, 234.16%, 116.17%, and 63.04% than the four methods when setting ρ as 1.1, 1.2, 1.3, and 1.4, respectively. This excellent performance attributes to JODR's unique design that encourages vehicles to share seats among multiple orders.

Despite JODR's ability to accommodate a greater number of orders in ridesharing service, it does not result in a significant increase in detour and waiting time, as the evidences shown in Figure 7(c) and 7(d). When compared to the four methods, JODR exhibits a relatively gradual increase in detour time. In fact, we observe that JODR even outperforms *pGreedyDP* and *Prohpet*, with a reduction of 5.01% and 28.20% in detour time, respectively, when $\rho = 1.3$. Furthermore, Figure 7(d) shows that JODR generally outperforms the four compared methods in terms of waiting time when $\rho < 1.5$. This range is considered normal for real-world ridesharing systems.

Impact of vehicles capacity. In this experiment, we vary vehicle capacity as 3, 6, 8, and 10, and meanwhile appropriately adjust flexible factor ρ as 1.3, 1.5, 1.6, and 1.7, respectively. The reason for such an adjustment is because a larger vehicle capacity allows more orders share one vehicle, while requiring loose requirement on the detour constraint. When vehicle capacity is enlarged, the same number of vehicles has much more seat supply, enabling them to serve more orders. As shown in Figure 8(a), each method can serve more orders with a larger vehicle capacity. The observation also holds for the number of served ridesharing orders, as shown in Figure 8(b). We find that JODR is still more efficient in serving much more orders than other methods, especially when the total seat supply is limited. Taking vehicle capacity of 3 as an example, JODR can serve an average of 12.26% more orders and 107.70% more ridesharing orders than the other four methods.

Impact of average riders per order. We study the impact of parameter c_r, average riders per order, on the performance of all methods. Similar as the experimental setting in [47], we set the value of c_r as 1.0, 1.5, 2.0, 2.5, and 3.0. Note that, an increase in c_r will result in that fewer orders can be shared by one vehicle due to the limit of vehicle capacity. As shown in Figure 9, all methods can serve more riders as c_r increases. Specifically, JODR can serve the most riders, while *PNAS* has the smallest number of served riders. The other three methods have similar performance in this experiment.

Impact of sliding time window size Δ_{sw}. As the batch-based methods, *PNAS* and JODR process a batch of orders collected within a sliding time window of size Δ_{sw}, which will affect the ridesharing performance. We study its impacts by varying size Δ_{sw} as $3s$, $6s$, $8s$, and $10s$. Figure 10 shows that as Δ_{sw} increases, both the number of served orders and ridesharing orders for *PNAS* slightly increase. While JODR consistently outperforms *PNAS*, *e.g.*, averagely serving 12.28% more orders and 189.32% more ridesharing orders. In terms of response time, Figure 10 shows that the computation time of *PNAS* does increase with Δ_{sw} and fluctuates significantly due to more orders being jointly processed. In contrary, JODR exhibits a more stable response time, almost unchanged, indicating that our mobility cluster design effectively addresses the high computational cost of batch-based methods and is robust with respect to the time window size.

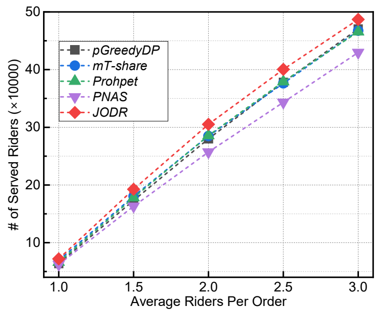


Fig. 9: Impact of the average riders per order.

5.4 Ablation Experiments

We have several design elements to improve the efficiency of JODR. Specifically, we propose mobility clustering to divide a batch of orders into fine-grained groups based on their travel directions. Furthermore, we consider the vehicle's remaining capacity, the service quality measured by detour and waiting time, and the mobility value of an order's travel direction into Equation (9) for computing edge costs in the flow network $\mathcal{G}_f$. These factors together determine order dispatching results. We thus conduct a series of ablation experiments by individually removing each design component to evaluate its effectiveness.

