Combinatorial Optimization Meets Reinforcement Learning: Effective Taxi Order Dispatching at Large-Scale

Yongxin Tong, *Member, IEEE,* Dingyuan Shi, Yi Xu, Weifeng Lv, Zhiwei (Tony) Qin, Xiaocheng Tang

Abstract—Ride hailing has become prevailing. Central in ride hailing platforms is taxi order dispatching which involves recommending a suitable driver for each order. Previous works use pure combinatorial optimization solutions for taxi dispatching, which suffer in practice due to complex dynamics of demand and supply and temporal dependency among dispatching decisions. Recent studies try to adopt data-driven method into combinatorial optimization hoping knowledge from history data would help overcome these challenges. Among these attempts, adoption of reinforcement learning shows great promise but current adoptions are a unidirectional integration which restricts the potential performance gains. In this work, we propose Learning To Dispatch(LTD), a systematic solution that allows synergic integration of reinforcement learning and combinatorial optimization for large-scale taxi order dispatching. We demonstrate the necessity of online learning and taxi scheduling for reinforcement learning to work in synergy with combinatorial optimization, and devise corresponding algorithms. We also devise many tricks for more efficient calculation of the bipartite matching. Experiments show our methods can improve 36.4% and 42.0% on utility and efficiency at most, respectively. Especially, it achieves state-of-the-art performance in terms of utility.

Index Terms—Reinforcement Learning, Bipartite Matching, Taxi Dispatching.

1 Introduction

Ride hailing has become a prevailing transport mode and tremendously improved the urban traffic capacity [1]. Central in ride hailing services is taxi order dispatching, where the ride hailing platform assigns taxi ride orders (*i.e.,* demand) to appropriate taxi drivers (*i.e.,* supply). Effective and efficient order dispatching at large scale is essential to both the overall utility (*e.g.,* total revenue) and the quality of service (*e.g.,* travel cost and waiting time) in urban ride hailing [2], [3], [4], [5].

Mathematically, order dispatching can be modeled as a bipartite matching problem. Specifically, drivers and orders are modeled as nodes on the two sides of bipartite graph and an edge between two nodes represents a potential assignment. Accordingly, finding a dispatch decision between drivers and orders is converted into calculating bipartite matching. The bipartite matching formulation naturally leads to solutions from a combinatorial optimization perspective [3], [4], [5], [6], [7]. Despite decades' of research, pure combinatorial optimization based strategies still fail to deliver the theoretically claimed performance in practice. Empirical studies show that many methods even cannot beat the naive greedy algorithm [4]. This is because combinatorial optimization solutions are often myopic and reply on strong assumptions, preventing them from handling the following challenges in real-world ride hailing.

- *Complex dynamics of demand and supply.* Most combinatorial optimization solutions oversimplify the spatial distribution of demand and supply *e.g.,* assuming the potential drivers upcoming following Zipf's Law [8] or some known independent identical distribution fitted on history data [9]. In practice, taxi demand can be highly irregular and volatile due to diverse factors such as rush hour, weather, events and so on. The supply can also be capricious in space due to the drivers' idle cruising, which is difficult to model and predict.
- *Temporal dependency among dispatching decisions.* Combinatorial optimization methods tend to assume independent decisions. Nevertheless, former dispatching decisions can affect later ones since the previously dispatched drivers are likely to appear near the destinations of their assigned orders later, which can change the supply distribution in subsequent dispatching time frames.

The availability of big data provides an opportunity to augment combinatorial optimization based solutions from a data-driven perspective [10], [11], [12], [13], [14]. Recent efforts have exploited data mining techniques [12], [13], [14] to predict demand and supply rather than reply on simplified assumptions of their distributions for order dispatching. However, since these methods overlook the impact of prior dispatching on the supply distribution in subsequent dispatches, their predictions tend to deviate from actual ones, thus limiting their effectiveness. A few pioneer studies [10], [11] propose to jointly optimize a batch of dispatching decisions as a sequential decision problem, which can be solved by reinforcement learning [15]. In the reinforcement

- *Y. Tong, D. Shi, Y. Xu, and W. Lv are with the State Key Laboratory of Software Development Environment, School of Computer Science and Engineering, Beihang University, PR China. E-mail: {yxtong, chnsdy, xuy, lwf}@buaa.edu.cn.*
- *Z. Qin and X. Tang are with Didi AI Labs, E-mail: {qinzhiwei, xiaocheng-tang}@didiglobal.com.*
- *Yi Xu is the corresponding author in this paper.*

learning setting, the platform learns the best dispatching strategies for a given time period from big historical data of supply, demand and dispatching decisions. This setting not only eliminates pre-assumptions on supply and demand, but also explicitly account for the dependencies among dispatches. However, as early explorations to integrate combinatorial optimization with reinforcement learning, existing proposals [10], [11] lack systematic designs to optimize the interplay between these two paradigms. For example, both [10] and [11] only apply reinforcement learning to augment combinatorial optimization. Such a *unidirectional* integration restricts the potential performance gains. In addition, the data-driven nature of reinforcement learning may bring adverse effect to combinatorial optimization, which can even jeopardize the performance.

In this work, we propose Learning To Dispatch (LTD), a systematic solution that allows synergic integration of reinforcement learning and combinatorial optimization for large-scale taxi order dispatching. For effective augmentation of reinforcement learning to combinatorial optimization, we adopt an online learning manner to adapt the learned dispatching strategies to the current situations. To avoid the cold start problem in reinforcement learning, we devise a combinatorial remedy *i.e.*, driver scheduling. We also propose optimizations to optimize the efficiency to apply reinforcement learning in large-scale settings. Extensive evaluations show that our *bi-directional* integration outperforms among all baselines by up to 36.4% and 42.0% on utility and efficiency. We also outperform the state-of-the-art method [11] by up to 28.7% on utility. Our main contributions are summarized as follows.

- We study the problems and solutions to integrate reinforcement learning with combinatorial optimization for taxi order dispatching. We demonstrate the necessity of online learning and taxi scheduling for reinforcement learning to work in synergy with combinatorial optimization for effective taxi dispatching. To the best of our knowledge, we are the first to explore the best practices to combine these two methodologies for ride hailing services.
- We devise many tricks for more efficient calculation of the bipartite matching. By adopting shared value function and approximation, we largely reduce the size of value function and help better exploration. We also use breadth first search to split the bipartite graph into several parts for a higher execution efficiency. These accelerations make our algorithm suitable for large-scale and high-frequency dispatching operations.
- We conduct experiments on a simulator developed by a major ride hailing platform based on real history data. The experiment results indicate that our newly proposed algorithm outperforms the baselines on utility and efficiency by up to 36.4% and 42.0%, respectively. A preliminary version of this work won the championship of KDD Cup 2020 reinforcement learning track order dispatching task[1]. Our performance of utility is the state-of-the-art.

1. https://www.kdd.org/kdd2020/files/KDD_Cup_Day_Program.pdf

The rest of this paper are organized as follows. We provide a review of related work in Sec. 2. In Sec. 3 we define our problem. We explain the reinforcement learning model and our algorithms in Sec. 4 and Sec. 5, respectively. In Sec. 6 we present the evaluations and conclude in Sec. 7.

2 Related Work

Our work studies order dispatching problem, the central issue in ride hailing applications.

Order dispatching is a typical task assignment problem [16] in spatial crowdsourcing [17], where the tasks and workers are orders and drivers respectively. The problem is commonly modeled by online bipartite graph matching [18], [19] with different objectives like maximizing the total revenue [2], [3], [18], minimizing the total waiting time of passengers [4], etc. This work mainly focuses on the maximum matching. We divide existing solutions into two categories: *combinatorial optimization* based approach and *reinforcement learning* based approach.

Combinatorial Optimization based Approach. In the early stage of ride hailing, the most widely adopted approach is the *Greedy* algorithm, which assigns an order to the closest driver [20], [21]. With the development of mobile Internet and sharing economy, ride hailing platforms such as Uber have emerged. These shared-mobility platforms are capable of collecting massive historical spatiotemporal data in order dispatching. Extensive studies [3], [4], [5], [6], [7], [18], [22], [23], [24] focus on designing wiser optimization algorithms with the help of historical data, considering that mining patterns from historical data would make dispatching more comprehensive. In [6] and [7], the order dispatching problem is modeled as a bipartite matching or max flow problem [25] and classical combinatorial optimization methods like the Hungarian algorithm [26] is applied. However, these works assume the bipartite graph is static, which is inconsistent with the real ride hailing applications. [18] is the first work to model the problem by two-sided online bipartite matching, where both orders and drivers appear dynamically. Some other works [5], [22] model order dispatching as an Integer Linear Programming (ILP) problem [27], where the order arrival distribution is considered. The drawback of ILP model is its large time consumption and [3] proposes a hill climbing method to improve the efficiency and [24] propose index structure for acceleration and adopt a carpooling order dispatching manner.

