Adaptive Dynamic Programming for Multi-Driver Order Dispatching at Large-Scale

Kai Jiang, Yue Cao, *Senior Member, IEEE*, Huan Zhou, *Member, IEEE*, Jie Wu, *Fellow*, *IEEE*, Zhao Zhang, *Member, IEEE*, and Zhi Liu, *Senior Member, IEEE*

***Abstract*—Order dispatching, which involves assigning orders to demand-matched vehicles, is an underlying issue for ride-sharing services. Previous works on order dispatching are often quasi-static and myopic[1], thus performing unsatisfactorily in the ride-sharing setting. To address these challenges, recent studies attempt to augment large-scale decision optimization from a data-driven perspective. Among them, Adaptive Dynamic Programming (ADP) has exhibited its particular potential for sequential decision-making with a long-term objective under uncertainty. In this paper, we investigate order dispatching with consideration of vehicle repositioning by exploiting ADP. We first formulate the optimization problem as a Markov Decision Process (MDP), where the dispatching decision is determined by a series of agents (the decision-making entity) under the time sequence model. Then, based on the generated available trips by a graph theory-based method, an ADP-based Multi-driver Order Dispatching method (AMOD) is proposed. In particular, AMOD reconstructs the Bellman update process around the post-decision states to avoid approximating the embedded expectations explicitly. As for non-linear function approximation, it converts the value function into a linear combination by a quadratic decomposition, and estimates the decomposed value function with neural network-based parameter approximation. In addition, vehicle repositioning is performed along with each batch dispatching to balance ride supply across geographic dimensions. Extensive simulations are conducted based on real-world data. Especially, AMOD can achieve 34.6% improvement at maximum and 15.9% on average compared with other baselines, when the capacity constraint is 10.**

***Index Terms*—Order Dispatching; Adaptive Dynamic Programming; Graph Theory; Markov Decision Process.**

I. Introduction

With the popularity of shared mobility in traffic resource reconfiguration, on-demand ride-sharing services prevail as an essential pillar of the modern transportation system [1]. The advanced user-centric ride-sharing platforms, exemplified by Uber, Lyft, DiDi Chuxing, *etc.*, have substantially revolutionized the transportation landscape. By one estimate, the global market value of the ride-sharing industry gets $135 billion in 2020, which is expected to reach $218 billion by 2025 [2].

On-demand ride-sharing services can coordinate between finite supply (vehicle) and asymmetric demand (order). This efficient and sustainable traffic mode facilitates a common vehicle to synchronously serve multiple passengers with similar and time-diverse itineraries [3]. It alleviated traffic congestion, emission, and energy consumption via elevating idle seat utilization, thereby offering enormous potential to improve transportation efficiency and living conditions [1].

As the ride-sharing service evolves imperative to promote, numerous issues remain yet to be resolved. Thereinto, order dispatching is a critical issue in alleviating the once-prominent gap between supply and demand. It requires instantaneous decision-making for extensive vehicles over an uncertain future demand. Meanwhile, each vehicle also has to decide whether to actively pick up new orders based on its vehicle status. Underlying these is a sequential decision-making problem, which corresponds to assigning a combination of order requests to a demand-matched vehicle (empty or partially filled) under delay and capacity constraints [4]. Considering the long-term (*e.g.*, several hours or a day) influences, the objective is to perform order dispatching by satisfying the current demand and optimizing the anticipated utility over time. Optimizing over time means that we need to consider the impact of decisions now on an uncertain future where the model is aware of the current travel patterns. This relies on situation awareness, real-time regulation, as well as adaptive tailoring of supply and demand [5], [6]. Therefore, a flexible and provident order dispatching method is urgently required.

Indeed, in a highly stochastic and non-stationary ride-sharing setting, the orders service record, actual routes, vehicle status (current position, passenger count, capacity constraint) should be updated continuously based on historical information. Previous works [7]–[9] using greedy or model-based heuristics often rely on regular distribution hypotheses and pre-specified rules. Nevertheless, the dynamics of supply and demand are uncertain in spatiotemporal space, which poses challenges to predict and model. Furthermore, these methods are quasi-static and myopic[1] for potential future demands.

This work was supported in part by the National Key Research and Development Program of China under Grant No. 2022YFE0139300, the Ministry of Education- Industry University Research Innovation Program under Grant No. 2021LDA07005, the National Natural Science Foundation of China (NSFC) under Grant No. 62172255, and the Outstanding Youth Program of Hubei Natural Science Foundation under Grant No. 2022CFA080. *(Corresponding author: Yue Cao.)*

K. Jiang and Y. Cao are with the School of Cyber Science and Engineering, Wuhan University, Wuhan 430000, China. (e-mail: kai.jiang, yue.cao@whu.edu.cn).

H. Zhou is with the School of Computer Science, Northwestern Polytechnical University, Xi'an 710129, China. (e-mail: zhouhuan117@gmail.com).

J. Wu is with the Department of Computer and Information Sciences, Temple University, Philadelphia PA19122, USA. (e-mail: jiewu@temple.edu).

Z. Zhang is with the School of Transportation Science and Engineering, Beihang University, Beijing 100191, China. (e-mail: zhaozhang@buaa.edu.cn).

Z. Liu is with the Department of Computer and Network Engineering, University of Electro-Communications, Tokyo, Japan. (e-mail: liu@ieee.org).

[1]Limited adaptabilities to dynamics and only optimizing for immediate returns without foresighted considerations.

They learn only from the current one-step return and neglect the long-term influences, *i.e.*, a prior dispatching decision could impact the supply distribution in subsequent time steps. Thus, they consequently perform unsatisfactorily in practice.

All these challenges have promoted technical advances in data analysis and computing power. Recently, these advances have provided opportunities to augment large-scale decision optimization from a data-driven perspective. Based on this, Adaptive Dynamic Programming (ADP) has exhibited its particular potential for the method design of order dispatching, and attracted widespread concern. Specifically, ADP is the interdiscipline of artificial intelligence and control domain, especially deemed for control and sequential decision-making with a long-term objective under uncertainty [10], [11]. In the ADP setting, order dispatching is performed through a function approximation structure. This structure estimates the anticipated value of an operational decision with spatiotemporal dependency.

In this paper, we investigate order dispatching with consideration of vehicle repositioning by exploiting ADP. Firstly, we formulate the optimization problem as a Markov Decision Process (MDP), where the dispatching decision is determined by a series of agents under the time sequence model. Secondly, to handle the order combinatorial complexity, we generate available trips for each vehicle through a graph theory-based method. Thirdly, based on demand and supply patterns from historical information, an **A**DP-based **M**ulti-driver **O**rder **D**ispatching method **(AMOD)** is proposed. Fourthly, we perform vehicle repositioning for non-occupied vehicles to balance ride supply across geographic dimensions. Along with each batch, all vehicles are repositioned toward the areas of extractive order requests by performing a linear program. Finally, we conduct simulations based on real-world data from New York City taxi daily operations.

The main contributions are summarized as follows:

1) We study the large-scale order dispatching problem from a data-driven perspective. Compared to previous works (greedy-based and model-based heuristic methods), our proposed AMOD makes dispatching decisions with adaptive and foresighted considerations. It captures hidden dynamics of the environment through real-time iteration, and estimates the anticipated future value rather than a one-step return.
2) Generally, the ability to place expectations over future knowledge in most dynamic programming works is taken for granted. However, this is not legitimate enough, and the expectation is rigid to compute. Therefore, we reconstruct the Bellman update process around the post-decision states to avoid approximating the embedded expectations explicitly. Besides, as for non-linear function approximation, we convert the value function into a linear combination by a quadratic decomposition, and estimate the decomposed value function with neural network-based parameter approximation.
3) Except for order dispatching, we also perform vehicle repositioning to balance ride supply across geographic dimensions. Unlike most previous works (repositioning decisions are dependent only on unassigned order requests), all vehicles are repositioned toward the areas of extractive order requests by performing a linear program. This ensures that sufficient vehicles are involved in the repositioning process along with each batch dispatching.

The remainder of this paper is organized as follows. Section II reviews the related work. Section III defines the system model formally. Section IV formulates the problem to maximize the order request response rate of long term in the system. Section V generates available trips for each vehicle and proposes AMOD for order dispatching. Furthermore, simulation results are analyzed in Section VI. Conclusion is summarized in Section VII.

II. Related Work

Order dispatching is a typical sequential decision-making problem in on-demand ride-sharing, where its generalized forms have been investigated substantially. Here, we categorize existing solutions of order dispatching into three threads: greedy-based methods [7], [8], [12]–[15], model-based heuristic methods [16]–[21], and ADP-based methods [3], [11], [23]–[26], [28], [29].

A. Greedy-based methods for order dispatching

In the infancy of ride-sharing, greedy-based methods are widely adopted for order dispatching. They dispatch the vehicle based on the nearest neighbour query or first-come-first-serve principles. For instance, Tong *et al.* [7] explored the performance of the greedy-based method in practice, and justified that this straightforward rule can also deliver good results. Cheng *et al.* [12] developed the utility-aware order dispatching problem on road networks, and operated the dispatching decision in a greedy manner to maximize the overall utility. Similarly, to maximize the revenue of the ride-sharing platform without compromising service quality, a distributed auction-based framework was proposed in [13], which dispatched the order requests to vehicles with the highest bids. Furthermore, Zhang *et al.* [14] investigated the theoretical queueing model for autonomous mobility-on-demand systems, and optimized the objective function upon the current arranged request. Lowalekar *et al.* [8] focused on assigning vehicles to the generated zone paths, where each zone path represents a combination of order requests. Duan *et al.* [15] proposed a greedy strategy based on iterative matching and merging, intending to reduce the number of vehicles required for order requests.

Although these greedy-based methods are simple to implement, they are myopic for potential future requests. They tend to prioritize immediate return rather than global supply utilization. As the mismatch between demand and supply in spatiotemporal distribution, these methods will lead to local sub-optimal solutions when considering the long-term run.

B. Model-based heuristic methods for order dispatching

With the availability of big data, another thread tries to design model-based heuristic methods from a data-driven perspective. Bei *et al.* [16] presented the ride-sharing assignment problem as a combinatorial optimization problem, and

addressed it from an algorithmic resource allocation perspective. To *et al.* [17] established the maximum task assignment problem in spatial crowdsourcing, and proposed alternative solutions by exploiting the spatial properties of the problem space. Furthermore, Greenwood *et al.* [18] devoted themselves to uncovering the societal benefits of order dispatching in ride-sharing services. Zhang *et al.* [19] demonstrated that considering spatial optimality alone can achieve a preferable success rate of global order matches, although they just focused on the current one-step return. Chen *et al.* [20] investigated the order dispatching in the circumstance of package delivery. They applied a two-phase solution, including order prediction and delivery route planning. Ma *et al.* [21] proposed a mobile-cloud-based real-time taxi-sharing system, which optimizes the delivery capability and total travel distance of vehicles.

However, these model-based heuristic methods are often quasi-static and rely on regular distribution hypotheses. They oversimplify demand-supply dynamics and fail to consider all the intricacies involved in order dispatching. Thus, when scaling to large-scale scenarios, this class of methods cannot provide the theoretically claimed performance.