Effectiveness of mobility clustering. As shown in Figure 11(a), JODR without mobility clustering has an 8.39% decrease in the number of served orders but a 7.12% increase in the number of ridesharing orders. This is because, without the constraint on travel directions, JODR tends to serve shorter-distance orders regardless of their travel direction consistency. In fact, we find that the majority of these served ridesharing orders belong to the cases where two orders share a vehicle. However, Figure 11(a) shows that without fine-grained order divisions the average response time accordingly increases by 4.90%. Furthermore, Figure 11(b) reports a 17.58% increase in average waiting time, and a 17.81% increase in the number of idle vehicle dispatches. These results demonstrate that it is necessary to consider the travel directions of both orders and vehicles during order assignments for better service quality.

Effectiveness of vehicle's remaining capacity. During order-vehicle matching, JODR tends to dispatch vehicles with more remaining capacity to serve orders with higher mobility values. The results in Figure 11(a) reveal that ignoring this factor may lead to a 2.72% decrease in the number of served orders, a 4.41% decrease in the number of served ridesharing orders, and a slight increase in the average response time. In Figure 11(b), despite a slight decrease in the average waiting time, we see about 4.05% increase in average detour time and 10.77% increase in the number of idle vehicle dispatches. These results suggest that vehicle's remaining capacity indeed affects platform revenue and the service quality.

Effectiveness of detour and waiting time. JODR treats both detour and waiting time as important factors on measuring the service quality of order dispatching. As shown in Figure 11(a), not accounting for them results in a great decrease in the number of served orders, *e.g.*, 37.44% decrease in served ridesharing orders. Figure 11(b) shows a 53.47% decrease in the idle vehicle dispatches, yet a significant increase in average waiting time by 62.97%. Without considering these factors, the system may assign orders with vehicles far away, and thus impairs the service quality.

Effectiveness of mobility value. JODR heavily relies on the mobility values for effective order dispatching and vehicle repositioning. By removing mobility value from Equation (9), we observe severe performance degradation across most of the metrics. For example, Figure 11(a) shows that excluding this factor results in 7.21% and 32.88% decrease in the number of served orders and ridesharing orders, respectively, and slight delay in response time. Despite the decrease in detour time and idle vehicle repositioning, as

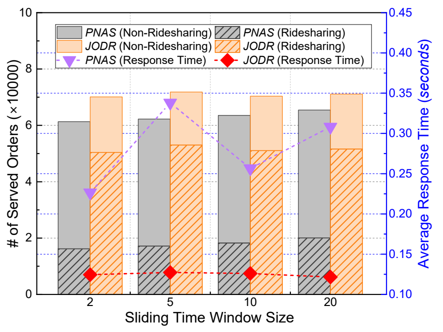


Fig. 10: Impact of sliding time window size Δ_{sw}.

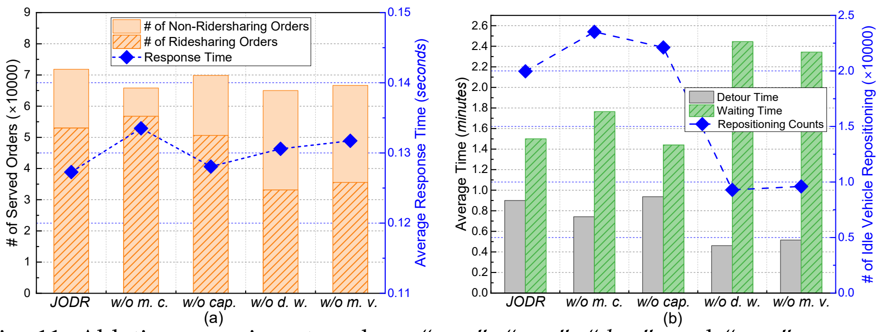


Fig. 11: Ablation experiments, where *"m.c."*, *"cap."*, *"d.w."*, and *"m.v."* represent *mobility clustering*, *capacity*, *detour and waiting time*, and *mobility value*, respectively.

shown in Figure 11(b), we see a clear increase in waiting time by 56.16%. The results suggest that mobility value function can effectively capture the long-term mobility values, and efficiently guide order assignments.