There is rich literature on dispatching solutions with theoretical performance guarantees on the *worst* cases. However, [4] systematically compares these methods and finds that *Greedy* has good performance in real scenarios as the worst case happens rarely. Therefore some work [18] investigates the average (or expected) theoretical performance. These studies indicate that instead of stressing too much on theoretical guarantees, an algorithm with good empirical performances yet lacking guarantees would be more practical. Following this philosophy, recent studies manage to adopt learning-based manners into the combinatorial optimization, especially reinforcement learning, as we will explain next.

Reinforcement Learning based Approach. Reinforcement learning shows great power on solving problems with sequential decisions and temporal dependencies[2]. In reinforcement learning, an agent interacts with the environment repeatedly. In each round, the agent takes an action according to some policy, gets reward from the environment and transits to a new state. The goal is to design a good policy that can maximize the accumulated rewards over all rounds [28]. Such a formulation fits the problem setting of order dispatching. In order dispatching, the dispatching decisions are made sequentially with certain reward (revenue of orders) as feedback. A decision will also affect the environment by changing the distribution of drivers, which makes it a typical sequential decision making problem. Reinforcement learning solves the problem by learning how to make decisions from massive historical data.

A few works have investigated reinforcement learning in order dispatching. [10] models the platform as the agent, the global dispatching decisions as the actions and solves the problem by Q-learning [29] with tabular Q-values. This work also combines learning with optimization but the learning process is separated in an offline manner. Our approach is more adaptive by integrating both online learning and scheduling. More recent studies [11], [30] further adopt deep neural networks to fit more complicated state value functions. Instead of directly making dispatching decisions by reinforcement learning, [2] uses reinforcement learning to dynamically changing the batch sizes in a batch-based dispatching framework while the decisions remain static in each batch. Unlike prior works adopt reinforcement learning in a relatively independent manner, we combine the reinforcement learning and combinatorial optimization more tightly making them work in synergy, which largely hoists the performance.

Some other works apply multi-agent reinforcement learning (MARL) [31] to order dispatching. For example, [32] models each area as an agent and optimizes the dispatching performance by reshaping the areas. In [33], [34], each driver is modeled as an agent, and dispatching decisions are made in an decentralized way. However, they ignored that drivers may compete with each other for personal profits and the global profits will be damaged. To address this issue, we consider centralized synchronization by the platform in a multi-agent model.

3 Problem Definition

This section formally defines the general order dispatching problem in ride hailing following [2], [4], [10], [11], [18].

Definition 1 (Batch). *Assume time is evenly split by a sequence of batches $\langle 1, 2, \cdots, T\rangle$. A batch (*i.e.*, time step) t is an element in the batch sequence, which is also known as a time index.*

The batch based time model is common in real ride hailing platforms [10], [11]. A typical batch size is 2 seconds.

Definition 2 (Order). *An order (*i.e.*, a taxi hailing request) r is denoted by a tuple $\langle o_r, d_r, p_r, \tau_r\rangle$, where o_r, d_r, p_r and τ_r represent the origin, destination, estimated price and estimated duration time of order r, respectively.*

2. https://deepmind.com/research/case-studies/alphago-the-story-so-far

In real ride hailing applications, the origin and destination are input by the passengers in the form of latitude and longitude. We make no assumption about their distribution. The platform will estimate the price and duration time based on the trip mileage and other factors (*e.g.*, traffic congestion) after receiving the passengers' requests. The concrete estimation methods are out of our scope. We assume the duration time is a multiple of batch size due to the small batch size.

Definition 3 (Driver). *A taxi driver w is denoted as a tuple $\langle l_w^{(t)}, \xi_w^{(t)}\rangle$, where $l_w^{(t)}$ and $\xi_w^{(t)}$ represent the location and status of driver w in batch t, respectively.*

The status $\xi_w^{(t)}$ can be either "idle" or "occupied", indicating whether the platform can assign orders to the driver. We assume the status stays the same within one batch. An occupied driver will appear as "idle" at the destination of his assigned order after finishing the trip. This is known as the "reusable" driver setting [22], which is more realistic than the independent identical distribution setting [35], where every idle driver is taken as newly and independently coming nodes following the same distribution. Note that the reusable setting captures the *temporal dependency property*, *i.e.*, current dispatching decisions will influence the distribution of drivers in the future.

Definition 4 (Dispatching Candidates). *The dispatching candidates in batch t is represented as a bipartite graph $G^{(t)} = \langle R^{(t)}, W^{(t)}, E^{(t)}\rangle$ where $R^{(t)}$ is the set of orders appearing in batch t, $W^{(t)}$ is the set of idle drivers in batch t and $E^{(t)}$ is the set of edges* i.e., *eligible order-driver pairs. Each edge e in $E^{(t)}$ is denoted by $\langle r, w, x_{r,w}\rangle$, where r, w and $x_{r,w}$ represent the order, driver and edge weight, respectively.*

The graph is often incomplete as an edge will be pruned if the distance between an order's origin and a driver's location exceeds a threshold θ like 3km. The weight $x_{r,w}$ is initially set to the price p_r of order r.

Definition 5 (Cancel Probability). *The cancel probability $\lambda_{r,w}$ refers to the probability of canceling an order r when it is assigned to driver w.*

An order can be canceled if the assigned driver is too faraway. The cancel probability $\lambda_{r,w}$ increases monotonically with the order-driver distance. It is a hidden variable and we can only know whether an order will be canceled after the dispatching decision is made. The cancel probability makes our problem setting more aligned with real applications. That is, the solution has to tolerate some impatient passengers that may cancel the order with too long waiting time. In contrast, some works [2], [5] optimize the total revenue by assigning orders to unreachable drivers and ignore the possibility of canceling.

Definition 6 (Dispatching Algorithm). *In batch t, a dispatching algorithm $\mathscr{A}$ takes the dispatching candidates,* i.e., *the bipartite graph $G^{(t)}$ as input, and outputs the matching allocation $M^{(t)}$. $M^{(t)}$ is a subset of edges (*i.e.*, order-driver pairs) in $G^{(t)}$ representing the order dispatching decisions.*

In each batch, the algorithm is executed by the ride hailing platform. The dispatching algorithm is not necessarily stationary, since historical data may affect future decisions. We still use $\mathscr{A}$ instead of $\mathscr{A}^{(t)}$ for simplification.

Definition 7 (Utility Score)**.** *The utility score of a matching allocation is defined by*

$$U(M^{(t)}) = \sum_{(r,w)\in M^{(t)}} x_{r,w} \cdot (1 - \mathbb{I}_{\lambda_{r,w}})$$

where $\mathbb{I}_{\lambda_{r,w}} \sim Bernoulli(\lambda_{r,w})$.

Definition 8 (Order Dispatching Problem)**.** *Given a batch sequence* $\langle 1, 2, \cdots, T\rangle$ *and a sequence of dispatching candidates* $\langle G^{(t)}\rangle$*, the order dispatching problem is to decide a dispatching algorithm* $\mathscr{A}$ *in each batch to maximize the total utility score,* i.e.,

$$\max \sum_{t=1}^{T} U(M^{(t)}) \tag{1}$$

We call the order r is *responded* if the algorithm assigns a driver w to it. We call the order r is *completed* if the assignment is not canceled (*i.e.*, $\mathbb{I}_{\lambda_{r,w}} = 0$). Table 1 summarizes the frequently used symbols in this paper.

Remark. Our problem is general and aligned with real scenarios due to the following reasons.

- We make assumptions on neither drivers (distributions of their locations) nor orders (distributions of their origins, destinations or arrival time).
- We consider the cancel probability. It encourages the dispatching algorithm to assign orders to nearby drivers, which can increase passengers' satisfaction.

TABLE 1
Summary of symbols.

Notation	Description
T	total number of batches
t	a batch
$R^{(t)}$	set of merged order in batch t
$W^{(t)}$	set of available drivers in batch t
$l_w^{(t)}, \xi_w^{(t)}$	location and status of driver w in batch t
o_r, d_r	origin and destination of order r
p_r, τ_r	estimated price and duration time of order r
$G^{(t)}$	dispatching candidates in batch t
$E^{(t)}$	edges in $G^{(t)}$
$M^{(t)}$	matching allocation on $G^{(t)}$
$\lambda_{r,w}$	cancel probability of assigning r to w
$x_{r,w}$	weight of edge (r, w)
θ	threshold of order-driver distance
U	utility score
$\mathscr{A}$	dispatching algorithm
$\mathscr{G}$	taxi scheduling algorithm
$\mathscr{S}^{(t)}$	scheduling scheme in batch t
$I^{(t)}$	idle drivers in batch t
$AS_w^{(t)}$	action space of agent w in batch t
$S_w^{(t)}$	state of driver w in batch t
δ	space discretion function
δ_S	discretion function based on square grid
δ_H	discretion function based on hexagon grid
γ	discount factor
V_S	value function of square state
V_H	value function of hexagon state
$V(l)$	final value of location l
$\mu_{r,w}$	evaluation term of edge (r, w)
$x'_{r,w}$	new edge weight after redefinition
$P^{(t)}$	sub-bipartite of $G^{(t)}$
b	normalization factor for value function
DIR	smooth direction set
C, k	parameter for cancel probability prediction
$dist(r, w)$	distance between order r and driver w

4 Reinforcement Learning Model

Reinforcement learning (resp. Multi-agent reinforcement learning) tries to maximize the accumulate rewards as agent (resp. agents) interacts with the environment. In our problem setting the dispatch decision of former rounds will influence the driver distribution, thus posing an impact on latter dispatch. Our problem also includes thousands of drivers. So it is natural to view each driver as an agent and adopt multi-agent reinforcement learning.