TABLE I
NOTATIONS AND SYMBOLS

Notation	Explanation
$\mathcal{I}$	the set of street intersections in $\mathcal{G}$
$\mathcal{A}$	the adjacency of street intersections in $\mathcal{G}$
$\mathcal{O}_t$	the set of order requests in time slot t
a_j^t, b_j^t	the origin and destination of order request $o_{j,t}$
$\mathcal{N}_t$	the set of available vehicles in time slot t
p_i^t, c_i^t	the position and capacity constraint of vehicle $v_{i,t}$
ξ_i^t	the list of locations that vehicle $v_{i,t}$ will traverse to capture subsequent orders
Φ_p, Φ_d	the maximum pick-up delay and maximum detour delay
P_t	a potential trip in time slot t
$\mathcal{Q}_{i,t}$	the set of available trips for vehicle $v_{i,t}$
$q_{i,t}$	an available trip for vehicle $v_{i,t}$ in time slot t
$x_{i,q}^t$	the decision of vehicle $v_{i,t}$ for its available trip $q_{i,t}$
$\eta_{i,q}^t$	the number of order requests in available trip $q_{i,t}$
$\mathcal{H}_t$	the sample set after the batch dispatching in time slot t
$\mathcal{E}_t$	the set of unassigned vehicles in time slot t
$c_{i,j}^t$	the repositioning decision of vehicle $v_{i,t}$ for order request $o_{j,t}$
κ_j^t	the number of vehicles repositioned toward order request $o_{j,t}$

C. ADP-based methods for order dispatching

Finally, several works consider optimizing dispatching decisions as a sequential decision problem, and use the ADP framework to cope with the myopic dispatching in ride-sharing. For instance, Ulmer *et al.* [22] integrated temporal and spatial anticipation of order requests into ADP, and introduced offline function approximation with online rollout algorithms to ensure strategy efficiency. Al-Kanj *et al.* [23] exploited flexible dispatch strategies by ADP, and employed hierarchical aggregation to improve value function estimations with slight observations. Similarly, Yu *et al.* [24] minimized the waiting and travel times by optimization-related subproblems under the ADP framework, and revealed the properties of the approximated value function at each stage. Besides, reinforcement learning can also be viewed as an ADP-based method in the context of operational research and control theory. Xu *et al.* [25] performed the temporal-difference update rule in large-scale ride-hailing platforms. They optimized the service experience and resource utilization from a global, long-term perspective. Chen *et al.* [26] optimized order dispatching and pricing strategies jointly, of which both components conduct value function approximations in a mutual bootstrapping manner. Recently, several works [3], [11], [28], [29] have adopted neural network-based parameter approximations for value function evaluations in ADP. Based on the generalizability of neural networks, they can extend beyond the spatiotemporal state and incorporate contextual information.

The aforementioned works highlight the efficacy of ADP for order dispatching. Nevertheless, their reliance on linear value function approximations hinders the immediate application of ADP in real-world ride-sharing scenarios. The assumption that dispatching can be modelled as a linear program may not always hold for all instances due to the dynamics of vehicle capacity throughout the ride-sharing process. Besides, dispatching multiple orders to partially filled vehicles at each bitch leads to extra combinatorial complexity. The efficient solution to this remains the key challenge for the aforementioned works.

D. Our Motivation

ADP surpasses the first two threads (greedy-based and model-based heuristic methods) in flexibility and adaptability. By leveraging the power of random policies and model ensemble methods, ADP can mitigate the impact of environmental uncertainty and produce more accurate results. This motivates us to investigate order dispatching by exploiting ADP.

Inspired by existing studies, our work operates the system under the time sequence model and proposes an integrated framework (AMOD). Thereinto, available trips are generated through a graph theory-based method to handle the order combinatorial complexity. We elaborate on the Bellman update process around post-decision states, and estimate the decomposed value function with neural network-based parameter approximation. Compared to solutions from the first two threads, our work makes dispatching decisions with adaptive and foresighted considerations. It captures hidden dynamics of the environment through real-time iteration, and estimates the anticipated future value rather than a one-step return. In addition, except for order dispatching, vehicle repositioning is also an essential factor in balancing ride supply across geographic dimensions. We consider these two aspects jointly in a stochastic ride-sharing setting. Along with each batch, all vehicles are repositioned toward the areas of extractive order requests by performing a linear program. Finally, based on real-world data, we conduct extensive simulations to verify the effectiveness of AMOD.

III. SYSTEM MODEL AND DEFINITION

We consider a ride-sharing system operating on a predefined road network. Passengers who want to travel from one lo-

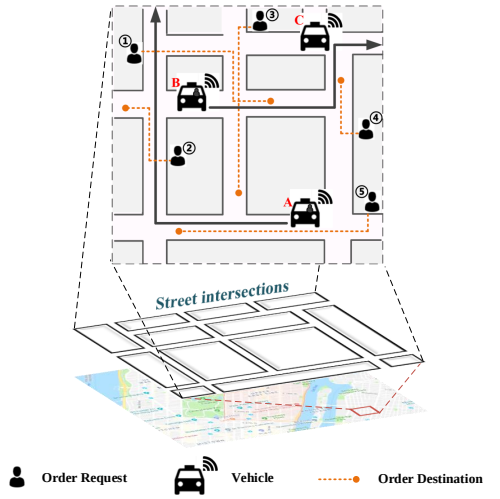


Fig. 1. **Example of a road network with three vehicles (A, B, C) and five order requests (1-5).** The orange dotted line refers to the origin and destination of a certain order request. Vehicles A and B have passengers, while Vehicle C is empty in the current time slot. The grey arrows indicate the moving direction of a vehicle due to the existing passengers.

cation to another send orders to a centralized decision platform via applications on mobile clients. In particular, vehicle status (positions, capacity constraints, and occupancy status) can be known immediately with the assistance of the onboard unit (a type of vehicular communication device). The platform is responsible for collecting order requests, and dispatching them to available vehicles within each time window. Accordingly, this section defines the system model formally as below. The main notations are listed in Table I.

Definition 1: *(Time slot) Assume the ride-sharing system operates in a fixed length of time slots (i.e., batching)* $t \in \{1, \ldots, \Gamma\}$*, where* Γ *denotes the finite time horizon. Each time slot* t *is an element in the batch sequence.*

The time sequence model is common in real ride-sharing settings.

Definition 2: *(Road network) A road network is a weighted graph, denoted by* $\mathcal{G} = (\mathcal{I}, \mathcal{A})$*, where* $\mathcal{I}$ *denotes the set of street intersections,* $\mathcal{A}$ *refers to the adjacency of these street intersections, and the weight corresponds to the travel time for each road segment.*

Street intersections are used to represent individual locations here. We assume that the vehicle can only pick up and drop passengers off at street intersections. Compared to the concept of "regions" [3], street intersections can provide the vehicle with turn-by-turn navigation guidance. We identify street intersections from OpenStreetMap (OSM) by taking *osmnx*[2], and estimate the travel time on each road segment through the derived daily mean travel time. Details of these methods could be found in [30] and [31], respectively. As shown in Fig. 1, we discretize the road network into various blocks through street intersections, assuming that the walking distance between its center to the nearest street intersection is acceptable.

[2]Osmnx is a Python library that simplifies the process of downloading, processing, and analyzing street map data from OSM.

Definition 3: *(Order request) Let* $\mathcal{O}_t = \{o_{1,t}, o_{2,t}, \ldots, o_{m,t}\}$ *denotes the set of order requests in time slot* t*. Each order request* $o_{j,t} (o_{j,t} \in \mathcal{O}_t)$ *is represented by an attribute tuple* $\langle a_j^t, b_j^t, t \rangle$*, where* a_j^t, b_j^t *and* t *capture the origin, destination, and arrived slot of order* $o_{j,t}$*, respectively.*

The origin and destination of an order are input in the form of the nearest street intersections. Notably, our work does not rely on any assumptions rigid about their distribution.

Definition 4: *(Vehicle) Let* $\mathcal{N}_t = \{v_{1,t}, v_{2,t}, \ldots, v_{n,t}\}$ *denotes the set of available vehicles in time slot* t*. Each vehicle* $v_{i,t}$ $(v_{i,t} \in \mathcal{N}_t)$ *is represented by an attribute tuple* $\langle p_i^t, c_i^t, \xi_i^t \rangle$*, where* p_i^t*,* c_i^t *denote the current position and capacity constraint (i.e., the maximum number of passengers that the vehicle can sever synchronously) of vehicle* $v_{i,t}$*, respectively;* ξ_i^t *is the list of locations that vehicle* $v_{i,t}$ *will traverse to capture subsequent orders. The combination of those three parts contributes to the current travel trajectory of a vehicle.*

A laden vehicle (*i.e.*, with a current passenger count equal to the capacity constraint) will appear "available" after completing whichever orders it was assigned. This reusable setting performs more realistic than the independent identical distribution, in which each "available" vehicle is treated as a newly arrived independent state element following the same distribution. Notably, the reusable setting promotes the temporally extended property of dispatching decisions, *i.e.*, current decisions will affect the future vehicle distribution and repositioning process.

Definition 5: *(Delay constraints) (1) the maximum pick-up delay* Φ_p*, which limits the difference between the arrival and pick-up time for an order request. (2) the maximum detour delay* Φ_d*, which limits the difference between the drop-off time of executing ride-sharing or not for an order request, i.e., detour delay is the difference between the time at which the destination reached in ride-sharing and without ride-sharing.*

Passengers are willing to accept ride-sharing only when the pick-up and detour delays are less than the constraints. Besides, for modeling simplicity, we assume that the maximum pick-up and detour delays, as well as the preceding capacity constraint are uniform global constraints.

Definition 6: *(Potential trip) A potential trip* $P_t (P_t \in \mathcal{O}_t)$ *represents the combination of order requests that an individual vehicle can serve.*

These potential trips are often unknown random variables because they depend on incoming order requests and route information that is typically uncertain. Multiple potential trips of different sizes may both contain a particular order request, *i.e.*, an order request may form part of several potential trips. Besides, a potential trip may admit more than one candidate vehicle for execution.

Definition 7: *(Trip availability) A potential trip becomes "available" for a vehicle as long as all order requests can be served, while the delay constraints get satisfying. Here, let* $\mathcal{Q}_{i,t}$ *denotes the set of available trips for vehicle* $v_{i,t}$ *in time slot* t*, therein each element* $q_{i,t} (q_{i,t} \in \mathcal{Q}_{i,t})$ *stands for the action that vehicle* $v_{i,t}$ *could execute.*

During the order dispatching phase, the action that vehicle $v_{i,t}$ can take is to execute an available trip $q_{i,t}$ from the order requests pool in each slot t, which attempts to maximize the system utility given in a later section.