6 Conclusion

This paper presents JODR, a comprehensive framework that jointly optimizes order dispatching and vehicle repositioning for dynamic ridesharing. The key idea of our framework lies in a novel mobility value function that effectively integrates these two essential tasks. By leveraging this value function, JODR efficiently determines the optimal order-vehicle assignments by formulating the order dispatching as a minimum-cost maximum-flow problem, while also supporting intelligent repositioning of idle vehicles. Extensive experiments on a large real-world dataset demonstrate the superior performance of our JODR compared to the state-of-the-art methods for dynamic ridesharing across a wide range of performance metrics.

Acknowledgment

This work was supported in part by National Natural Science Foundations of China under Grants 62172284 and U2001207, the grant of Guangdong Basic and Applied Basic Research Foundation (No.2022A1515010155), Guangdong Provincial Key Lab of Integrated Communication, Sensing and Computation for Ubiquitous Internet of Things (No.2023B1212010007), Guangzhou Municipal Science and Technology Project (No.2023A03J0011), and the Project of DEGP (No.2023KCXTD042 and No.2021ZDZX1068).

References

[1] Didi. https://www.didiglobal.com/, Accessed in October 2024.
[2] Uber. https://www.uber.com/, Accessed in October 2024.
[3] A. O. Al-Abbasi, A. Ghosh, and V. Aggarwal. DeepPool: distributed model-free algorithm for ride-sharing using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 20(12):4714–4727, 2019.
[4] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017.
[5] X. Azagirre, A. Balwally, G. Candeli, N. Chamandy, B. Han, A. King, H. Lee, M. Loncaric, S. Martin, V. Narasiman, et al. A better match for drivers and riders: reinforcement learning at Lyft. *INFORMS Journal on Applied Analytics*, 54(1):71–83, 2024.
[6] X. Bei and S. Zhang. Algorithms for trip-vehicle assignment in ride-sharing. In *AAAI*, 2018.
[7] W. H. Cunningham. A network simplex method. *Mathematical Programming*, 11:105–116, 1976.
[8] G. B. Dantzig. Application of the simplex method to a transportation problem. *Activity Analysis and Production and Allocation*, 1951.
[9] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.
[10] S. Ge, X. Zhou, T. Qiu, G. Wu, and X. Wang. Towards supply-demand equilibrium with ridesharing: an elastic order dispatching algorithm in MoD system. *IEEE Transactions on Mobile Computing*, 23(5):5229–5244, 2024.
[11] G. Guo and Y. Xu. A deep reinforcement learning approach to ride-sharing vehicle dispatching in autonomous mobility-on-demand systems. *IEEE Intelligent Transportation Systems Magazine*, 14(1):128–140, 2020.
[12] X. Guo, N. S. Caros, and J. Zhao. Robust matching-integrated vehicle rebalancing in ride-hailing system with uncertain demand. *Transportation Research Part B: Methodological*, 150:161–189, 2021.
[13] M. Haliem, G. Mani, V. Aggarwal, and B. Bhargava. A distributed model-free ride-sharing approach for joint matching, pricing, and dispatching using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 22(12):7931–7942, 2021.
[14] S. He and K. G. Shin. Spatio-temporal capsule-based reinforcement learning for mobility-on-demand coordination. *IEEE Transactions on Knowledge and Data Engineering*, 34(3):1446–1461, 2020.
[15] J. Holler, R. Vuorio, Z. Qin, X. Tang, Y. Jiao, T. Jin, S. Singh, C. Wang, and J. Ye. Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem. In *IEEE ICDM*, pages 1090–1095, 2019.
[16] X. Huang, J. Ling, X. Yang, X. Zhang, and K. Yang. Multi-agent mix hierarchical deep reinforcement learning for large-scale fleet management. *IEEE Transactions on Intelligent Transportation Systems*, 24(12):14294–14305, 2023.
[17] Y. Huang, F. Bastani, R. Jin, and X. S. Wang. Large scale real-time ridesharing with service guarantee on road networks. *Proceedings of the VLDB Endowment*, 7(14):2017–2028, 2014.
[18] A. Kumar, A. Gupta, M. Parida, and V. Chauhan. Service quality assessment of ride-sourcing services: a distinction between ride-hailing and ride-sharing services. *Transport Policy*, 127:61–79, 2022.
[19] K. Lin, R. Zhao, Z. Xu, and J. Zhou. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *ACM SIGKDD*, pages 1774–1783, 2018.
[20] Q. Lin, L. Deng, J. Sun, and M. Chen. Optimal demand-aware ride-sharing routing. In *IEEE INFOCOM*, pages 2699–2707, 2018.
[21] Q. Lin, W. Xu, M. Chen, and X. Lin. A probabilistic approach for demand-aware ride-sharing optimization. In *ACM MobiHoc*, pages 141–150, 2019.
[22] C. Liu, C.-X. Chen, and C. Chen. Meta: a city-wide taxi repositioning framework based on multi-agent reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 23(8):13890–13895, 2021.
[23] L. Liu, Y. Zhou, and J. Xu. A cloud-edge-end collaboration framework for cruising route recommendation of vacant taxis. *IEEE Transactions on Mobile Computing*, 23(5):4678–4693, 2024.
[24] Z. Liu, Z. Gong, J. Li, and K. Wu. Mobility-aware dynamic taxi ridesharing. In *IEEE ICDE*, pages 961–972, 2020.
[25] Z. Liu, Z. Gong, J. Li, and K. Wu. mT-Share: a mobility-aware dynamic taxi ridesharing system. *IEEE Internet of Things Journal*, 9(1):182–198, 2021.