A multi-agent reinforcement learning model consists of environment, agents, state, reward, value function and discount factor. The paradigm of multi-agent reinforcement learning involves agents who take actions based on policies derived from value functions. Upon agents taking action, either independently or collaboratively, the environment will feedback the states and rewards to agents. Agents will then update their value functions based on environment's feedback for better policy. The discount factor is a weight guiding the agent pays more attention on immediate reward rather than future expectation. We will expound details of each composition below.

Environment. The environment will build the bipartite graph, simulate transition of agents' states, and feedback the rewards and states back to agents. Note that the environment needs to reflect the dynamics of traffic situation, which poses a large impact on the order estimation time of arrival, order price and so on. The traffic situation range will reflect on the history data and the build of simulation environment will be introduced in Sec. 6.

Agent and action space. We adopt multi-agent reinforcement learning model and model each driver as an agent instead of modeling the platform as the only agent to largely reduce the action space of agent. If we model only one agent, then the action of the agent is the results of the bipartite matching, which leads to a large action space. The multi-agent formulation, however, reduces the action space of each agent, with only several candidate assignments. Specifically, the action space $AS_w^{(t)}$ of agent w in batch t is getting some order r from candidate assignment of bipartite or remain idle next batch, *i.e.*, $AS_w^{(t)} = \{A_r, |\langle r, w\rangle \in G^{(t)}\} \cup \{idle\}$. Different from traditional multi-agent reinforcement learning, one order can only be assigned to one agent and vice versa, so conflicts may occur if agents take actions directly. To solve this, each agent only proposes the action he or she wants to take, and the matching component will make the final decision in a global view, which will be explained in Sec. 5.3.

States and transitions. Since order dispatching has long been categorized as a spatial temporal problem, it is natural to set the state $S_w^{(t)}$ of agent w in batch t contains both spatial and temporal information, *i.e.*, $S_w^{(t)} = \langle l_w^{(t)}, t\rangle$. However, this will cause the curse of dimensionality. For example, if we cut the whole day into 10-minute length segment and split the city area into square grids whose side length are 500m, there will be 144 time segment and 1600 grids (assuming the city covers a square area of 20km side length), hence the total spatial temporal space number would be $144 \times 1600 = 230400$. This is much greater than an agent exploration ability (each driver can only serve 20~30 orders a day and the explored states will be less than 10^2). Though a neural network may overcome the curse of dimensionality, in large-scale ride hailing, however, forward propagation in large neural network would be too time-consuming. So we still adopt tabular solution (*i.e.*, state number is finite and the value function can be expressed as a table).

Noting that most orders do not last long (within an hour) and it is reasonable to use the current evaluation to approximate the future one, we reduce the state number by using only the spatial information. That is, location $l_w^{(t)}$of agent w in batch t, and the state $S_w^{(t)}$ can be expressed as:

$$S_w^{(t)} = \delta(l_w^{(t)}) \tag{2}$$

where δ is the discretion function to map location to state. We discretize the agent location by meshing (*i.e.*, split the whole geographical space into small adjacent grids). We adopt two types of grids: square and hexagon. Their shape boundaries have different number and directions, making the value function more adaptive to the urban layout (see Sec. 5.2.1). The discretion function using two types of grids are denoted as δ_S and δ_H, respectively. Then the state becomes a tuple $S_w^{(t)} = \langle \delta_S(l_w^{(t)}), \delta_H(l_w^{(t)}) \rangle$

The state transition depends on the order's origin and destination. If an agent w gets an order r from o_r to d_r, then its state transits from $\langle \delta_S(o_r), \delta_H(o_r) \rangle$ to $\langle \delta_S(d_r), \delta_H(d_r) \rangle$.

Discount factor. The discount factor γ is a hyper parameter. It can be viewed as a weight, indicating the confidence for the evaluation of the future. As mentioned before, we use the current evaluation to approximate the future one, so an order with long trip duration will weaken the confidence on the approximation. To account for such variation in the confidence, we adopt trip duration into the discount factor. Specifically, the discount factor will decay exponentially to the order duration τ_r (*i.e.*, γ^{τ_r}).

Reward. Reward R is the immediate gain of each assignment. Our goal is to maximize the total revenue, so we set the reward to the order price. Since we need to make the matching decision *before* the order finish, the price we use is a predicted price, which may slightly differ from the actual one. The predicted price is given by the platform as input and the its calculation is out of scope of this paper.

Value function. In reinforcement learning, there are two types of value functions. One is state value function, denoted as V. It maps the state $S_w^{(t)}$ to the expected accumulated reward from this state, *i.e.*, $V(S_w^{(t)}) = \mathbb{E}[\sum_t R^{(t)} | S_w^{(t)}]$. The other is action-state value function, denoted as q. It maps the state and action $\langle S^{(t)}, A \rangle$ to the expected accumulated reward from this state $S_w^{(t)}$ taking action A, *i.e.*, $q(S^{(t)}, A) = \mathbb{E}[\sum_t R^{(t)} | S_w^{(t)}, A]$. In a model-free situation, where little is known about the environment, we usually derive policy π based on action-state value function q. In ride hailing, however, the action space AS is changing in different batches, causing a tabular q infeasible. Nevertheless, for high efficiency, the tabular method is preferred than neural networks. Hence we derive our action decision based on V rather than q. In our method, the policy π is derived by Hungarian based matching (see Sec. 5.3) in a holistic manner over all agents and it is implicit, so for simplicity, we omit the notion π. Since we have set the state of agent to the location, the state value function V_w of agent w can be defined as follows:

$$V_w(S_w^{(t)}) = \mathbb{E}[\sum_t R^{(t)} | S_w^{(t)}] \tag{3}$$

To reduce the value function space for better exploration, we apply the following strategies.

- *Share value function among agents.* To ensure the value function will be fully trained, all agents share the same value function. The intuition is that the matching algorithm will treat all drivers equally, so each driver will have a similar evaluation on the same location. This trick makes all agents work collaboratively. For instance, a driver is working east of a city, and now comes an order heading for the west. Since the driver has not been to the west yet, without the shared value function, he cannot make an accurate evaluation of the destination. However, drivers working in the west can make an evaluation by their value functions. By value function sharing, the driver in the east can utilize this information to evaluate the destinations he/she has never been. We omit the subscript w to denote the shared value function. Thus the value function is defined as follows.

$$V(S^{(t)}) = \mathbb{E}[\sum_t R^{(t)} | S^{(t)}] \tag{4}$$

- *Adopt two value functions for different discretion states.* As mentioned above, state $S_w^{(t)}$ is a tuple derived by two functions δ_S and δ_H. Direct mapping from a tuple to the value introduces a large value function space to explore. Thus, we adopt two value functions V_S and V_H to map $\delta_S(l_w^{(t)})$ and $\delta_H(l_w^{(t)})$ separately.

$$V_S(\delta_S(l_w^{(t)})) = \mathbb{E}[\sum_t R^{(t)} | \delta_S(l_w^{(t)})] \tag{5}$$