Definition 8: *(Route planning) Since the weight in road network corresponds to the travel time for each road segment, route planning can be implemented by a function $\mathcal{F}(v, P)$, which returns "valid" if a route exists.*

Along with the existing passengers, the function provides the optimal route that a vehicle should take. Based on this, the vehicle can satisfy the assigned order requests with the minimum trip delay. Besides, considering the order of travel depends or partially depends on order requests generated within the temporal dimension, this route also determines the sequence in which the location should be traversed by a vehicle, *i.e.*, element ξ_i^t. Along the lines of [32], [33], route planning can be implemented via heuristic methods, such as Lin-Kernighan and simulated annealing. Specific method procedures are out of our scope in this paper.

IV. Order Dispatching Problem Formulation

This section first formulates the optimization objective for order dispatching, and then models the order dispatching process as a MDP.

A. Objective Function

The problem formulation step takes the generated available trip $q_{i,t}$ as input, and explores the optimal dispatching decisions between vehicles and available trips. Our objective is to maximize the order request response rate of long-term in the system. To this end, the corresponding optimization problem can be formulated as:

$$\begin{aligned} \max_{x_{i,q}, q_i} \lim_{\Gamma\to\infty} \frac{1}{\Gamma} \sum_{t=1}^{\Gamma} \sum_{v_{i,t}\in\mathcal{N}_t} \sum_{q_{i,t}\in\mathcal{Q}_{i,t}} \frac{x_{i,q}^t \eta_{i,q}^t}{|O_t|} \\ s.\ t. \quad C1: x_{i,q}^t \in \{0,1\}, \quad \forall v_{i,t}, \forall q_{i,t} \\ C2: \sum_{q_{i,t}\in\mathcal{Q}_{i,t}} x_{i,q}^t = 1, \quad \forall v \\ C3: \sum_{v_{i,t}\in\mathcal{N}_t} \sum_{\substack{q_{i,t}\in\mathcal{Q}_{i,t}, \\ o_{j,t}\in q_{i,t}}} x_{i,o_j}^t \le 1, \quad \forall o_{j,t} \in O_t. \end{aligned} \tag{1}$$

Here, $\lim_{T\to\infty} \frac{1}{T}(\dots)$ is the time-averaged order request response rate in the system; $\eta_{i,q}^t$ denotes the number of order requests that vehicle $v_{i,t}$ can serve through dispatched trip $q_{i,t}$ in time slot t; binary decision $x_{i,q}^t \in \{0,1\}$ denotes the trip choice of vehicle $v_{i,t}$ for the generated available trip $q_{i,t}$ in time slot t, where $x_{i,q}^t = 1$ indicates that vehicle $v_{i,t}$ executes available trip $q_{i,t}$, while $x_{i,q}^t = 0$ is the opposite. Consequently, we define the trip choice of vehicle $v_{i,t}$ for all generated available trips as $\mathcal{X}_{i,t} = \left\{ x_{i,q}^t \mid q_{i,t} \in \mathcal{Q}_{i,t} \right\}$.

The meaning of the above constraints is as follows: C1 guarantees the constraint of the binary decision's integer nature; C2 ensures that each vehicle is assigned only with an available trip; C3 denotes that each order request is part of one available trip at most; C2 and C3 are the constraints on conflicting decisions, *i.e.*, each order request can only be assigned to one vehicle.

B. Problem and Challenges

In fact, to solve the above problem in Eq. (1), we have to obtain the optimal coordination of dispatching decision variables $x_{i,q}^t$ ($v_{i,t} \in \mathcal{N}_t, q_{i,t} \in \mathcal{Q}_{i,t}$). However, the dispatching decision variable $x_{i,q}^t$ for any vehicle is binary and time-varying, which aggravates the difficulty by solving the problem at a time slot exhaustively. The system has to collect multitudinous traffic state information and make the global decision for each vehicle. Moreover, we are devoted to a more practical case in which the prior information of the order request pattern is unknown. Thus, the objective function is undoubtedly NP-hard. Since the feasible set of the problem is not convex and the complexity is enormous, conventional methods may be unadaptable to make intelligent dispatching decisions under the stochastic system property.

C. MDP Definition

The MDP model is typically used to describe almost all sequential decision-making problems. Here, given historical data, we model the order dispatching process as a MDP with discrete time and state space. Six critical elements are identified as follows:

- **Agent:** The agent corresponds to an individual decision-making unit. Although configuring the agent from a centralized perspective (*i.e.*, system-level modeling) might achieve superior performance, the complexity of action space is exponential. Hence, we model each vehicle as an agent for the straightforward definition of state transition, action, and reward. Naturally, this setting transfers the system to multi-agent contexts, and the overall objective is to maximize the long-term order request response rate.
- **State:** The state is a representation to reflect the system environment at a given time slot, including relevant information required to make actions. Accordingly, under the agent setting, the state is determined by the realization of order requests and vehicle situations, which can be given as $z_t = \{O_t, \mathcal{N}_t\}$ in time slot t. As described earlier, the former O_t is the set of current order requests; each order request $o_{j,t}$ is represented by an attribute tuple $\left\langle a_j^t, b_j^t, t \right\rangle$. The latter $\mathcal{N}_t$ is the set of current available vehicles; each vehicle $v_{i,t}$ is represented by an attribute tuple $\left\langle p_i^t, c_i^t, \xi_i^t \right\rangle$. The above information will be assembled as a state and sent to the agent in each time slot.
- **Action:** Once receiving a state in each time slot, vehicle is responsible for adopting an available trip. Therefore, under current state z_t, action d_t involves two parts, $q_{i,t}$ and $x_{i,q}^t$. The available trip $q_{i,t}$ reflects the behavior that vehicle $v_{i,t}$ could execute in current slot, and binary variable $x_{i,q}^t$ indicates the decision of vehicle $v_{i,t}$ for its available trip $q_{i,t}$. Together, we can present the current action as $d_t = \mathcal{X}_{i,t} \cdot \mathcal{Q}_{i,t} = \left\{ x_{i,q}^t q_{i,t} \mid q_{i,t} \in \mathcal{Q}_{i,t} \right\}$, with the combination of possible values of these two parts. Notably, the action still exists when no order is dispatched to the vehicle in a time slot. For simplicity, we assume this null action $d_t = \emptyset$ continues along its state transition without reward.

- **Exogenous Information:** Compared with the general MDP, the only exception is that we introduce exogenous information variables here, as the agent is incompetent to observe the system state precisely. The exogenous information variable may depend on both state and action in the system, representing the sources of uncertainty and randomness. Thus, in our case, we define exogenous information ω as the demand appearance, where ω_{t+1} denotes the arrived order requests between time slots t (that is, after the action has been made) and $t+1$ (when we have to make the next action).
- **Transition Function:** This function depicts the evolution of system states over time slots. For some or all states, we adopt the explicit equation T that relates the next state to the current state, action, and exogenous information learned by the agent after taking actions.
- **Reward Function:** The reward value is the metric for each agent to evaluate the action quality. In general, the optimization objective is related to the immediate reward. Our objective is to maximize the order request response rate of long-term in the system. This is positively correlated with the goal of an agent that tries to achieve a maximum cumulative discounted reward. Intuitively, the reward function is defined as normalized $r(z_t, d_t) = \sum_{v_{i,t} \in \mathcal{N}_t} \sum_{q_{i,t} \in \mathcal{Q}_{i,t}} \frac{x^t_{i,q} \eta^t_{i,q}}{|\mathcal{O}_t|}$. Furthermore, the reward value of each agent should satisfy the constraint conditions to ensure the validity of the results. Conversely, punitive negative values will be thoughtfully incorporated into the reward if constraints are violated in Eq. (1). These penalties discourage undesirable behavior strategically. Notably, the punitive negative value range often necessitates a trade-off between enforcing strict adherence to constraints and permitting a certain degree of constraint relaxation, thus optimizing performance while ensuring sufficient exploration field.

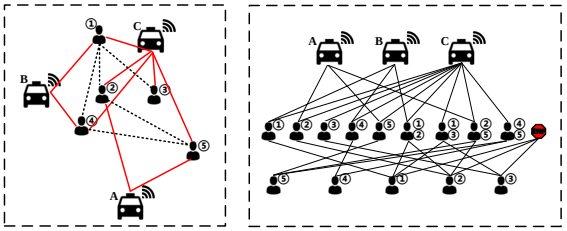
V. AMOD: Framework Design

In this section, we generate available trips for each vehicle by a graph theory-based method, and propose AMOD for order dispatching.

A. Available Vehicle Trips Generation

In contrast to the ride-hailing service (each vehicle can serve only one passenger at a time), order dispatching in ride-sharing services exhibits inherently challenging. When multiple order requests are combined and assigned to a single vehicle, it is computationally complex to generate available trips for each vehicle. Here, we generate the available trips for each vehicle through a graph theory-based method. We get forward in two steps:

Firstly, we construct an order-vehicle graph (OV-graph) to reflect the ride-sharing potential, as shown in Fig. 2(a). Specifically, the OV-graph describes which order requests can be picked up by a vehicle, and which order requests can be combined into a potential trip. Based on the output of function $\mathcal{F}(v, P)$, order request $o_{j,t}$ and vehicle $v_{i,t}$ are connected when the vehicle can serve the order under the delay



(a) (b)

Fig. 2. (a) OV-graph of orders and vehicles. (b) OPV-graph of potential trips and corresponding pick-up vehicles.

Algorithm 1: Generate Available Trips For Vehicles

Input: the OV-graph
Output: the set of available trips $\mathcal{Q}_t$

for each time slot t **do**
  Each vehicle $v_{i,t} = \langle p^t_i, c^t_i, \xi^t_i \rangle$, $v_{i,t} \in \mathcal{N}_t$ **do**
  Initialize potential trip $P_{i,t} \leftarrow \emptyset$, the set of available trips $\mathcal{Q}_{i,t} \leftarrow \emptyset$
  for edge $e(o_{j,t}, v_{i,t})$ in predefined VO-Graph **do**
    Add potential trips of size one
    $\mathcal{Q}^1_{i,t} \leftarrow P_{i,t} = \{o_{j,t}\}$
    Add edge $e(v_{i,t}, P_{i,t})$ and $e(o_{j,t}, P_{i,t})$
  for elements $\{o_{1,t}\}, \{o_{2,t}\} \in \mathcal{Q}^1_{i,t}$, and edge $e(o_{1,t}, o_{2,t})$ in predefined VO-Graph **do**
    Add potential trips of size two
    if $\mathcal{F}\left(v_{i,t}, \{o_{1,t}, o_{2,t}\}\right) =$ *"valid"* **then**
      $\mathcal{Q}^2_{i,t} \leftarrow P_{i,t} = \{o_{1,t}, o_{2,t}\}$
      Add edge $e(o_{1,t}, P_{i,t})$, $e(o_{2,t}, P_{i,t})$, and $e(v_{i,t}, P_{i,t})$
  for $\forall f \in \{3, \ldots, c^t_v\}$, potential trips $P^1_{i,t}, P^2_{i,t} \in \mathcal{Q}^{f-1}_{i,t}$, and $\left|P^1_{i,t} \cup P^2_{i,t}\right| = f$ **do**
    Add potential trips of size f
    if $\forall j \in \{1, \ldots, f\}$, $(P^1_{i,t} \cup P^2_{i,t}) \backslash o_{j,t} \in \mathcal{Q}^{f-1}_{i,t}$, and $\mathcal{F}\left(v_{i,t}, P^1_{i,t} \cup P^2_{i,t}\right) =$ "$valid$" **then**
      $\mathcal{Q}^f_{i,t} \leftarrow P_{i,t} = P^1_{i,t} \cup P^2_{i,t}$, $P^1_{i,t} \cup P^2_{i,t} = \{o_{1,t}, \ldots, o_{f,t}\}$
      Add edge $e(o_{j,t}, P_{i,t})$ for $\forall o_{j,t} \in P_{i,t}$, and edge $e(v_{i,t}, P_{i,t})$
  return $\mathcal{Q}_{i,t} \leftarrow \bigcup_{j \in \{1,\ldots,c_v\}} \mathcal{Q}^j_{i,t}$

constraint (*i.e.*, maximum pick-up and detour delays Φ_p, Φ_d). The edge of order-vehicle pairs is denoted by $e(o_{j,t}, v_{i,t})$. Meanwhile, considering the origin and destination, two order requests $o_{1,t}$, $o_{2,t}$ are combined pairwise if a vehicle can serve them synchronously under the delay and capacity constraints. The corresponding delay consumption is associated to edge $e(o_{1,t}, o_{2,t})$.