[26] Z. Liu, J. Li, and K. Wu. Context-aware taxi dispatching at city-scale using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 23(3):1996–2009, 2022.

[27] Z. Liu, J. L. Lin, Z. Xia, C. Chen, and K. Wu. Towards efficient ridesharing via order-vehicle pre-matching using attention mechanism. In *IEEE ICDM*, pages 1–10, 2024.

[28] Z. Liu, H. Zhang, G. Ouyang, J. Chen, and K. Wu. Data-driven pick-up location recommendation for ride-hailing services. *IEEE Transactions on Mobile Computing*, 23(2):1001–1015, 2024.

[29] H. Luo, Z. Bao, F. M. Choudhury, and J. S. Culpepper. Dynamic ridesharing in peak travel periods. *IEEE Transactions on Knowledge and Data Engineering*, 33(7):2888–2902, 2019.

[30] Q. Ma, Z. Cao, K. Liu, and X. Miao. QA-Share: toward an efficient QoS-aware dispatching approach for urban taxi-sharing. *ACM Transactions on Sensor Networks*, 16(2):1–21, 2020.

[31] S. Ma, Y. Zheng, and O. Wolfson. T-share: a large-scale dynamic taxi ridesharing service. In *IEEE ICDE*, pages 410–421, 2013.

[32] S. Ma, Y. Zheng, and O. Wolfson. Real-time city-scale taxi ridesharing. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1782–1795, 2014.

[33] F. Miao, S. Han, A. M. Hendawi, M. E. Khalefa, J. A. Stankovic, and G. J. Pappas. Data-driven distributionally robust vehicle balancing using dynamic region partitions. In *ACM/IEEE ICCPS*, pages 261–271, 2017.

[34] F. Miao, S. Lin, S. Munir, J. A. Stankovic, H. Huang, D. Zhang, T. He, and G. J. Pappas. Taxi dispatch with real-time sensing data in metropolitan areas: a receding horizon control approach. In *ACM/IEEE ICCPS*, pages 100–109, 2015.

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[36] F. Murtagh and P. Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[37] T. Oda and C. Joe-Wong. MOVI: a model-free approach to dynamic fleet management. In *IEEE INFOCOM*, pages 2708–2716, 2018.