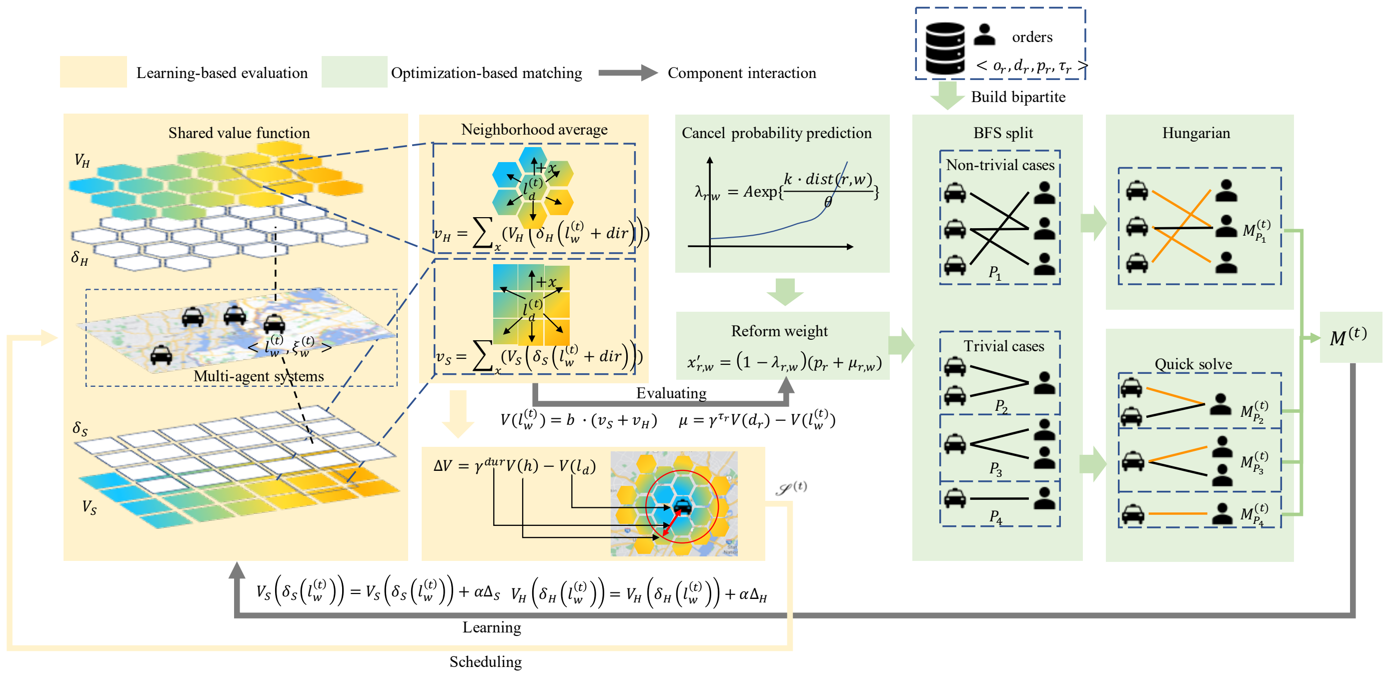
$$V_H(\delta_H(l_w^{(t)})) = \mathbb{E}[\sum_t R^{(t)} | \delta_H(l_w^{(t)})] \tag{6}$$

  To derive the final value functions from V_S and V_H, we design a neighborhood averaging method to better adapt to different urban layouts (see Sec. 5.2.1).

Our multi-agent reinforcement learning model characterizes the essence of our problem (such as temporal dependency). As next, we describe how to integrate this model into our dispatching algorithm.

5 Method

In this section, we present our <u>L</u>earning <u>T</u>o <u>D</u>ispatch (LTD) framework, which consists of two components: *Learning-based Evaluation* and *Optimization-based Matching*.



Learning-based evaluation
Optimization-based matching
Component interaction
orders
$< o_r, d_r, p_r, \tau_r >$
Build bipartite
Shared value function
V_H
δ_H
Multi-agent systems
$< l_w^{(t)}, \xi_w^{(t)} >$
δ_S
V_S
Neighborhood average
$v_H = \sum_x (V_H(\delta_H(l_w^{(t)} + dir)))$
$v_S = \sum_x (V_S(\delta_S(l_w^{(t)} + dir)))$
Cancel probability prediction
$\lambda_{r,w} = A\exp\{\frac{k \cdot dist(r,w)}{\theta}\}$
Reform weight
$x'_{r,w} = (1 - \lambda_{r,w})(p_r + \mu_{r,w})$
Evaluating
$V(l_w^{(t)}) = b \cdot (v_S + v_H)$
$\mu = \gamma^{\tau_r} V(d_r) - V(l_w^{(t)})$
$\Delta V = \gamma^{dur} V(h) - V(l_d)$
$\mathscr{S}^{(t)}$
BFS split
Non-trivial cases
P_1
Trivial cases
P_2
P_3
P_4
Hungarian
$M_{P_1}^{(t)}$
Quick solve
$M_{P_2}^{(t)}$
$M_{P_3}^{(t)}$
$M_{P_4}^{(t)}$
$M^{(t)}$
$V_S(\delta_S(l_w^{(t)})) = V_S(\delta_S(l_w^{(t)})) + \alpha\Delta_S$
$V_H(\delta_H(l_w^{(t)})) = V_H(\delta_H(l_w^{(t)})) + \alpha\Delta_H$
Learning
Scheduling

Fig. 1. The LTD framework. yellow and green boxes indicate evaluation and matching component, respectively. grey arrows indicate interactions between two components.

5.1 Framework Overview

The framework of LTD is illustrated in Fig. 1. It consists of two components: *Learning-based Evaluation* and *Optimization-based Matching*. The two components are performed iteratively repeatedly every batch. *Learning-based Evaluation* is built on the model in Sec. 4 which aims to evaluate the value of each candidate assignment based on previous decisions and corresponding rewards. *Optimization-based Matching* makes dispatching decisions by an optimized Hungarian algorithm, taking the evaluated values as part of weights on the bipartite graph. Afterwards, the dispatching decisions will feed back to the evaluation component for updating values. The two components will run in a circle with the following interactions:

- *From evaluation to matching*. The matching component takes the value function produced from the evaluation component as input, and reformulate the bipartite graph based on the values.
- *From matching to evaluation*. After the matching component finds the matching allocations, the decisions will be sent to the evaluation component. The value function will be updated based on the decisions.

Alg. (1) illustrates the entire process. We explain the two components in detail as below.

```
Algorithm 1: LTD
    Input: The bipartite graph
        $G^{(t)} =< R^{(t)}, W^{(t)}, E^{(t)} >$, idle drivers $I^{(t)}$
    Output: Matching $M^{(t)}$, scheduling scheme $\mathscr{S}^{(t)}$
1   if $t = 1$ then
2       $V_S \leftarrow 0$
3       $V_H \leftarrow 0$
4   end
5   if $t$ mod SchedulingInterval $= 0$ then
6       $\mathscr{S}^{(t)} \leftarrow \mathscr{G}(I^{(t)})$
7   end
8   else
9       $\mathscr{S}^{(t)} \leftarrow \emptyset$
10  end
11  calculate $V$ by Eq. (7)
12  calculate new edge weights by Eq. (12)
13  $M^{(t)} \leftarrow \mathscr{A}(G^{(t)})$
14  for $\langle r, w \rangle \in M^{(t)}$ do
15      update $V_S$ by Eq. (8)
16      update $V_H$ by Eq. (9)
17  end
18  return $M^{(t)}, \mathscr{S}^{(t)}$
```

5.2 Learning-based Evaluation

This component has two parts: the first explains how to infer the values given specific states, and the second reveals how to learn the values by temporal difference (TD) learning.

5.2.1 Inferring the Values

We use two value functions, *i.e.*, square value function V_S and hexagon value function V_H to represent the state values, as mentioned in Sec. 4.

To enrich the features, we apply tile coding [28], and the value function can be defined as

$$V \leftarrow b \cdot (\sum_{dir \in DIR} (V(\delta_S(l + dir)) + V(\delta_H(l + dir))) \quad (7)$$

where DIR is the tiling directions (shapes) and b is a normalization factor. We omit the superscript (t) and subscript w for simplification. The tile coding makes the value function more adaptive to urban layouts. Square boundaries

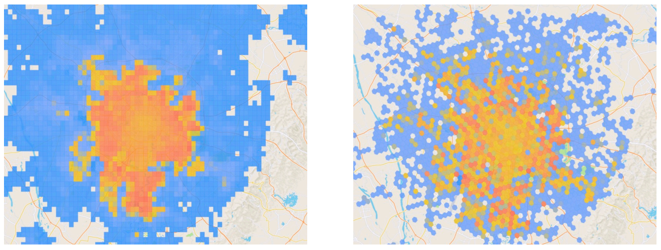

(a) (b)

Fig. 2. Hexagon and square value functions V_S and V_H at 9:00am. Warmer colors indicate higher values.

are parallel to longitude and latitude lines, showing regional patterns such as busy areas (see Fig. 2(a)), while hexagon boundaries have more directions and show a radial pattern aligned with trunk roads (see Fig. 2(b)). For simplification, we use $V(l)$ to represent the entire value function in Eq. (7).

5.2.2 Learning the Values

We learn the state value function V by temporal difference (TD) learning. TD learning [15] is a classic learning method in reinforcement learning. It is a model-free method, since it estimates the value function without any information or assumptions about the environment. Specifically, it learns the value function by synthesizing the immediate rewards and bootstrap approximation to future rewards. The immediate rewards may contain several steps, which derives different TD method, such as TD(0) (immediate rewards within one step), TD(λ) (immediate rewards within steps taking λ proportion of the total steps in one episode.) We choose *online TD(0)* for the following reasons:

- *Online v.s. Offline.* Some studies [10], [11] learn the value function offline with massive historical data. Yet offline learning has shortcomings. The value function V is strongly correlated with the matching algorithm. There is a gap between the matching rules behind the historical data and the online matching decisions following V, which may cause inconsistency and thus huge errors. In contrast, online learning ensures consistent decisions in both learning and matching. Instead, online updating of value function can stay adaptive to the test environment.
- *TD(0) v.s. TD(λ).* In TD(λ) or Monte Carlo method, there is a delay between the learning and the real-time decision making steps. In other words, the values are updated with several further steps of decisions. However, in ride hailing, the duration of each order varies dramatically. Therefore, different agents may transit to their next states at different time steps. The error will accumulate with steps and thus we choose only one step look ahead *i.e.*, TD(0).