Secondly, by exploring the clique partition (the closed-loop sub-graph) in OV-graph, we further construct a graph of potential trips and corresponding pick-up vehicles (OPV-graph), as shown in Fig. 2(b). That is to say, a potential

trip P_t is available only if the clique containing one vehicle and multiple orders is closed-loop. Recall that a potential trip P_t is a combination of order requests, which can be served by at least one vehicle within the delay and capacity constraints. Two different edges in OPV-graph can be denoted by $e(v_{i,t}, P_t)$ and $e(o_{j,t}, P_t)$, respectively. The former edge $e(v_{i,t}, P_t)$, between a pick-up vehicle $v_{i,t}$ and a potential trip P_t, reflects that the potential trip is available for the pick-up vehicle, *i.e.*, $\exists\ e(v_{i,t}, P_t) \Leftrightarrow \mathcal{F}(v_{i,t}, P_t)\ is\ valid$. Likewise, the latter edge $e(o_{j,t}, P_t)$, between a potential trip P_t and an order request $o_{j,t}$, reflects that the order request exists in the potential trip, *i.e.*, $\exists\ e(o_{j,t}, P_t) \Leftrightarrow o_{j,t} \in P_t$. In addition, for order requests that corresponding constraints cannot be satisfied, a red identifier is added to OPV-graph.

Based on the orchestrated OPV-graph, the processes of generating available trips for each vehicle are summarized in **Algorithm 1**. Notably, with the size of the potential trip, the computation complexity of the algorithm will lift substantially. Here, **Lemma 1** is introduced to accelerate the algorithm. Based on this, we conduct a pre-classification step before computing available trips. We only need to confirm whether a potential trip P_t is available for a vehicle when all of its sub-trips P_t^{sub} (comprising all order requests in P_t except one) are available and already represented as edges $e(v_{i,t}, P_t^{sub})$ in current OPV-graph.

Lemma 1: *(Sub-trip Availability) A potential trip P_t is available for a vehicle signifies that all of its sub-trips $P_t^{sub} = P_t \backslash o_{j,t}$ $(o_{j,t} \in P_t)$ are equally available for this vehicle. That is,*

$$\exists\ e(v_{i,t}, P_t) \Rightarrow \exists\ e(v_{i,t}, P_t \backslash o_{j,t}), \forall o_{j,t} \in P_t.$$

B. Value Function Approximation

Generally, MDP can be solved by linear or dynamic programming methods. However, these solutions may not always be feasible as their model does not hold for the stochastic ride-sharing setting with arbitrary capacity constraints of vehicles. The value function approximation presents nonlinear with uncertain state transition probability and immediate reward, since multiple orders are assigned to one vehicle at each batch. Therefore, we investigate this stochastic dynamic programming by exploiting ADP. This method is able to approximate the value function and handle the order combinatorial complexity.

In the ADP setting, order dispatching is performed through a function approximation structure, which estimates the anticipated future value of executing a particular trip. Specifically, the approximate structure focuses on the agent. Each agent confronts the control and sequential decision-making under uncertainty. At a time slot t, the agent observes the current state of the environment as z_t, selects and takes action d_t from the admissible action space based on its policy π, where the policy π is derived as a mapping from the current state to the corresponding action. After that, the agent samples the exogenous information ω_{t+1}, and transfers to a new state z_{t+1} with the engineered transition function $z_{t+1} = T(z_t, d_t, \omega_{t+1})$. Finally, the agent will receive an immediate reward $r(z_t, d_t) = \mathbb{E}[r_{t+1} \mid z_t, d_t]$ and consider an anticipated reward as it proceeds.

Considering the long-term influences, the recursive state value function[3] $V^{\pi}(z_t)$ is defined under the policy $\pi(z_t)$. It maps state z_t to the expection of cumulative discounted reward value. Specifically, cumulative discounted reward U_t measures the total expected reward accumulated over time slots, while considering the influence of future rewards being discounted. That is,

$$\begin{aligned} U_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{\Gamma-t-1} r_T \\ &= \sum\nolimits_{k=0}^{\Gamma-t-1} \gamma^k r_{t+k+1}, \end{aligned} \tag{2}$$

where discount factor $\gamma \in (0, 1)$ indicates the importance of the predicted future rewards, k is the index of time slot.

According to the Bellman Equation, the state value function $V^{\pi}(z_t)$ can be transformed to the temporal difference form:

$$\begin{aligned} V^{\pi}(z_t) &= \mathbb{E}\left[\sum\nolimits_{k=0}^{\Gamma-t-1} \gamma^k r_{t+k+1} \mid z_t\right] \\ &= \mathbb{E}\left[\left(r_{t+1} + \sum\nolimits_{k=1}^{\Gamma-t-1} \gamma^k r_{t+k+1}\right) \mid z_t\right] \\ &= \mathbb{E}\left[\left(r_{t+1} + \gamma \sum\nolimits_{k=0}^{\Gamma-t-2} \gamma^k r_{t+k+2}\right) \mid z_t\right] \\ &= \mathbb{E}\left[(r_{t+1} \mid z_t) + \gamma \mathbb{E}\left(\sum\nolimits_{k=0}^{\Gamma-(t+1)-1} \gamma^k r_{(t+1)+k+1} \mid z_{t+1}\right)\right] \\ &= r(z_t, d_t) + \gamma \mathbb{E} V(z_{t+1}), \end{aligned} \tag{3}$$

where $\mathbb{E}$ represents the expectation, and Γ is the finite time horizon.

Along with the process above, the goal of the agent is to make an optimal control policy $\pi^*(z_t) \rightarrow d_t^*$ that maximizes the expection of cumulative discounted reward value. Consequently, the optimization problem in this paper is converted to an optimal value function $V^{\pi^*}(z_t)$, which is expressed as:

$$V^{\pi^*}(z_t) = \max_{\pi \rightarrow d_t} \left[r(z_t, d_t) + \gamma \mathbb{E} V(z_{t+1}) \right], \tag{4}$$

$$\text{s.t. constraints in (C1) – (C3).}$$

This recursive value optimal function equation (Eq. (4)) resolves the sequential decision problem into a series of shorter, tractable time steps, where the action and exogenous information have to be determined in each step. Explicitly, the control policy $\pi^*(z_t)$, which satisfies Eq. (4) is guaranteed to become the optimal policy. The optimal action for state z_t is easily obtained as

$$d_t^* = \underset{d_t}{\text{argmax}}\ V(z_t). \tag{5}$$

C. Bellman Update Process around Post-decision States

The majority of dynamic programming works perform Bellman updates around pre-decision states, where the ability to place expectations over future knowledge is taken for

[3]For ease of presentation, we have not distinguished the concept of "state value function" and "value function" in this paper.

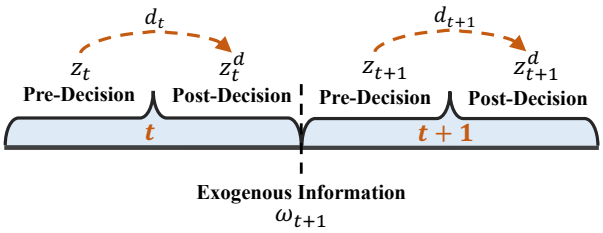


Fig. 3. Time convention for the pre-decision and post-decision state.

granted. However, this is not legitimate enough to disregard the computation process of state transition matrixes, *i.e.*, the expectation in Eq. (4) is rigid to compute.

Therefore, to avoid approximating the embedded expectations explicitly, we reconstruct the Bellman update process around the post-decision state. Indeed, the post-decision state conforms to the sequential nature of decision-making, which is especially beneficial in dynamic programming. Here, the difference in dimensionality between the pre-decision and post-decision states is typically contingent on the action selection and state definition in the current batch. This difference reflects the change in the information demands between the pre-decision and post-decision states. Empirical studies [35], [36] proved that adopting compact post-decision states can provide computation simplicity and lower dimensionality than pre-decision states.

We start the exposition by contrasting pre- and post-decision states to clarify this distinction. As shown in Fig. 3, the pre-decision state z_t is determined before the agent takes action d_t in the time sequence model. In contrast, there are situations in which it is beneficial to represent the state that follows an action in the model. The post-decision state z_t^d is reached immediately after the agent takes action d_t, but before a new exogenous information ω_{t+1} arrives. Then, with exogenous information ω_{t+1} realized, the state will transfer to the next pre-decision z_{t+1}. Thus, our sequence model evolves as $\left(z_0, d_0, z_0^d, \omega_1, z_1, d_1, z_1^d, \cdots, z_t, d_t, z_t^d, \omega_{t+1}, z_{t+1} \cdots\right)$. Owing to above concepts, the corresponding state transition function is rewritten as:

$$z_t^d = T^d\left(z_t, d_t\right), z_{t+1} = T^\omega\left(z_t^d, \omega_{t+1}\right). \tag{6}$$

Recall that the agent is expected to execute an action under optimal control policy over each time slot. The post-decision state value $V^d(z_t^d)$ is derived as the maximum expected cumulative discounted reward the agent achieves, which is equivalent to the expectation of next pre-decision state value $V(z_{t+1})$. That is,

$$V^d\left(z_t^d\right) = V^d\left[T^d\left(z_t, d_t\right) \mid z_t^d, \omega_t\right] = \mathbb{E}\left[V\left(z_{t+1}\right) \mid z_t^d, \omega_{t+1}\right]. \tag{7}$$

Formally, we can rewrite Eq. (4) with the following form by substituting Eq. (7) into it,

$$V\left(z_t\right) = \max_{\pi \to d_t}\left[r\left(z_t, d_t\right) + \gamma V^d\left(z_t^d\right)\right]. \tag{8}$$

Since non-linear post-decision state value $V^d(z_t^d)$ is integrated into the function approximation, Eq. (8) belongs to non-linear. Compared to the linear counterpart, it cannot get updated by dual variables from the linear programming solution in current sequential optimization. Meanwhile, non-linear value function approximation often leads to non-convex optimizations, making finding the optimal solution in the parameter space complex. This results in training instability and hinders the convergence towards a suitable value. Hence, we configure a quadratic decomposition for non-linear function approximation in the next subsection.