[38] Z. T. Qin, H. Zhu, and J. Ye. Reinforcement learning for ridesharing: an extended survey. *Transportation Research Part C: Emerging Technologies*, 144:103852, 2022.

[39] D. Shi, Y. Tong, Z. Zhou, B. Song, W. Lv, and Q. Yang. Learning to assign: towards fair task assignment in large-scale ride hailing. In *ACM SIGKDD*, pages 3549–3557, 2021.

[40] J. Sun, H. Jin, Z. Yang, L. Su, and X. Wang. Optimizing long-term efficiency and fairness in ride-hailing via joint order dispatching and driver repositioning. In *ACM SIGKDD*, pages 3950–3960, 2022.

[41] J. Tang, F. Liu, Y. Wang, and H. Wang. Uncovering urban human mobility from large scale taxi GPS data. *Physica A: Statistical Mechanics and its Applications*, 438:140–153, 2015.

[42] X. Tang, F. Zhang, Z. Qin, Y. Wang, D. Shi, B. Song, Y. Tong, H. Zhu, and J. Ye. Value function is all you need: a unified learning framework for ride hailing platforms. In *ACM SIGKDD*, pages 3605–3615, 2021.

[43] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, and K. Xu. Unified route planning for shared mobility: an insertion-based framework. *ACM Transactions on Database Systems*, 47(1):1–48, 2022.

[44] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu. A unified approach to route planning for shared mobility. *Proceedings of the VLDB Endowment*, 11(11):1633, 2018.

[45] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double Q-learning. In *AAAI*, 2016.

[46] J. Wang, P. Cheng, L. Zheng, L. Chen, and W. Zhang. Online ridesharing with meeting points. *Proceedings of the VLDB Endowment*, 15(13):3963–3975, 2022.

[47] T. Wang, H. Luo, Z. Bao, and L. Duan. Dynamic ridesharing with minimal regret: towards an enhanced engagement among three stakeholders. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):3712–3726, 2023.

[48] H. Wei, Z. Yang, X. Liu, Z. Qin, X. Tang, and L. Ying. A reinforcement learning and prediction-based lookahead policy for vehicle repositioning in online ride-hailing systems. *IEEE Transactions on Intelligent Transportation Systems*, 1(1):1–11, 2023.

[49] J. Xi, F. Zhu, Y. Chen, Y. Lv, C. Tan, and F. Wang. DDRL: a decentralized deep reinforcement learning method for vehicle repositioning. In *IEEE ITSC*, pages 3984–3989, 2021.

[50] J. Xi, F. Zhu, P. Ye, Y. Lv, H. Tang, and F.-Y. Wang. HMDRL: hierarchical mixed deep reinforcement learning to balance vehicle supply and demand. *IEEE Transactions on Intelligent Transportation Systems*, 23(11):21861–21872, 2022.

[51] X. Xie, F. Zhang, and D. Zhang. PrivateHunt: multi-source data-driven dispatching in for-hire vehicle systems. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(1):1–26, 2018.

[52] B. Yu, Y. Ma, M. Xue, B. Tang, B. Wang, J. Yan, and Y.-M. Wei. Environmental benefits from ridesharing: a case of Beijing. *Applied Energy*, 191:141–152, 2017.

[53] Z. Yu and M. Hu. Deep reinforcement learning with graph representation for vehicle repositioning. *IEEE Transactions on Intelligent Transportation Systems*, 23(8):13094–13107, 2021.

[54] N. J. Yuan, Y. Zheng, L. Zhang, and X. Xie. T-finder: a recommender system for finding passengers and vacant taxis. *IEEE Transactions on Knowledge and Data Engineering*, 25(10):2390–2403, 2012.

[55] D. Zhang, T. He, F. Zhang, M. Lu, Y. Liu, H. Lee, and S. H. Son. Carpooling service for large-scale taxicab networks. *ACM Transactions on Sensor Networks*, 12(3):1–35, 2016.

[56] Y. Zhao, G. Fan, H. Jin, W. Ma, B. He, and X. Wang. Joint order dispatch and repositioning for urban vehicle sharing systems via robust optimization. In *IEEE ICDCS*, pages 663–673, 2021.

