IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 14, 2020, accepted July 30, 2020, date of publication August 4, 2020, date of current version August 17, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3014076

Dynamic Fleet Management With Rewriting Deep Reinforcement Learning

WENQI ZHANG, QIANG WANG, (Member, IEEE), JINGJING LI, AND CHEN XU
National Engineering Laboratory for Mobile Network Technologies, Beijing University of Posts and Telecommunications, Beijing 100876, China
Corresponding author: Qiang Wang (wangq@bupt.edu.cn)

This work was supported by the National Key Research and Development Program of China under Grant 2018YFE0205503.

ABSTRACT Inefficient supply-demand matching makes the fleet management a research hotpot in ride-sharing platforms. With the booming of mobile network services, it is promising to abate the supply-demand gap with effective vehicle dispatching. In this article, we propose a QRewriter - Dueling Deep Q-Network (QRewriter-DDQN) algorithm, to dispatch multiple available vehicles in ahead to the locations with high demand to serve more orders. The QRewriter-DDQN algorithm factorizes into a Dueling Deep Q-Network (DDQN) module and a QRewriter module, which are parameterized by neural networks and Q-table with Reinforcement Learning (RL) methods, respectively. Particularly, DDQN module utilizes the Kullback-Leibler (KL) distribution distance between supply (available vehicles) and demand (orders) as excitation to capture the complex dynamic variations of supply-demand. Afterwards, the QRewriter module learns to improve the DDQN dispatching policy with the streamlined and effective Q-table in RL. Importantly, the higher performance improvement space of the DDQN dispatching policy can be obtained by aggregating QRewriter state into low-dimension meta state. A simulator is designed to train and test the performance of QRewriter-DDQN, the experiment results show the significant improvement of QRewriter-DDQN in terms of order response rate.

INDEX TERMS Deep reinforcement learning (DRL), fleet management, learn to improve, multi-agent.

I. INTRODUCTION

With the vigorous development of ride-sharing platforms, such as DiDi Chuxing [1] and Uber [2], transportation becomes convenient and flexible. In the meantime, fleet management has attracted many researchers' attention as efficient and effective vehicle dispatching is able to make full use of traffic resources [3]–[5]. Consequently, how to make vehicle dispatching decisions to abate the traffic supply-demand gap, enhance customer experience and improve the order response rate is a significant problem.

The fleet management (vehicle dispatching) is a complex dynamic process as the dispatching decisions for current vehicles will affect the gap of future traffic supply-demand. Fortunately, Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL) are able to capture the complex dynamic supply-demand variations and provide the opportunity to solve the vehicle dispatching problem. Recently, there are some existing works successfully solve the fleet management problem with RL and DRL techniques under many scenes [6]–[14]. For example, the authors in [6] found a near-optimal solution for Vehicle Routing Problem (VRP), which is a variation of fleet management problem, under the traffic network with a single vehicle by using the RL method. In [7], the authors employed DRL method to tackle Traveling Salesman Problem (TSP), which also is a variation of fleet management problem, with the known demand distribution. Nevertheless, in actual traffic, there are huge vehicles in the traffic network and the traffic demand cannot be obtained in advance. Consequently, the methods for solving multi-vehicle dispatching problem are expected.

For multi-vehicle dispatching problem, the supply (available vehicles) distribution and demand (orders) distribution should be provided to RL/DRL machines to capture the dynamic supply-demand distribution variations expediently and efficiently. For example, the authors in [8] designed a layered multi-agent DRL algorithm to solve the joint order dispatching and vehicle dispatching problem with inputting the real-time vehicle distribution and order distribution. In [9], an end-to-end multi-agent DRL algorithm was proposed to dispatch the nearby orders to idle vehicles and dispatch the idle vehicles without nearby orders

The associate editor coordinating the review of this manuscript and approving it for publication was Yang Tang.

to the locations with lager demand. Additionally, the lager and complex supply-demand distribution also provided to the proposed DRL algorithm as input. The authors in [10] combined Convolutional Neural Network (CNN) and RL to plan routes for vehicles to deal with complex supply-demand variations and obtain satisfactory passenger waiting time. In [11], a distributed DRL algorithm with supply-demand distribution as input was proposed to design vehicle routing to improve the performance of ride-sharing platform. A model-free DRL algorithm was proposed in [12] to reduce unserved orders by solving the dynamic fleet management problem, and the complex supply-demand distribution as a part of state input. In [13], a distributed model-free DRL algorithm was designed to learn effective vehicle dispatching policy with the New York taxi dataset. However, to let agents learn independently, the supply-demand distribution is regarded as the input of each agent. The authors in [14] proposed a multi-agent DRL algorithm to dispatch available vehicles to high-demand locations. Furthermore, the real-time traffic supply-demand distribution used as input and the contextual information is introduced into DRL frame to reduce the state-action space. According to the aforementioned researches, the supply-demand distribution is necessary for solving the multi-vehicle dispatching problem. However, the complex supply-demand distribution will case a massive state- action space since there are thousands of vehicles and orders in the traffic network at each time interval. Besides, the massive state-action space reduces the efficiency of DRL frames and limits the performance space of available vehicle dispatching. Therefore, the effective solution to abating the effect of larger state-action space and providing higher performance improvement space of multi-vehicle dispatching problem is expected.