D. Value Function Decomposition

To approximate Eq. (8) linearly, a general method is to evaluate the state value for all possible post-decision states and integrate these values as constants. However, the post-decision state is polynomial in the number of available actions, but exponential in the number of vehicles in terms of space complexity. Their associated state function typically involves more intricate computational procedures. Constrained by computational resources, accurately evaluating the state value for all possible post-decision states may not be feasible in practice.

For the above challenges, we apply a quadratic decomposition to convert the overall value function into a linear combination of individual value functions of vehicles. First, after each batch dispatching, we assume that the platform will passively discard the remaining unassigned order requests (*i.e.*, unsatisfied demand is immediately lost, probably because there are no available or suitable vehicles nearby). Actually, this pattern indicates that the current vehicle situation $\mathcal{N}_t^d$ will determine the post-decision state, leaving the order request element O_t as an empty set. We therefore get:

$$z_t^d = T^a\left(z_t, d_t\right) = \left\{\varnothing, \mathcal{N}_t^d\right\} = \mathcal{N}_t^d. \tag{9}$$

Accordingly, in respect of the post-decision state value function, we have:

$$V\left(z_t^d\right) = V\left(\mathcal{N}_t^d\right). \tag{10}$$

At this moment, we can train a central agent (a central agent refers to the arbitrator, which combines the return of all agents in command arbitration.) by crowdsourcing the experience of all vehicles in this multi-agent context. Then, we apply the central agent to all other agents to generate their dispatching decisions, which means that a single experience can lead to multiple updates. From a local viewpoint, the decomposition assumes that the overall reward is equal to the sum of individual rewards, since each vehicle enables to have rewards associated with states. In this way, the overall value function can be additively decomposed into the individual value function of vehicles. The overall value is therefore converted to:

$$V\left(\mathcal{N}_t^d\right) = \sum_i^n V_i\left(\mathcal{N}_t^d\right), \tag{11}$$

The relevant theoretical proof for this may refer to *ref.* [37].

Second, we approximate the individual value function of a vehicle through a quadratic decomposition. The premise underlying the decomposition is that the behaviours of other

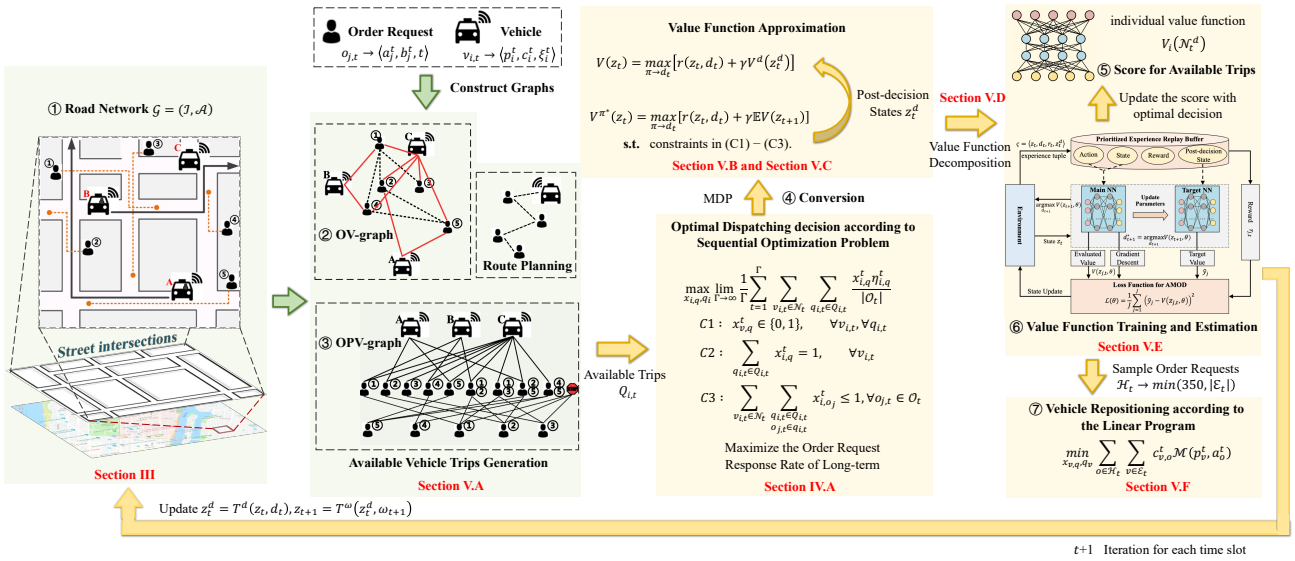


Fig. 4. **Schematic overview of our proposed method, green and yellow boxes indicate action generation and action evaluation component, respectively.** ① Example of a road network with three vehicles and five order requests. ② We construct the OV-graph to reflect the ride-sharing potential. A potential trip is available only if the clique (sub-graph) in OV-graph is closed-loop. ③ We construct the OPV-graph to generate available trips for each vehicle. We only confirm the availability of a potential trip when all of its sub-trips are available. ④ We convert the sequential optimization problem to an optimal value function. ⑤ We offline score the individual value function of available trips under the current state via the neural network. ⑥ We make an order dispatching by estimating the optimal overall value function with neural network-based parameter approximation. ⑦ We perform vehicle repositioning to balance ride supply across geographic dimensions.

vehicles have a negligible impact on the long-term utility of a vehicle (individual state value) in order dispatching. This is reasonable since, from a macroscopic perspective, the long-term utility of a vehicle only gets influenced by its interaction with other vehicle trajectories, which will not vary dramatically during a time slot. Therefore, the output result is insusceptible whether other vehicles adopt pre- or post-decision states in the state value computation process. We have:

$$V_i\left(\mathcal{N}_t^d\right) = V_i\left(\{v_{i,t}^d, \boldsymbol{v}^{\boldsymbol{d}}_{\mathcal{N}\setminus\{i\},t}\}\right) \approx V_i\left(\{v_{i,t}^d, \boldsymbol{v}_{\mathcal{N}\setminus\{i\},t}\}\right), \quad (12)$$

where $\mathcal{N}\setminus\{i\}$ represents the set of all vehicles except vehicle $v_{i,t}$, and v_t^d denotes the post-decision state of a vehicle in the time slot t.

This logical transition between pre- and post-decision states is crucially essential for reducing the evaluation number of the non-linear value function. From this, the second part of an individual value function $\boldsymbol{v}_{\mathcal{N}\setminus\{i\},t}$ will not lie on the exponential post-decision state of vehicles, which can be considered constants now.

Accordingly, we rewrite Eq. (11) with the following form by incorporating Eq. (12), (11):

$$V\left(\mathcal{N}_t^d\right) = \sum_i^n V_i\left(\{v_{i,t}^d, \boldsymbol{v}_{\mathcal{N}\setminus\{i\},t}\}\right). \quad (13)$$

We calculate individual state values V_i versus possible post-decision states $v_{i,t}^d$, and hereafter aggregate these values of all vehicles as a linear combination into the non-linear programming in Eq. (8).

E. Value Function Training and Estimation

Despite encountering potential domain adaptation issues, neural networks remain indispensable for tackling the intricacies of order dispatching problems. Their strengths lie in handling complex decision-making, modeling non-linearity, circumventing the curse of dimensionality, facilitating generalization, and enabling data-driven learning. Here, to estimate the value function $V\left(z_t^d\right)$ over the post-decision state, we propose AMOD, a systematic solution that estimates the decomposed value function with neural network-based parameter approximation. The schematic overview of AMOD is illustrated in Fig. 4.

Throughout the training process, the input training samples that neural networks expect should be distributed independently for exploitation and exploration. Nevertheless, due to the high correlation of continuous states, value function estimations through on-policy updates is easy to perform non-uniform overestimation. This will introduce instability and adverse impacts on the actual value. Therefore, we adopt off-policy updates in practice, which separate the policy of behaviour and target. To do this, we directly store the set of available trips for each vehicle during the sample collection, rather than double-counting them for each iteration. Notably, this step also has the advantage of shrinking the generation time for available trips. Hereafter, the agent offline scores the individual value functions of these available trips in the current state, and expects to make an optimal action by solving the sequential optimization in Eq. (8) using the neural network. Finally, each agent updates the saved post-decision state value with the target value through this dispatching decision.

Nevertheless, the optimal action is generally bounded by a finite search field, which relies on the quantity and quality

of the training data. To introduce randomness in the optimal action selection, we apply and modify the ϵ-greedy strategy for ensuring adequate exploration, where ϵ is a decreasing parameter to achieve a tradeoff between exploitation and exploration. Specifically, the agent gets the action that maximizes the $V(z_t)$ in Eq. (8) with the probability $1-\epsilon$ (exploiting), and the other action with the tiny probability ϵ (exploring).

Meanwhile, the value function update process in AMOD largely follows the Double Deep Q Network (DDQN) method [38], [39]. Instead of performing bootstrapping directly, a target network mechanism exists in AMOD to disrupt the relevance. Specifically, two neural networks with the same structure but different parameters are maintained. The target neural network aims to acquire the temporal difference within the one-step return value while the main neural network evaluates the current post-decision state value. For the sake of learning stability, the weight parameters $\acute{\theta}$ of the target neural network are updated periodically (spaced a few training steps) by the counterpart θ of the main neural network. Their update rule follows $\acute{\theta} = \zeta\theta + (1-\zeta)\acute{\theta}$ with $\zeta \ll 1$. Note that the one-step return method (*i.e.*, TD(0)) in the target neural network is simply based on the immediate reward value. Then, the target network mechanism is used to generate the one-step target value $\acute{y}$ as:

$$\acute{y} = r_t + \gamma V\left(T(z_t, \text{argmax}_d\ V\left(z_t^d\right)), \acute{\theta}\right). \tag{14}$$

Furthermore, to ensure the stability of this neural network-based nonlinear approximation, we utilize a prioritized experience replay buffer to reuse the pre-existing experiences and break the correlation among data training. During training sample collection, the agent stores its experience tuple $\varsigma = \left\langle z_t, d_t,\ r_t,\ z_t^d \right\rangle$ into the experience replay buffer, which involves its current state and available action set. The arrived experience samples are used to train the parameters of the value function approximation in neural networks. Practically, this replaying enables the agent to extract a minibatch of previous experience samples from the replay buffer for learning at each iteration.