[57] B. Zheng, L. Ming, Q. Hu, Z. Lü, G. Liu, and X. Zhou. Supply-demand-aware deep reinforcement learning for dynamic fleet management. *ACM Transactions on Intelligent Systems and Technology*, 13(3):1–19, 2022.

[58] B. Zheng, L. Ming, Q. Hu, Z. Lü, G. Liu, and X. Zhou. Supply-demand-aware deep reinforcement learning for dynamic fleet management. *ACM Transactions on Intelligent Systems and Technology*, 13(3):1–19, 2022.

[59] L. Zheng, L. Chen, and J. Ye. Order dispatch in price-aware ridesharing. *Proceedings of the VLDB Endowment*, 11(8):853–865, 2018.

[60] L. Zheng, P. Cheng, and L. Chen. Auction-based order dispatch and pricing in ridesharing. In *IEEE ICDE*, pages 1034–1045, 2019.

Zhidan Liu received the Ph.D. degree in computer science and technology from Zhejiang University, Hangzhou, China, in 2014. After that, he worked as a Research Fellow in Nanyang Technological University, Singapore, and a faculty member with College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. He is currently an Assistant Professor at Intelligent Transportation Thrust, System Hub, The Hong Kong University of Science and Technology (Guangzhou). His research interests include Artificial Internet of Things, mobile computing, urban computing, and big data analytic. He is a senior member of CCF, a member of IEEE and ACM.

Guofeng Ouyang received the B.S. degree from Zhaoqing University, Zhaoqing, China, in 2021, and the master degree from Shenzhen University, Shenzhen, China, in 2024, under the supervision of Dr. Zhidan Liu. His research interests are in the areas of ridesharing and trajectory data analysis.

Bolin Zhang received the B.S. degree in Software Engineering from Shenzhen University, Shenzhen, China, in 2023. He is currently a second-year master student with College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China, under the supervision of Dr. Zhidan Liu. His research interests are in the areas of trajectory data analysis and urban computing.

Bo Du received the Ph.D. degree in Transportation Engineering from Nanyang Technological University in Singapore. He is currently a Senior Lecturer at Griffith Business School in Australia with research interests spanning transportation and logistics system modeling and optimization, optimal planning and operation of zero-emission and emerging mobility, data analytics and data-driven modeling. He is an Associate Editor of *IEEE Transactions on Intelligent Transportation Systems*.

Chao Chen is a Full Professor at College of Computer Science, Chongqing University, Chongqing, China. He obtained his Ph.D. degree from Sorbonne University and Institut Mines-Télécom/Télécom SudParis, France in 2014. He received the B.Sc. and M.Sc. degrees in control science and control engineering from Northwestern Polytechnical University, Xi'an, China, in 2007 and 2010, respectively. Dr. Chen got published over 100 papers including 40 ACM/IEEE Transactions. His work on taxi trajectory data mining was featured by IEEE SPECTRUM in 2011, 2016, and 2020, respectively. He was also the recipient of the Best Paper Runner-Up Award at MobiQuitous 2011. His research interests include pervasive computing, mobile computing, urban logistics, data mining from large-scale GPS trajectory data, and big data analytics for smart cities. He is a senior member of IEEE and CCF.

Kaishun Wu received his Ph.D. degree in Computer Science and Engineering at The Hong Kong University of Science and Technology (HKUST). Before joining HKUST(GZ) as a Full Professor at DSA Thrust and IoT Thrust in 2022, he was a distinguished Professor and Director of Guangdong Provincial Wireless Big Data and Future Network Engineering Center at Shenzhen University. Prof. Wu is an active researcher with more than 200 papers published on major international academic journals and conferences, as well as more than 100 invention patents, including 12 from the USA. He received the 2012 Hong Kong Young Scientist Award, the 2014 Hong Kong ICT Awards: Best Innovation, and 2014 IEEE ComSoc Asia-Pacific Outstanding Young Researcher Award. He is a Fellow of IEEE, IET, and AAIA.