According to the online TD(0) learning algorithm, if driver w takes order r in batch t, the updating of value functions V_S and V_H will be:

$$V_S(\delta_S(l_w^{(t)})) \leftarrow V_S(\delta_S(l_w^{(t)})) + \alpha\Delta_S \tag{8}$$

$$V_H(\delta_H(l_w^{(t)})) \leftarrow V_H(\delta_H(l_w^{(t)})) + \alpha\Delta_H \tag{9}$$

where α is the learning rate, $\Delta_S = p_r + \gamma^{\tau_r} V_S(\delta_S(d_r) - V_S(\delta_S(l_w^{(t)}))$ and $\Delta_H = p_r + \gamma^{\tau_r} V_H(\delta_H(d_r) - V_H(\delta_H(l_w^{(t)})))$.

Algorithm 2: Scheduling algorithm $\mathscr{G}$

```
Input: Idle drivers $I^{(t)}$
Output: scheduling scheme $\mathscr{S}^{(t)}$
1 $\mathscr{S}^{(t)} \leftarrow \emptyset$
2 for $w \in I^{(t)}$ do
3     find $h$ that can maximize $\Delta V$ in Eq. (10)
4     $\mathscr{S}^{(t)} \leftarrow \mathscr{S}^{(t)} \bigcup \{\langle w, h \rangle\}$
5 end
6 return $\mathscr{S}^{(t)}$
```

5.2.3 Cold start correction

The value $V(l)$ indicates the expected accumulated rewards from location l since now. With this information, we can actively schedule idle drivers to areas for a better distribution. We can evaluate the value increment as Eq. (10).

$$\Delta V = \gamma^{dur} V(h) - V(l_w) \tag{10}$$

where dur is the duration of driver w from location l_w to area h, whose candidate set is the same as hexagon tiles. Note that directing a large number of drivers to one area may notably change the value function and thus impair its accuracy. However, since the drivers can be assigned with orders during scheduling, few drivers will reach the destination finally. Thus little impact is imposed on the value function. Alg. (2) shows the scheduling algorithm. This algorithm is executed in a relatively low frequency(*e.g.*, for 150 batches).

5.3 Optimization-based Matching

The matching component is designed to make dispatching decisions based on the learned values. There are two challenges in designing the matching algorithm.

- *Lack of a holistic view.* There are problems if agents greedily make decisions by themselves. *(i)* There may be conflicting dispatching decisions. This may break the capacity constraint, *i.e.*, each order can only be assigned to one driver. *(ii)* Making decisions on their own benefit may trap in a local optimum.
- *Need for real-time response.* Large-scale ride hailing demands fast response, *i.e.*, high matching efficiency.

To solve the first challenge, we combine value learning with the Hungarian algorithm. For the second, we devise acceleration techniques to improve the efficiency.

5.3.1 Hungarian Matching with Values

We first reformulate the edge weights as:

$$x'_{r,w} \leftarrow (1 - \lambda_{r,w}) \cdot (x_{r,w} + \mu_{r,w}) \tag{11}$$

where $\lambda_{r,w}$ is the cancel probability and $\mu_{r,w}$ is the TD error with the following definition:

$$\mu_{r,w} = \gamma^{\tau_r} V(d_r) - V(l_w) \tag{12}$$

Intuitively, it indicates the expected accumulated reward gain if the driver moves from current location l_w to d_r.

We use an exponential distribution to estimate the cancel probability $\lambda_{r,w}$ with historical data.

$$\lambda_{r,w} = C \cdot \exp\{\frac{k \cdot dist(r,w)}{\theta}\} \quad (13)$$

where $dist(r,w)$ is the order-driver distance, θ is the threshold, C and k are parameters of the exponential distribution.

Compared with the old edge weight $x_{r,w}$, *i.e.*, the myopic revenue of the current order r, the new edge weight $x'_{r,w}$ is more farsighted, as it considers how much expected total revenue the agent (*i.e.*, the driver w) will obtain if assigned to order r. For example, if the destination of r is in downtown, which means the driver may have more opportunities to take orders with high revenue, the edge weight will be larger and vice versa.

However, if each agent w takes action greedily according to all the candidate orders to itself, *i.e.*, it chooses the order $r^* = \arg\max_r x_{r,w}$, there will be conflicts as the same order may be assigned to multiple drivers. Hence we enforce all the agents to take actions in a global view by solving the maximum bipartite matching problem with the Hungarian algorithm [26]. The time complexity is $O(N^4)$ where $N = \max\{|W^{(t)}|, |R^{(t)}|\}$.

5.3.2 Matching Acceleration Techniques

In large-scale ride hailing, the matching algorithm is executed frequently. Especially in rush hours when the numbers of orders and drivers are large, the time complexity of $O(N^4)$ can be intolerable. Meanwhile,the bipartite graph in our case is highly unbalanced. Usually the number of orders is far more than that of drivers. Thus, we adopt the slack optimization [36] in the Hungarian matching algorithm. It models the assignment problem as a max-flow problem. To avoid explicit construction of network flow graph, the optimized algorithm adds maintain tags on each nodes (called slack array) to record the current flow.

Furthermore, note that in real geometrical space the bipartite graph can be sparse as there is no edge between distant drivers and orders. So we use breadth first search (BFS) to split the graph into multiple sub-bipartite graphs and solve the multiple sub-bipartite matching problem independently. There are also many sub-bipartite graphs that contain only one driver or one order, and we use a naive greedy decision instead of the Hungarian algorithm for such special cases for further accelaration.

The details of the optimized matching algorithm are illustrated in Alg. (3). In Line 1-4, we use Eq. (11) to reformulate the weights based on value function calculated by the evaluation component. Then in Line 5 we use BFS to split the bipartite graph. In Line 7-20, the algorithm calculates the matching allocations in each sub-bipartite graph iteratively. If the graph contains only one driver (order), we directly choose the order (driver) with the largest edge weight, as shown in Line 8-11 (12-15). Otherwise we calculate the matching allocation using the Hungarian algorithm with slack optimization (Line 16-18).

5.3.3 Complexity Analysis

We analyze the time complexity of Alg. (3) as below. With the slack optimization, the time complexity of Hungarian algorithm is reduced from $O(N^4)$ to $O(M^2N)$, where $N = \max\{|R^{(t)}|, |W^{(t)}|\}$ and $M = \min\{|R^{(t)}|, |W^{(t)}|\}$. With the optimization of bipartite graph splitting, the time complexity becomes $O(\sum_{P^{(t)}} M^2_{P^{(t)}} N_{P^{(t)}})$, where $P^{(t)}$ is a sub-bipartite graph, $N = \min\{|\{r|r \in P^{(t)}\}|, |\{w|w \in P^{(t)}\}|\}$ and $M = \min\{|\{r|r \in P^{(t)}\}|, |\{w|w \in P^{(t)}\}|\}$. In the worst case, the entire bipartite graph is connected and cannot be split and the time complexity remains $O(M^2N)$, while in most cases, the bipartite graph can be split and the complexity would be reduced. Experimental results in Sec. 6 show that our optimization techniques improve the efficiency by up to 42.0%.