Since $V(z_t, \theta) = \acute{y}$ is possible when the error is smaller, we attempt to find a set of parameters for the value function approximation. Thus, the loss function is interpreted as the Euclidean distance between the target value and the estimated value of value functions. For a minibatch of J experience sample $\left\{\varsigma_j\right\}_{j=1}^{J}$, it can be converted as:

$$\begin{aligned}\mathcal{L}(\theta) =& \frac{1}{J}\sum_{j=1}^{J}\left(\acute{y}_j - V\left(z_{j,t}, \theta\right)\right)^2 \\ =& \frac{1}{J}\sum_{j=1}^{J}\left(r_{j,t} + \gamma V\left(T(z_{j,t}, \text{argmax}_d\ V\left(z_{j,t}^d\right), \acute{\theta}\right)\right. \\ & \left. - V\left(z_{j,t}, \theta\right)\right)^2,\end{aligned} \tag{15}$$

where each experience samples $\varsigma_j = \left\langle z_{j,t}, d_{j,t},\ r_{j,t},\ z_{j,t}^d \right\rangle$ is used to update the parameter θ toward the target value

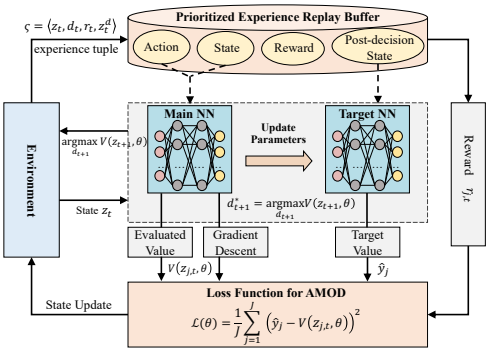


Fig. 5. The update process of value function.

by miniminzing loss function $\mathcal{L}(\theta)$, and a gradient guiding updates of θ can be calculated by $\frac{\partial \mathcal{L}(\theta)}{\partial \theta}$.

Figure 5 shows the update process of the value function, and more details of AMOD are summarized in **Algorithm 2**. With a sample path in each episode[4], we estimate decomposed value function in Eq. (8) for each time slot under the parameter approximation. By the constructed loss function, the neural network updates parameters and further obtains optimal behavior decisions. After that, we update the post-decision and next pre-decision states (*e.g.*, z_t^d and z_{t+1}) via Eq. (6), respectively. Finally, to continue the process to the next slot, the aforementioned value function approximation is solved again until a finite time horizon within an episode. AMOD is terminated when it reaches the given maximum number of episodes, and the desired value function becomes stable as well.

Remark. Unlike the standard experience replay, the agent has partial knowledge of the transition function in our case. Therefore, it will preferentially extract a valuable minibatch sample from the replay buffer to deal with the challenge of reward scarcity.

F. Vehicle Repositioning

Vehicle repositioning significantly impacts global supply-demand distributions by guiding vehicles to specific locations. Therefore, along with each batch dispatching, we perform vehicle repositioning for non-occupied vehicles (*i.e.*, no order requests were assigned to them) to balance ride supply across geographic dimensions. Unlike most previous works [14], [34], our repositioning decisions are not dependent only on unassigned order requests in this paper. This is because previous works failed to ensure that a sufficient number of vehicles were repositioned in the system. Generally, the number of repositioned vehicles is the minimum of non-occupied vehicles and unassigned order requests. Explicitly, in that way, the majority of vehicles will resist repositioning and easily be trapped in low-demand areas where order requests are infrequent.

To this end, we sample $\mathcal{H}_t \rightarrow min(350, |\mathcal{E}_t|)$ order requests from the order requests pool O_t after each batch

[4]Episode and time slot are both periods in the training process. An episode means one complete play of the agent interacting with the environment, which usually contains many slots.

Algorithm 2: AMOD: **A**DP-based **M**ulti-driver **O**rder **D**ispatching method

Input: agent set $\mathcal{N}$, learning rate β, discount factor γ, exploration factor ϵ, replay buffer, minibatch size J

Output: neural network parameters θ

Each vehicle $v_{i,t} \in \mathcal{N}$ **do**
  Initialize experience replay buffer, main neural network with random weight θ, target value function V with $\acute{\theta} = \theta$.
  for each episode t **do**
    Set t=0, obtain the initial state z_0 by randomly positioning vehicles
    Randomly pick a sample path
    for each slots of episode **do**
      Implement **Algo. 1** to generate the available trips $\mathcal{Q}_{i,t}$
      Derive an action d_t based on the ϵ-greedy strategy in the current z_t
      Store experience tuple $\varsigma = \langle z_t, d_t,\ r_t,\ z_t^d \rangle$ into the replay buffer
      if t % updateFrequency == 0 **then**
        Extract a minibatch of J experience samples from the replay buffer
        for each experience sample $\{\varsigma_j\}_{j=1}^{J}$ **do**
          Update parameter θ by minimizing $\mathcal{L}(\theta)$ as: $\mathcal{L}(\theta) = \frac{1}{J}\sum_{j=1}^{J}\left(\acute{y}_j - V\left(z_{j,t}, \theta\right)\right)^2$
          Perform a gradient descent step on $\mathcal{L}(\theta)$
          Update parameter $\acute{\theta}$ periodically by θ
      Update the state via Eq. (6): $z_t^d = T^d\left(z_t, d_t\right), z_{t+1} = T^{\omega}\left(z_t^d, \omega_{t+1}\right)$

dispatching, where $\mathcal{E}_t$ denotes the set of non-occupied vehicles in time slot t. Then, we reposition all vehicles to move to the areas of these samples by performing the linear program in Eq. (16), which aims to minimize the sum of travel delay.

$$\begin{aligned} \min_{x_{v,q}, q_v} \ & \sum_{o\in\mathcal{H}_t}\sum_{v\in\mathcal{E}_t} c_{v,o}^t \mathcal{M}(p_v^t, a_o^t) \\ s.t. \quad & C1: c_{v,o}^t \in \{0,1\}, \quad \forall v, \forall o \\ & C2: \sum_{v\in\mathcal{E}_t} c_{v,o}^t \le \kappa_o^t, \quad \forall o \\ & C3: \sum_{o\in\mathcal{H}_t} c_{v,o}^t = 1, \quad \forall v. \end{aligned} \tag{16}$$

Here, binary decision variable $c_{v,o}^t$ indicates whether vehicle v_t is dispatched toward order request o_t; κ_o^t denotes the number of vehicles that are repositioned toward order request o_t, which can be obtained by a Floor or Ceiling function[5] with satisfying $\sum_{o\in\mathcal{H}_t}\kappa_o^t = |\mathcal{E}_t|$; $\mathcal{M}(p_v^t, a_o^t)$ denotes the travel time from position p_v^t of vehicle v_t to origin a_o^t of order request o_t, which is obtained by the method mentioned above.

The meaning of the above constraints is as follows: C1 guarantees the constraint of the binary decision's integer nature; C2 is the constraint on the number of vehicles repositioned toward an order request; C3 ensures that each vehicle is repositioned toward only an order request.

[5]Floor function $\kappa_o^t = \left\lfloor \frac{|\mathcal{E}_t|}{300} \right\rfloor$ for $|\mathcal{E}_t| > 300$, and Ceiling function $\kappa_o^t = \left\lceil \frac{|\mathcal{E}_t|}{300} \right\rceil$ for $|\mathcal{E}_t| < 300$.

VI. Evaluation Results

This section conducts extensive experiments based on real-world data from New York City taxi daily operations. To verify the effectiveness of AMOD, results are exhibited from various perspectives after introducing the settings and dataset.

A. Simulation Settings

We provide quantitive performance analyses for order dispatching by populating ride-sharing vehicles over the Manhattan borough of New York City. Unless otherwise stated, there are 1000 vehicles with a capacity constraint of 4 passengers (excluding the driver seat). The maximum pick-up delay is initially 150s, while the maximum detour delay is set at double that value. For comparison measurements, we concentrate on varying these values to affect instance sizes in later analysis. Multiple order requests are combined and assigned in a batch sequence model with a slot size of 40s. This makes sense by considering the computation cost of AMOD and the order response time for a passenger. Furthermore, we test instances generated based on real-world data to configure and update parameters for implementing AMOD. For hyper-parameters, the initial learning rate and discount factor are set to 0.015 and 0.9, respectively. All experiments are conducted in Python 3.8 and tested on a processor (AMD®Threadripper™PRO5995WX@2.70GHz).

For performance comparison, we compare AMOD with the following four baselines.

1) **PMA [15]:** A greedy-based method, which employs the maximum matching algorithm and iteratively determines the merge-ability of each order-vehicle pair.
2) **HCRS [34]:** A heuristic method combines integer linear programming and model-predictive control to explore possible actions and optimize dispatch decisions.
3) **DRL [29]:** A DDQN-based model-free method, in which each vehicle learns its own dispatch decisions by interacting with the external environment.
4) **NoRS:** Apply AMOD without considering ride-sharing. Note that it follows as a particular case of AMOD by dispatching only one order request to a vehicle.

Besides, except for the order request response rate (RR), the following metrics are considered to evaluate AMOD quantitatively: 1) Detour and pick-up delays, refer to Definition 5; 2) Computational time of AMOD (CTA), involves the time of generating available trips, value function approximation, and vehicle repositioning.

B. Network Training Details

The neural network model adopted in our simulations incorporates various inputs, including current vehicle and pending order request information. These inputs are derived based on associated trajectories, and undergo embedding before being processed by the LSTM (Long Short Term Memory network). Specifically, location embedding is computed through a two-layer neural network, which aims to estimate the travel time between two locations.

Then, supplementary context (the count of nearby vehicles for a single agent, the count of arrival order requests, delay constraints, decision epoch, *etc.*) is taken to connect to the output of LSTM and processed through two dense layers. The overall architecture estimates the value function, with mean squared error and Adam optimizer employed for loss optimization. Notably, the stochastic action space necessitates exploration despite deterministic transitions and rewards. The vehicle offline yields the individual value function in the current state. During online execution, assignments maximizing precomputed offline value functions guide order dispatching. Overall, our model employs neural networks to enhance order dispatching via informed decision-making and efficient resource utilization.

C. Dataset

1) Road Network: We consider the complete road network of the Manhattan borough in New York City, which contains 4121 road nodes (street intersections) and 9417 edges (road segments). Following the description in Definitions 2 and 8, we can identify street intersections and estimate the travel time on each road segment.

2) Historical Trip Record: New York Taxi and Limousine Commission provides the public NYC dataset [40]. It contains historical trip records for three travel services (yellow and green taxis, for-hire vehicles) in recent years. Similar to the previous works [15], [24], we conduct experiments based on the demand distribution from yellow taxi trace data in the Manhattan borough. The subset we used included historical order requests for different periods of the day and different days of the week. For each trip record, we can extract the pick-up and drop-off time, location (the coordinate of latitude and longitude), pick-up delay (multiple of the time slot), travel distances, and driver-reported passenger count, where the nearest street intersection is mapped to the corresponding location. Furthermore, we analyzed the trip records for a week to appear the demand distribution pattern. After data cleaning, there were an average of 415,133 order requests per day, and the average travel distance of passengers is 3.257 miles. Fig. 6 displays the pick-up and drop-off location distribution of historical trip records over road segments in a someday. Obviously, order requests are roughly distributed sparsely across the road network.

D. Numerical Results

To explore the ability of AMOD for spatiotemporal order dispatching, we rewind actual order requests from trip records as demands. Specifically, we consider the trip data from July 2017 for training, and one-week data from August 2017 for validation. During this stage, AMOD is trained over 12000 epochs, while recent 3000 experience tuples are extracted as a replay buffer. Then, we evaluate AMOD over 24 hours using the trip data of the penultimate week from August 2017, and take the average value. The initial location of the vehicles uniformly corresponds to the sampled position from a historical demand distribution at midnight.