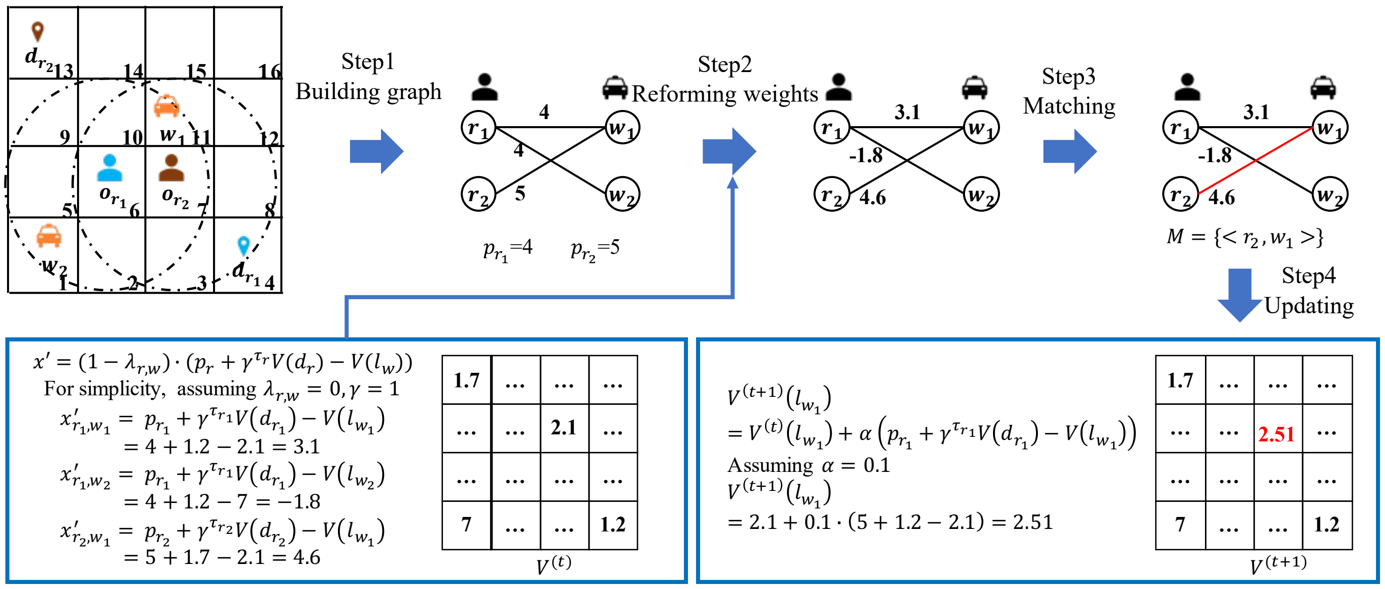
Algorithm 3: Matching algorithm $\mathscr{A}$.

```
Input: The bipartite graph
       $G^{(t)} =< R^{(t)}, W^{(t)}, E^{(t)} >$
Output: Matching $M^{(t)}$
1  for $\langle r, w, x_{r,w} \rangle \in G^{(t)}$ do
2      calculate $x'_{r,w}$ by Eq. (11)
3      $x'_{r,w} \leftarrow x_{r,w}$
4  end
5  $P^{(t)} \leftarrow$ split $G^{(t)}$ by BFS.
6  $M^{(t)} \leftarrow \emptyset$
7  for $p \in P^{(t)}$ do
8      if there is only one driver $w$ then
9          $r^* \leftarrow \arg\max_r x'_{r,w}$
10         $M_p \leftarrow \{\langle r^*, w \rangle\}$
11     end
12     else if there is only one order $r$ then
13         $w^* \leftarrow \arg\max_w x'_{r,w}$
14         $M_p \leftarrow \{\langle r, w^* \rangle\}$
15     end
16     else
17         $M_p \leftarrow$ Hungarian$(p)$
18     end
19     $M^{(t)} \leftarrow M^{(t)} \bigcup M_p$
20 end
21 return $M^{(t)}$
```

5.4 A Running Example of LTD

We now give an example in Fig. 3 to illustrate our method. For simplification, we only consider the square value function. We omit the scheduling step since the calculation is almost the same, We also assume the cancel probability is 0 and the discount factor is 1. Our approach contains 4 steps in a batch: building graph, reforming weights, matching and updating.

- **Building graph.** There are two drivers w_1, w_2 and two orders r_1, r_2. The prices of r_1 and r_2 are 4 and 5, respectively. For each order we choose the drivers within the distance threshold θ and build the bipartite graph with the price of orders as edge weights.
- **Reforming weights.** We reform the weights by the current value function based on Eq. (11). In this example, the weights of edges between r_1, w_1, r_1, w_2 and r_2, w_1 are 3.1, -1.8 and 4.6 respectively.



Step1 Building graph
Step2 Reforming weights
Step3 Matching
Step4 Updating
$p_{r_1}=4$ $p_{r_2}=5$
$M = \{<r_2, w_1>\}$
$x' = (1-\lambda_{r,w})\cdot(p_r + \gamma^{\tau_r}V(d_r) - V(l_w))$
For simplicity, assuming $\lambda_{r,w}=0, \gamma=1$
$x'_{r_1,w_1} = p_{r_1} + \gamma^{\tau_{r_1}}V(d_{r_1}) - V(l_{w_1}) = 4 + 1.2 - 2.1 = 3.1$
$x'_{r_1,w_2} = p_{r_1} + \gamma^{\tau_{r_1}}V(d_{r_1}) - V(l_{w_2}) = 4 + 1.2 - 7 = -1.8$
$x'_{r_2,w_1} = p_{r_2} + \gamma^{\tau_{r_2}}V(d_{r_2}) - V(l_{w_1}) = 5 + 1.7 - 2.1 = 4.6$
$V^{(t)}$
$V^{(t+1)}(l_{w_1}) = V^{(t)}(l_{w_1}) + \alpha\left(p_{r_1} + \gamma^{\tau_{r_1}}V(d_{r_1}) - V(l_{w_1})\right)$
Assuming $\alpha = 0.1$
$V^{(t+1)}(l_{w_1}) = 2.1 + 0.1\cdot(5 + 1.2 - 2.1) = 2.51$
$V^{(t+1)}$

Fig. 3. An running example of our LTD.

- **Matching.** We run the Hungarian algorithm (we omit the optimizations for simplification) to calculate the maximum matching allocation, *i.e.*, $\{\langle r_2, w_1\rangle\}$.
- **Updating.** Based on the matching results, we update the value function by Eq. (4). In this example, the value in driver w_1's location is updated to 2.51.

If we directly execute the Hungarian algorithm on the original bipartite graph, the matching result is $\{\langle r_1, w_2\rangle, \langle r_2, w_1\rangle\}$, whereas our approach has only matched one order. The intuition behind such decision (we do not match w_2 to r_1) is that w_2 is in a pretty good location: the value of l_{w_2} is 7, far higher than other places which means more orders with high prices may appear in this grid in the future. In that case, driver w_2 is unwilling to move to other grids unless for orders of sufficiently high price.

6 Experiment

6.1 Experiment Setting

Datasets and Simulation Environment. We conduct experiments on a simulator developed by a major ride-hailing platform. The simulator adopts batch-based dispatching with a batch size of 2 seconds. In each batch, it builds a bipartite graph based on the location of current idle drivers and the newly upcoming orders. Then the simulator executes the dispatching algorithm and cancels orders according to a cancel probability distribution that is unknown to the dispatching algorithm. Afterwards, it simulates the dynamics of the system, including the pick-up and delivery behavior of occupied drivers, random walks of idle drivers and the log-on and log-off of idle drivers. The simulator is built on real historical order and driver data in 3 cities in China (A, B and C) with 3 weeks. The average daily order number in the largest city is nearly 1 million.

Baselines. We compare LTD with 4 dispatching algorithms.

- Greedy Algorithm (GRE): a classical and naive dispatching solution. It sorts all edges on the bipartite graph by the descending order of edge weights in each batch, and then makes assignments by successively picking the edges with the largest weight and deleting it from the graph.
- Hungarian Algorithm (HUN) [36]: it executes the Hungarian algorithm on the bipartite graph in each batch and takes the matching results as dispatching decisions directly. We use the Hungarian algorithm with the slack optimization.
- Nearest Neighbor Priority (NNP) [6]: it converts the matching problem to a minimum cost maximum flow problem, where the cost is the distance between order and driver. The algorithm finds the matching allocation with the maximum matching cardinality and the minimum total cost in each batch.
- V-Net [11]: it trains the value function with deep neural network on historical data and assigns orders to drivers according to the learned value function. It is the deep reinforcement learning extension of [10].

Parameter Settings and Implementation. In our LTD, the learning rate α, discount factor γ are set to 0.025 and 0.9, respectively. The distance threshold θ is set to 3km. The side lengths of square and hexagon grids are set to 1100m and 645m. The parameters C and k in Eq. (13) are set to 0.01 and $\ln 20$. All algorithms are implemented by Python 3 and the experiments are conducted on Intel Xeon CPU E5-2630 v4 @ 2.20GHz with 12GB memory.

Evaluation metrics. We compare the performance of different algorithms via the following metrics.

- Total Utility (U): the summation of all the revenue of completed orders (see Eq. (1)).
- Respondence Rate (RR): the ratio of number of assigned orders to total number of orders, *i.e.*,

$$RR = \frac{\sum_{t=1}^{T} |\{o|\exists d, s.t.(o,d) \in M^{(t)}\}|}{\sum_{t=1}^{T} |O^{(t)}|} \tag{14}$$

- Completion Rate (CR): the ratio of the number of completed orders to the total number of orders, *i.e.*,

$$CR = \frac{\sum_{t=1}^{T} |\{o|\exists d, s.t. (o,d) \in M^{(t)} \wedge \mathbb{I}_{B(\lambda_{o,d})} = 0\}|}{\sum_{t=1}^{T} |O^{(t)}|} \tag{15}$$

- Time Consumption (TC): the time consumed for executing the matching algorithm.

A higher RR or CR often contributes to a higher U, so they are also utility metrics. A higher RR or CR also means a passenger is more likely to be served, indicating a higher user satisfaction. Time consumption is the efficiency metric.

6.2 Utility Results

Table 2 summarizes the overall results of U, RR and CR.

Metrics Comparison. Fig. 4(a), Fig. 5(a) and Fig. 6(a) illustrate the total utility comparisons over three cities. Influenced by weather, day of week and other contextual reasons, the order numbers and total utility varies with date. Our proposed method outperforms the baseline GRE in every city and on every day by 10.9% ~36.4%. Fig. 4(b), Fig. 5(b) and Fig. 6(b) summarize the comparison on RR. Still, our LTD consistently outperforms the others. Specifically, LTD achieves a 36.8% improvement at maximum and 19.1% on average. The results on CR are similar (see Fig. 4(c), Fig. 5(c) and Fig. 6(c)). LTD shows an enhancement at 19.8% and 37.9% on average and at maximum, respectively. As for the state-of-the-art V-Net, LTD outperforms 9.6%, 10.1% and 11.4% averagely on U, RR and CR, respectively.

Illustration of LTD's Advantages. We provide an intuitive illustration for why our LTD performs better than other methods. To be more conspicuous, we choose the most basic baseline GRE to compare with our LTD (see Fig. 7). We draw the spatial distribution of order-driver gap. Warmer colors indicate severer gap between orders and available drivers and requiring more drivers, while colder colors mean a superfluous number of drivers. A good order dispatch algorithm should maintain the balance between orders and drivers, which, reflecting on figures, means less warmer and colder areas. Compared with V-Net (see Fig. (a)), LTD (Fig. (b)) maintains more balance between the number of orders and available drivers.

6.3 Efficiency Results

Fig. 4(d), Fig. 5(d) and Fig. 6(d) show the results for efficiency. Since our LTD and V-Net both adopt some learning process, they are slightly slower than those simpler algorithms. However, with the acceleration tricks such as BFS split, our LTD runs faster than V-Net. Especially during rush hours, when the bipartite expands quickly, the execution time of our algorithm avoid a rapid growth. Our LTD is faster than V-Net by up to 42.0%.

7 Conclusion

In this paper, we propose Learning To Dispatch(LTD), an order dispatching algorithm which is both efficacious and efficient. Modeling the order dispatching problem from the view of multi-agent reinforcement learning, together with classical combinatorial optimization algorithm Hungarian, LTD shows a powerful evaluation on assignment candidates and provide a dispatching decision which outperforms current state-of-the-art order dispatching algorithms. Experiments performed on industrial simulator with real data show our LTD achieves 36.4% improvement on total utility and 42.0% reduction on time consuming at most over the state-of-the-art order dispatching algorithms.

Acknowledgments

We are grateful to anonymous reviewers for their constructive comments. This work is partially supported by the National Key Research and Development Program of China under Grant No. 2018AAA0101100, the National Science Foundation of China (NSFC) under Grant No. 61822201, U1811463, 62076017 and 61690202, the Science and Technology Project of Beijing No. Z191100002519012, the CAAI Huawei MindSpore Open Fund No. CAAIXSJLJJ-2020-020-A. and the State Key Laboratory of Software Development Environment Open Funding No. SKLSDE-2020ZX-07.

References

[1] R. R. Clewlow and G. S. Mishra, "Disruptive transportation: The adoption, utilization, and impacts of ride-hailing in the united states," *UC Davis: Institute of Transportation Studies Research Reports*, no. 7, pp. 1–35, 2017.

[2] Y. Wang, Y. Tong, C. Long, P. Xu, K. Xu, and W. Lv, "Adaptive dynamic bipartite graph matching: A reinforcement learning approach," in *ICDE*, 2019, pp. 1478–1489.

[3] L. Zhang, T. Hu, Y. Min, G. Wu, J. Zhang, P. Feng, P. Gong, and J. Ye, "A taxi order dispatch model based on combinatorial optimization," in *SIGKDD*, 2017, pp. 2151–2159.

[4] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu, "Online minimum matching in real-time spatial data: Experiments and analysis," *PVLDB*, vol. 9, no. 12, pp. 1053–1064, 2016.

[5] P. Xu, Y. Shi, H. Cheng, J. P. Dickerson, K. A. Sankararaman, A. Srinivasan, Y. Tong, and L. Tsepenekas, "A unified approach to online matching with conflict-aware constraints," in *AAAI*, 2019, pp. 2221–2228.

[6] L. Kazemi and C. Shahabi, "Geocrowd: enabling query answering with spatial crowdsourcing," in *SIGSPATIAL*, 2012, pp. 189–198.

[7] H. To, C. Shahabi, and L. Kazemi, "A server-assigned spatial crowdsourcing framework," *ACM Trans. Spatial Algorithms Syst.*, vol. 1, no. 1, pp. 2:1–2:28, 2015.

[8] U. ul Hassan and E. Curry, "A multi-armed bandit approach to online spatial task assignment," in *2014 IEEE Conference on Ubiquitous Intelligence and Computing*, 2014, pp. 212–219.

[9] J. P. Dickerson, K. A. Sankararaman, A. Srinivasan, and P. Xu, "Assigning tasks to workers based on historical data: Online task assignment with two-sided arrivals," in *AAMAS*, 2018, pp. 318–326.

[10] Z. Xu, Z. Li, Q. Guan, D. Zhang, Q. Li, J. Nan, C. Liu, W. Bian, and J. Ye, "Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach," in *SIGKDD*, 2018, pp. 905–913.

[11] X. Tang, Z. T. Qin, F. Zhang, Z. Wang, Z. Xu, Y. Ma, H. Zhu, and J. Ye, "A deep value-network based approach for multi-driver order dispatching," in *SIGKDD*, 2019, pp. 1780–1790.

[12] X. Geng, Y. Li, L. Wang, and et al, "Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting," in *AAAI*, 2019, pp. 3656–3663.

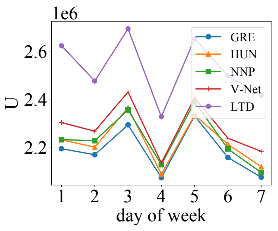
[13] G. Jin, Y. Cui, L. Zeng, H. Tang, Y. Feng, and J. Huang, "Urban ride-hailing demand prediction with multiple spatio-temporal information fusion network," *Transportation Research Part C: Emerging Technologies*, vol. 117, p. 102665, 2020.

[14] S. Ichoua, M. Gendreau, and J. Potvin, "Exploiting knowledge about future demands for real-time vehicle dispatching," *Transp. Sci.*, vol. 40, no. 2, pp. 211–225, 2006.

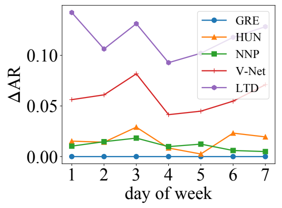
[15] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, pp. 9–44, 1988.

TABLE 2
Overall results on total utility U, respondence rate RR and completion rate CR of different algorithms. Larger numbers mean better performances.

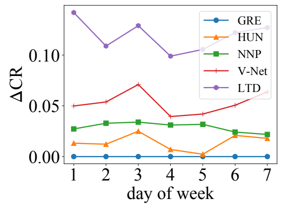
City	Method	U (weekday)	RR (weekday)	CR (weekday)	U (weekend)	RR (weekend)	CR (weekend)
A	GRE	$2,211,524$	$+0.00000$	$+0.00000$	$2,115,692$	$+0.00000$	$+0.00000$
	HUN	$2,242,993$	$+0.01390$	$+0.01192$	$2,165,011$	$+0.02130$	$+0.01939$
	NNP [6]	$2,265,687$	$+0.01308$	$+0.03137$	$2,143,352$	$+0.00548$	$+0.02298$
	V-Net [11]	$2,372,615$	$+0.05705$	$+0.05120$	$2,282,413$	$+0.06287$	$+0.05715$
	LTD (ours)	$\mathbf{2,554,002}$	$\mathbf{+0.11493}$	$\mathbf{+0.11685}$	$\mathbf{2,455,991}$	$\mathbf{+0.12305}$	$\mathbf{+0.12459}$
B	GRE	$1,356,998$	$+0.00000$	$+0.00000$	$1,442,235$	$+0.00000$	$+0.00000$
	HUN	$1,373,281$	$+0.01145$	$+0.00961$	$1,452,750$	$+0.00586$	$+0.00516$
	NNP [6]	$1,306,734$	-0.03295	-0.01441	$1,368,986$	-0.03813	-0.02124
	V-Net [11]	$1,435,992$	$+0.03247$	$+0.02865$	$1,549,928$	$+0.04069$	$+0.03388$
	LTD (ours)	$\mathbf{1,486,576}$	$\mathbf{+0.06150}$	$\mathbf{+0.05788}$	$\mathbf{1,625,191}$	$\mathbf{+0.08239}$	$\mathbf{+0.0721}$
C	GRE	$758,244$	$+0.00000$	$+0.00000$	$707,528$	$+0.00000$	$+0.00000$
	HUN	$771,253$	$+0.01049$	$+0.00896$	$718,777$	$+0.00875$	$+0.00756$
	NNP [6]	$780,672$	$+0.02210$	$+0.02598$	$706,033$	$+0.00691$	$+0.01342$
	V-Net [11]	$822,712$	$+0.03413$	$+0.02833$	$753,206$	$+0.02885$	$+0.02230$
	LTD (ours)	$\mathbf{1,002,708}$	$\mathbf{+0.15922}$	$\mathbf{+0.13509}$	$\mathbf{799,148}$	$\mathbf{+0.05789}$	$\mathbf{+0.05000}$



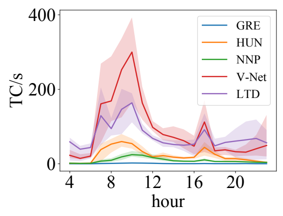
(a) U comparison



(b) RR comparison

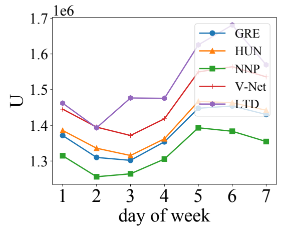


(c) CR comparison

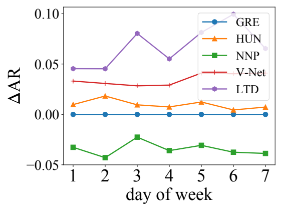


(d) TC comparison

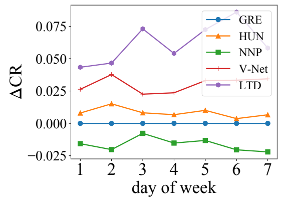
Fig. 4. Comparisons of U, RR, CR, TC among all methods on city A.