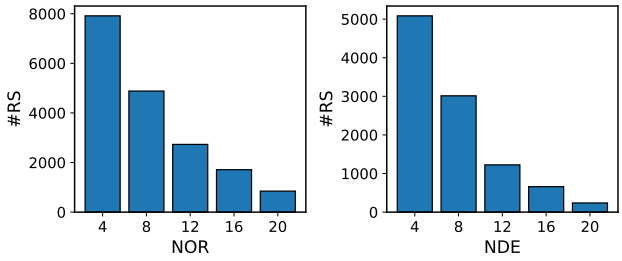


Fig. 6. The pick-up and drop-off location distribution of historical trip records. (#RS: the number of road segments; NOR: the number of order requests originated on the road segment; NDE: the number of order requests destined on the road segment.)

1) Robustness Analysis: We demonstrate the robustness of AMOD versus the duration of each time slot (*i.e.*, the aggregate duration of order requests) and the number of order requests (*i.e.*, the demand density). The displayed results represent the mean value over a week of data in the Manhattan borough. Notably, to vary the number of order requests, we cumulate daily demands for sequential days or eliminate demands over each period.

We first vary the duration of the time slot from 20s to 60s, and display results to confirm the robustness of this change. As shown in Fig. 7 (a), we observe that the order request response rate slightly improves from 71% to 75% with the increase of the duration of the time slot. Indeed, this increase seems practically negligible. Meanwhile, as shown in Fig. 7 (b) and (c), the detour delay decreases rapidly, while the pick-up delay exhibits an opposite trend. This is reasonable because more order requests can be combined and assigned to vehicles as the duration of the time slot increases. Thus, a larger aggregate duration can enable the platform to achieve better order dispatchings. Although passengers have to wait longer to receive their responses, the delivery route planning will be more efficient. Besides, the computational time of AMOD also increases since more order requests need to be combined and assigned. Specifically, it achieves a time of 16s in the initial stage, and enhances to 35s when the duration of each time slot is 60s.

Next, we show robustness results for the number of order requests in Fig. 8. We vary the number of order requests from 0.5 to 1.5 times compared to the original. As expected, increased order requests negatively impact both performance metrics. Even so, AMOD can still serve about 60% of order requests even when there exist a massive number of order requests (*i.e.*, $\times 1.5$). Especially, as shown in Fig. 8 (b) and (c), detour and pick-up delays increase with the increase of order requests. Nonetheless, these increases are both within acceptable threshold ranges. In addition, although NoRS has

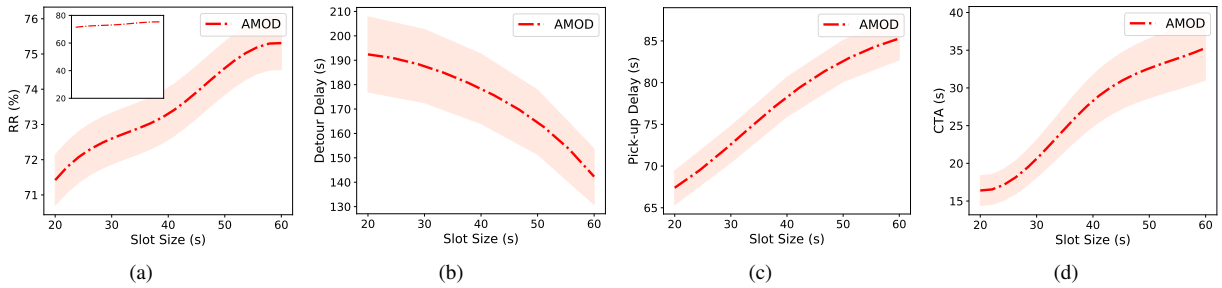


Fig. 7. **Performance metrics versus the slot size:** (a) Order request response rate. (b) Detour delay. (c) Pick-up delay. (d) Computational time of AMOD.

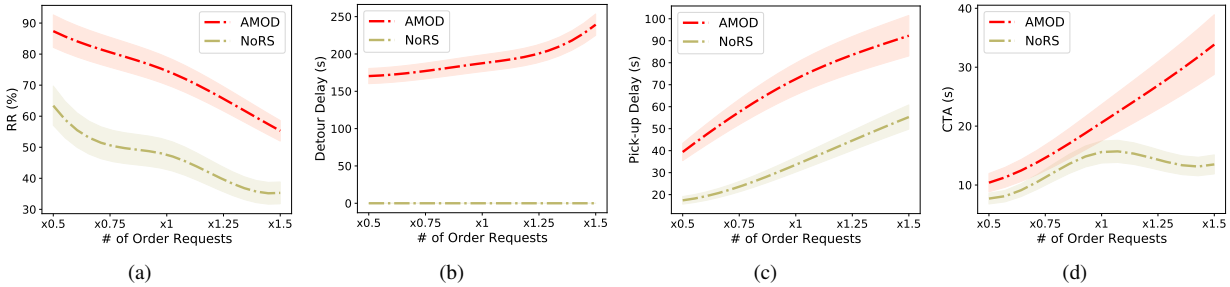


Fig. 8. **Performance metrics versus the number of order requests:** (a) Order request response rate. (b) Detour delay. (c) Pick-up delay. (d) Computational time of AMOD.

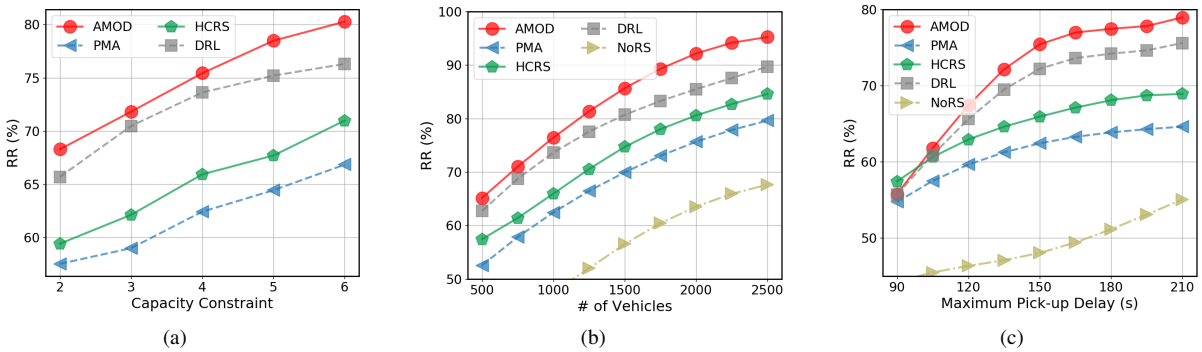


Fig. 9. (a) Order request response rate versus the capacity constraint. (b) Order request response rate versus the number of vehicles. (c) Order request response rate versus the maximum pick-up delay.

zero detour delay, it performs badly in order request response rate since it does not consider ride-sharing.

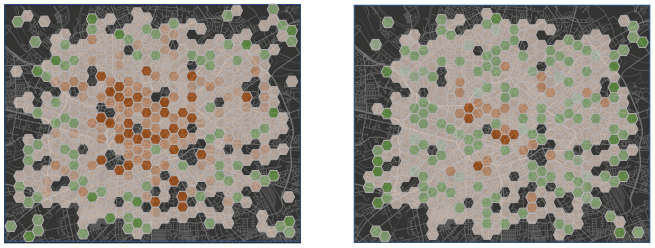
To this end, it can be inferred that AMOD remains robust with the changes in the duration of the time slot and the number of order requests.

2) Performance Evaluation: In this part, we compare AMOD with other baselines to evaluate the performance quantitatively. For smoothness, these results are averaged over 20 times.

First, we compare the order request response rate for various capacity constraints of vehicles (NoRS is excluded as its performance is not affected by cache capacity). As shown in Fig. 9(a), AMOD is superior to other baselines in most cases. It achieves 13.5% improvement at maximum and 8.9% on average when the capacity constraint varies to 6. Notably, the difference between ADP-based methods (DRL, AMOD) and other methods grows larger with the increase of the capacity constraint. This is consistent with our motivation that PMA and HCRS are myopic, while ADP-based methods are derived from observed historical experiences and concentrate more on the anticipated reward. Thus, there exist upside potential for vehicles with larger capacity constraints if considering the long-term influences of order dispatching.

Similarly, the increasing number of vehicles also has a positive impact on the order request response rate. As shown in Fig. 9(b), all methods achieve their performance gains with the increase of the number of vehicles, especially for AMOD. This situation may occur because more order can be fulfilled thanks to widespread supply distribution, although profits per vehicle could be slightly less. As for state-of-the-art DRL, AMOD still achieves 5.3% and 6.7% gains for 1500 and 2500 vehicles, respectively. In particular, we observe that 2500 vehicles with capacity 4 achieve an order request response rate of over 95%, showing better quality of service.

In the following, we change the maximum pick-up delay to compare the order request response rate of different methods. As shown in Fig. 9(c), although AMOD performs weaker than HCRS, and DRL in case of the maximum pick-



(a) (b)

Fig. 10. The mismatch between demand and supply in spatiotemporal distribution. (a) PMA (b) AMOD.

up delay is 90s (probably because of strict delay constraints), it will outperform them with enough search for available action generation. In particular, AMOD outperforms PMA, HCRS, DRL, and NoRS with an improvement in the order request response rate of 14.4%, 10.1%, 3.8%, and 23.9%, respectively, when the maximum pick-up delay is 210s.

Finally, except for quantitive results, we provide an intuitive illustration by visualizing the algorithm's performance on city maps. To be more conspicuous, PMA (without considering vehicle repositioning) is chosen to compare with AMOD. The spatiotemporal mismatch between demand and supply during a peak hour is drawn in Fig. 10. Warmer colors signify a pronounced gap between supply and demand, indicating a need for more available drivers. Colder colors signify that current demands are adequately met with available vehicles still in the vicinity. A good order dispatching method should strive to maintain more balance between supply and demand, as evident in Fig. 10(b) by the presence of fewer warm areas. In this regard, AMOD does indeed demonstrate promising result.

Given these promising results, it can be observed that AMOD consistently has the best performance among all methods. Therefore, we demonstrate that AMOD is effective under different scenarios.

VII. Conclusion

This paper investigated the order dispatching with consideration of vehicle repositioning by exploiting ADP. We first formulated the optimization problem as a MDP, where the dispatching decision is determined by a series of agents under the time sequence model. Then, based on the generated available trips by a graph theory-based method, an ADP-based Multi-driver Order Dispatching method (AMOD) was proposed. In particular, AMOD reconstructed the Bellman update process around the post-decision states. As for non-linear function approximation, it converted the value function into a linear combination, and estimated the decomposed value function with neural network-based parameter approximation. In addition, vehicle repositioning was performed along with each batch dispatching to balance ride supply across geographic dimensions. Extensive simulations were conducted to verify the effectiveness of AMOD.