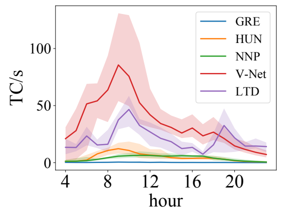
(a) U comparison



(b) RR comparison



(c) CR comparison



(d) TC comparison

Fig. 5. Comparisons of U, RR, CR, TC among all methods on city B.

[16] R. E. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*. SIAM, 2009.

[17] Y. Tong, Z. Zhou, Y. Zeng, L. Chen, and C. Shahabi, "Spatial crowdsourcing: a survey," *VLDB J.*, vol. 29, no. 1, pp. 217–250, 2020.

[18] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen, "Online mobile micro-task allocation in spatial crowdsourcing," in *ICDE*, 2016, pp. 49–60.

[19] Y. Tong, Y. Zeng, B. Ding, L. Wang, and L. Chen, "Two-sided online micro-task assignment in spatial crowdsourcing," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 5, pp. 2295–2309, 2021.

[20] Z. Liao, "Real-time taxi dispatching using global positioning systems," *Commun. ACM*, vol. 46, no. 5, pp. 81–83, 2003.

[21] D.-H. Lee, H. Wang, R. L. Cheu, and S. H. Teo, "Taxi dispatch system based on current demands and real-time traffic conditions," *Transportation Research Record*, vol. 1882, no. 1, pp. 193–200, 2004.

[22] J. P. Dickerson, K. A. Sankararaman, A. Srinivasan, and P. Xu, "Allocation problems in ride-sharing platforms: Online matching with offline reusable resources," in *AAAI*, 2018, pp. 1007–1014.

[23] Y. Li, J. Fang, Y. Zeng, B. Maag, Y. Tong, and L. Zhang, "Two-sided online bipartite matching in spatial data: experiments and analysis," *GeoInformatica*, pp. 175–198, 2020.

[24] H. Luo, Z. Bao, F. M. Choudhury, and J. S. Culpepper, "Dynamic ridesharing in peak travel periods," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 7, pp. 2888–2902, 2021.

[25] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows - theory, algorithms and applications*. Prentice Hall, 1993.

[26] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[27] A. Schrijver, *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[28] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[29] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

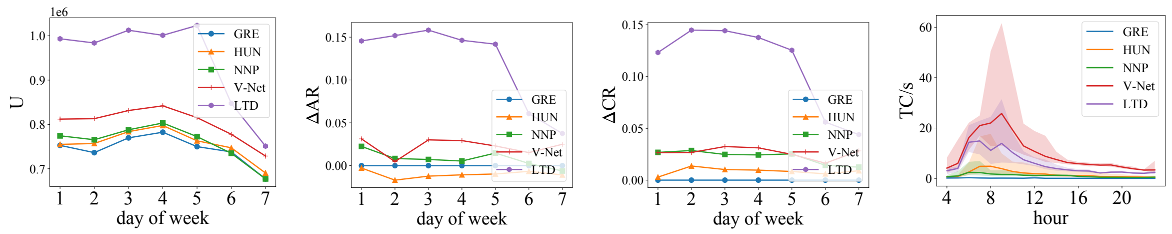
[30] J. Holler, R. Vuorio, Z. Qin, X. Tang, Y. Jiao, T. Jin, S. Singh, C. Wang, and J. Ye, "Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem," in *ICDM*, 2019, pp. 1090–1095.

[31] J. Hu and M. P. Wellman, "Multiagent reinforcement learning: Theoretical framework and an algorithm," in *ICML*, 1998, pp. 242–250.

[32] A. Alshamsi, S. Abdallah, and I. Rahwan, "Multiagent self-organization for a taxi dispatch system," in *AAMAS*, 2009, pp. 21–28.

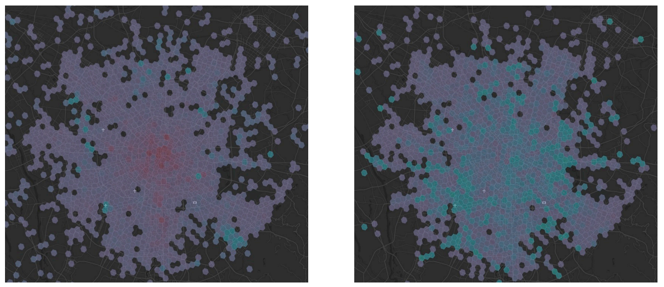
[33] K. T. Seow, N. H. Dang, and D. Lee, "Towards an automated multiagent taxi-dispatch system," in *CASE*, 2007, pp. 1045–1050.

[34] M. Li, Z. Qin, Y. Jiao, Y. Yang, J. Wang, C. Wang, G. Wu, and J. Ye,



(a) U comparison (b) RR comparison (c) CR comparison (d) TC comparison

Fig. 6. Comparisons of U, RR, CR, TC among all methods on city C.



(a) V-Net (b) LTD

Fig. 7. Spatial distribution of order-driver number gap, warmer color means a larger gap between numbers of orders and drivers.

"Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning," in *WWW*, 2019, pp. 983–994.

[35] A. Mehta, "Online matching and ad allocation," *Found. Trends Theor. Comput. Sci.*, vol. 8, no. 4, pp. 265–368, 2013.

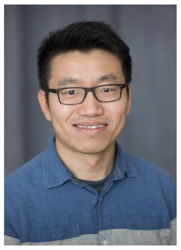
[36] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the ACM*, vol. 19, no. 2, pp. 248–264, 1972.

Yongxin Tong received the Ph.D. degree in computer science and engineering from the Hong Kong University of Science and Technology in 2014. He is currently a professor in the School of Computer Science and Engineering, Beihang University. His research interests include big spatio-temporal data analytics, crowdsourcing, reinforcement learning, federated learning and smart city. He received the championship of KDD Cup 2020 RL track and VLDB 2014 Excellent Demonstration Award. He is a member of the IEEE.

Dingyuan Shi is currently a postgraduate student in the School of Computer Science and Engineering, Beihang University. His research interests include reinforcement learning and big spatio-temporal data analytics. He won the championship of KDD Cup 2020 RL track.

Yi Xu received his B.E, M.E. and Ph.D. degree from Beihang University in 2009, 2013, and 2020, respectively. He is an assistant professor in the Institute of Artificial Intelligence, Beihang University. His research interests include big spatio-temporal data analytics and mining, crowdsourcing, crowd intelligence, federated learning and smart city.

Weifeng Lv received the PhD degree in computer science from Beihang University. He is the vice-president and a professor of Beihang University. His research interests include big spatio-temporal data analytics, smart city, crowdsourcing, crowd intelligence and reinforcement learning. He is the leader of the Group of National Smart City Standard.



Zhiwei (Tony) Qin is Principal Research Scientist and Director of the Decision Intelligence group at DiDi AI Labs, working on core problems in ridesharing marketplace optimization. Tony received his Ph.D. in Operations Research from Columbia University. His research interests span optimization and machine learning, with a particular focus in reinforcement learning and its applications in operational optimization, digital marketing, and smart transportation. He has published and served in the program committee of top-tier conferences and journals in machine learning and optimization. He and his team received the INFORMS Daniel H. Wagner Prize for Excellence in Operations Research Practice in 2019 and were selected for the NeurIPS 2018 Best Demo Awards.

Xiaocheng Tang Dr. Xiaocheng Tang is a research scientist in DiDi AI Labs. Since his graduate study Dr. Tang has been actively engaged in analyzing and designing practical intelligent algorithms at the intersection of machine learning, optimization, and most recently, RL and OR. His work on joint optimization of order dispatching and reposition via reinforcement learning won Best Demo Awards at NeurIPS 2018. He has also served as PC of conferences and journals including NeurIPS, ICML, ICLR, AAAI, JMLR and SIOPT.