References

[1] Z. Qin, H. Zhu, and J. Ye, "Reinforcement learning for ridesharing: An extended survey," *Transp. Res. C, Emerg. Technol.*, vol. 144, pp. 1-28, Nov. 2022.

[2] S. C. K. Chau, S. Shen, and Y. Zhou, "Decentralized Ride-Sharing and Vehicle-Pooling Based on Fair Cost-Sharing Mechanisms," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 3, pp. 1936-1946, 2022.

[3] X. Tang et al., "A deep value-network based approach for multi-driver order dispatching," *in Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pp. 1780–1790, 2019.

[4] Y. Tong et al., "Combinatorial Optimization Meets Reinforcement Learning: Effective Taxi Order Dispatching at Large-Scale," *IEEE Trans. Knowl. Data Eng.*, Nov. 2021.

[5] T. G. Nguyen et al., "Federated Deep Reinforcement Learning for Traffic Monitoring in SDN-Based IoT Networks," *IEEE Trans. Cognitive Commun. Netw.*, vol. 7, no. 4, pp. 1048–1065, 2021.

[6] L. Xiao, C. Xie, M. Min, and W. Zhuang, "User-Centric View of Unmanned Aerial Vehicle Transmission Against Smart Attacks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 4, pp. 3420–3430, 2018.

[7] Y. Tong et al., "Online minimum matching in real-time spatial data: Experiments and analysis," *in Proc. VLDB Endowment*, vol. 9, no. 12, pp. 1053–1064, 2016.

[8] M. Lowalekar, P. Varakantham, and P. Jaillet, "ZAC: A zone path construction approach for effective real-time ridesharing," *in Proc. Int. Conf. on Autom. Plann. Sched.*, vol. 29, no. 1, pp. 528–538, Jul. 2020.

[9] L. Zhang et al., "A taxi order dispatch model based on combinatorial optimization," *in Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pp. 2151–2159, 2017.

[10] A. Heydari, "Revisiting Approximate Dynamic Programming and its Convergence," *IEEE Transactions on Cybernetics*, vol. 44, no. 12, pp. 2733-2743, Dec. 2014.

[11] S. Shah, M. Lowalekar, and P. Varakantham, "Neural approximate dynamic programming for on-demand ride-pooling," *in Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 1, pp. 507-515, 2019.

[12] P. Cheng, H. Xin, and L. Chen, "Utility-aware ridesharing on road networks," *in Proc. ACM Int. Conf. Manage. Data*, pp. 1197-1210, 2017.

[13] M. Asghari et al., "Price-aware real-time ride-sharing at scale: an auction-based approach," *in Proc. ACM Int. Conf. Adv. Geogr. Inf. Syst.*, pp. 1-10, 2016.

[14] R. Zhang, and M. Pavone, "Control of robotic mobility-on-demand systems: a queueing-theoretical perspective," *Int. J. Robot. Res.*, vol. 35, no. 1, pp. 186-203, 2016.

[15] Y. Duan, J. Wu, and H. Zheng, "A greedy approach for carpool scheduling optimisation in smart cities," *Int. J. Parall. Emerg. Distr. Syst.*, pp. 1-15, 2018.

[16] X. Bei, and S. Zhang, "Algorithms for trip-vehicle assignment in ridesharing," *in Proc. AAAI Conf. Artif. Intell.*, vol. 32, no. 1, pp. 3-9, 2018.

[17] H. To, C. Shahabi, and L. Kazemi, "A server-assigned spatial crowdsourcing framework," *ACM Trans. Spatial Algorithms Syst.*, vol. 1, no. 1, pp. 1-28, 2015.

[18] B. N. Greenwood, and S. Wattal, "Show me the way to go home: an empirical investigation of ride-sharing and alcohol related motor vehicle fatalities," *MIS Quart.*, vol. 41, no. 1, pp. 163-187, 2017.

[19] L. Zhang et al., "A taxi order dispatch model based on combinatorial optimization," *in Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pp. 2151-2159, 2017.

[20] Y. Chen et al., "PPtaxi: Nonstop package delivery via multi-hop ridesharing," *IEEE Trans. Mobile Comput.*, vol. 19, no. 11, pp. 2684-2698, 2020.

[21] S. Ma, Y. Zheng, and O. Wolfson, "Real-time city-scale taxi ridesharing," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 7, pp. 1782-1795, Jul. 2015.

[22] M. W. Ulmer, J. C. Goodson, D. C. Mattfeld, and M. Hennig, "Offline-Online Approximate Dynamic Programming for Dynamic Vehicle Routing with Stochastic Requests," *Transp. Sci.*, vol. 53, no. 1, pp. 185-202, 2019.

[23] L. Al-Kanj, J. Nascimento, and W. B. Powell, "Approximate dynamic programming for planning a ride-hailing system using autonomous fleets of electric vehicles," *Eur. J. Oper. Res.*, vol. 284, no. 3, pp. 1088-1106, 2020.

[24] X. Yu, and S. Shen, "An integrated decomposition and approximate dynamic programming approach for on-demand ride pooling," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 9, pp. 3811-3820, 2019.

[25] Z. Xu et al., "Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach," *in Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pp. 905-913, 2018.

[26] H. Chen et al., "Inbede: Integrating contextual bandit with td learning for joint pricing and dispatch of ride-hailing platforms," *in Proc. IEEE Int. Conf. Data Mining*, pp. 61-70, 2019.

[27] C. Li, D. Parker, and Q. Hao, "A Value-based Dynamic Learning Approach for Vehicle Dispatch in Ride-Sharing," *in Proc. IEEE/RSJ IROS Int. Conf. Intell. Robots Syst.*, pp. 11388-11395, 2022.

[28] M. Haliem et al., "A Distributed Model-Free Ride-Sharing Approach for Joint Matching, Pricing, and Dispatching using Deep Reinforcement Learning," *CSCS: Computer Science in Cars Symposium*, no. 5, pp. 1-10, 2020.

[29] A. Singh, A. O. Al-Abbasi, and V. Aggarwal, "A Distributed Model-Free Algorithm for Multi-Hop Ride-Sharing Using Deep Reinforcement Learning," *IEEE Trans. Intell. Trans. Syst.*, vol. 28, no. 7, pp. 8595-8605, 2022.

[30] G. Boeing, "OSMnx: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks," *Comput. Env. Urban Syst.*, vol. 65, no. 4, pp. 126-139, 2017.

[31] P. Santi et al., "Quantifying the benefits of vehicle pooling with shareability networks," *Proc. Nat. Acad. Sci.*, vol. 111, no. 37, pp. 13290-13294, 2014.

[32] G. C. Crişan, E. Nechita, and D. Simian, "On Randomness and Structure in Euclidean TSP Instances: A Study With Heuristic Methods," *IEEE Access*, vol. 9, no. 1, pp. 5312-5331, 2021.

[33] D. T. Pham, and D. Karaboga, "Intelligent Optimisation Techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks," *ISpringer Science & Business Media*, 2012.

[34] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment," *Proc. Nat. Acad. Sci.*, vol. 114, no. 3, pp. 462-467, 2017.

[35] W. B. Powell, "Approximate Dynamic Programming: Solving the Curses of Dimensionality," *Wiley Series in Probability and Statistics*, 2011.

[36] I. Hull, "Approximate dynamic programming with post-decision states as a solution method for dynamic economic models," *Journal of Economic Dynamics and Control*, vol. 55, pp. 57-70, 2015.

[37] R. Stuart, and A. Zimdars, "Q-decomposition for reinforcement learning agents," *in Proc. Int. Conf. Mach. Learn.*, pp. 656-663, 2003.

[38] S. Wang et al., "Deep Reinforcement Learning for Dynamic Multichannel Access in Wireless Networks," *IEEE Trans. Cognitive Commun. Netw.*, vol. 4, no. 2, pp. 257-265, 2018.

[39] Y. Xiao, L. Xiao, X. Lu, H. Zhang, S. Yu, and H. V. Poor, "Deep-Reinforcement-Learning-Based User Profile Perturbation for Privacy-Aware Recommendation," *IEEE Internet of Things J.*, vol. 8, no. 6, pp. 4560-4568, 2021.

[40] NYC Taxi & Limousine Commission. (2020). *New York City's Taxi Trip Data.* [Online]. Available: https://www.nyc.gov/site/tlc/about/ tlc-trip-record-data.page.

Kai Jiang (Student Member, IEEE) is currently pursuing the Ph. D. degree in cyberspace security at Wuhan University, China. His research interests include Deep Reinforcement Learning, Intelligent Transportation, Internet of Vehicles, and Mobile Edge Computing.

Yue Cao (Senior Member, IEEE) received the Ph.D. degree from the Institute for Communication Systems (ICS) formerly known as Centre for Communication Systems Research, University of Surrey, Guildford, U.K., in 2013. Further to his PhD study, he had conducted a Research Fellow with the University of Surrey, and academic faculty with Northumbria University, U.K., Lancaster University, U.K., and Beihang University, Beijing, China. He is currently a Professor with the School of Cyber Science and Engineering, Wuhan University, Wuhan, China. His research interests include intelligent transport systems, including E-Mobility, V2X, and edge computing.

Huan Zhou (Member, IEEE) received his Ph. D. degree from the Department of Control Science and Engineering at Zhejiang University. He was a visiting scholar at the Temple University from Nov. 2012 to May, 2013, and a CSC supported postdoc fellow at the University of British Columbia from Nov. 2016 to Nov. 2017. Currently, he is a Professor at the College of Computer, Northwestern Polytechnical University, Xi'an, China. He was a Lead Guest Editor of Pervasive and Mobile Computing, and TPC Chair of EAI BDTA 2020, Local Arrangement Chair of I-SPAN 2018, and TPC member of IEEE MASS, ICCCN, Globecom, ICC, WCSP, etc. He has published more than 70 research papers in some international journals and conferences, including IEEE JSAC, TPDS, TMC, TWC and so on. His research interests include Mobile Social Networks, VANETs, Opportunistic Mobile Networks, and mobile data offloading. He receives the Best Paper Award of I-SPAN 2014 and I-SPAN 2018.

Jie Wu (Fellow, IEEE) is the Chair and a Laura H. Carnell Professor in the Department of Computer and Information Sciences at Temple University, Philadelphia, PA, USA. Prior to joining Temple University, he was a Program Director at the National Science Foundation and a Distinguished Professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu has regularly published in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Services Computing, and the Journal of Parallel and Distributed Computing. Dr. Wu was general co-chair/chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, ACM MobiHoc 2014, IEEE ICPP 2016, and IEEE CNS 2016, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a CCF Distinguished Speaker and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.

Zhao Zhang (Member, IEEE) received the Ph.D. degree from Southwest Jiaotong University in 2013. He is currently an Assistant Professor with the School of Transportation Science and Engineering, Beihang University, Beijing, China. His research interests include traffic modeling and control, ITS, and evacuation optimization.

Zhi Liu (Senior Member, IEEE) received the PhD degree in informatics from National Institute of Informatics. He is currently an Associate Professor with the University of Electro-Communications. His research interest includes video network transmission and mobile edge computing. He is now an editorial board member of Springer wireless networks and IEEE Open Journal of the Computer Society.