In this article, we propose a multi-vehicle dispatching algorithm, QRewriter - Dueling Deep Q-Network (QRewriter-DDQN) algorithm, to dispatch available vehicles to high-demand locations to serve more orders. Compared to the existing works, QRewriter-DDQN algorithm is capable of aggregating high-dimension supply-demand state into low-dimension meta state and abating the effect of complex supply-demand dimension. In addition, QRewriter-DDQN agents autonomously learn to rewrite the dispatching policy under the low-dimension meta state, and higher improvement space can be obtained. Therefore, due to the above advantages, QRewriter-DDQN algorithm is able to achieve stable and sterling dispatching performance. Finally, our detailed contributions are as follows,

- We consider the available vehicle dispatching under the dynamic traffic supply-demand variations. Multi-agent DRL algorithm is leveraged to learn the dispatching policy to satisfy more demand and improve the order response rate. In particular, the KL distribution distance between supply and demand acted as the reward signal to inspire the multi-agent DRL algorithm.
- We propose the QRewriter-DDQN algorithm, which is composed of DDQN module [15] and QRewriter module, to let agents learn to improve the dispatching policy with the streamlined and effective Q-table. Furthermore, the low-dimension QRewriter state can be obtained by aggregating the complex supply-demand distribution into a low-dimension meta state. The QRewriter module is capable of abating the effect of larger supply-demand distribution with the low-dimension QRewriter state and acquiring strong dispatching performance.
- Last but not least, a simulator is designed to train and test the performance of our proposed QRewriter-DDQN algorithm. Furthermore, experiments with different vehicle initializations are demonstrated to test the effectiveness and robustness of QRewriter-DDQN algorithm. The experiment results demonstrate QRewriter-DDQN outperforms the state-of-the-art algorithm like multi-agent DDQN and other baselines in terms of order response rate.

The reminder of this article is organized as follows. Our problem formulation is presented in Section II. In Section III, the background of DRL/RL algorithm and the proposed QRewriter-DDQN algorithm, which included the DDQN module and the QRewriter module, for multi-driver dispatching problem are demonstrated. Section IV analyzes the experiment results. At last, Section V concludes this article.

II. PROBLEM FORMULATION

In this article, we investigate the fleet management problem (multi-vehicle dispatching problem) of ride-sharing platform with the objective of maximizing the order response rate. To tackle the multi-driver dispatching problem, we propose QRewriter-DDQN algorithm to dispatch the idle vehicles in ahead to the locations with larger demand and serve more orders. In practice, the given area is divided into many hexagonal grids and one day is spilt into 144 time intervals in order to facilitate available vehicle dispatching. Furthermore, there are two significant modules in our proposed algorithm, DDQN module and QRewriter module. DDQN module gives the dispatching action policy to each idle vehicle and QRewriter module rewrites the dispatching action policy, which based on the DDQN dispatching policy, for each available vehicle. Afterwards, according to our dispatching policy, the available vehicles are dispatched to places with high demand and serve more orders. The overall architecture of the QRewriter-DDQN algorithm is demonstrated in Fig. 1.

A. DDQN MODULE

Obviously, the fleet dispatching problem in DDQN module is a MDP problem, and each idle vehicle is an agent. Consequently, we employ a five-tuple $\langle \mathcal{S}_{\mathcal{D}}, \mathcal{A}_{\mathcal{D}}, \mathcal{R}_{\mathcal{D}}, \mathcal{P}_{\mathcal{D}}, \gamma_D \rangle$ to exhibit the problem. The detailed explanation as follows,

- **State Space** $\mathcal{S}_{\mathcal{D}}$: State of agent n, $s_D^n \in \mathcal{S}_{\mathcal{D}}$. We define the state as $(D_v, D_o, ID_{time}, ID_{grid})$, where D_v denotes the distribution of available vehicles, D_o denotes the distribution of orders, ID_{time} denotes the ID of time

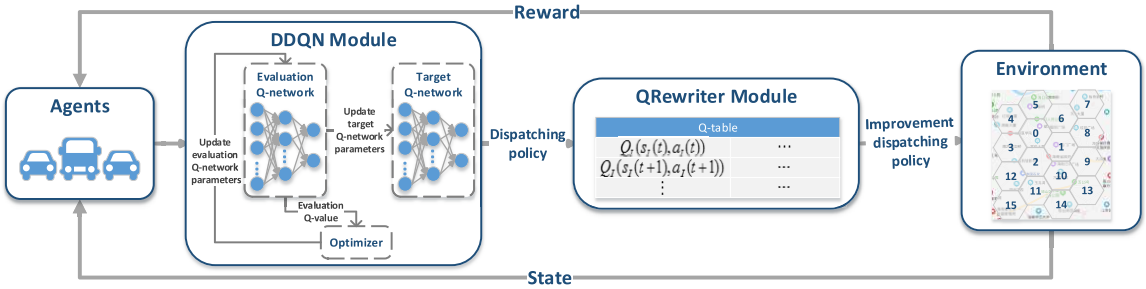

FIGURE 1. Overall architecture of QRewriter-DDQN algorithm.

interval and ID_{grid} denotes the ID of grid where the idle vehicle is located. Specially, ID_{time} and ID_{grid} are one-hot code in order to distinguish the time and grid effectively. The state describes the detailed information of time-space and demand-supply distribution.

- **Action Space** $\mathcal{A}_{\mathcal{D}}$: Action of agent n, $a_D^n \in \mathcal{A}_{\mathcal{D}}$. Each agent has seven actions, which is described as $\{0, 1, 2, 3, 4, 5, 6\}$. Particularly, the seven actions indicate the available vehicles are dispatched to the current grid and the six adjacent grids, respectively. For example, $a_D^n = 0$ means the available stays in the current grid and $a_D^n = 1$ means the available vehicle is dispatched to the first adjacent grid.
- **Reward Space** $\mathcal{R}_{\mathcal{D}}$: Reward of agent n, $r_D^n \in \mathcal{R}_{\mathcal{D}}$. The immediate reward is defined as

$$r_D^n = p_v^i(t) log \frac{p_v^i(t)}{p_o^i(t)} - p_v^i(t+1) log \frac{p_v^j(t+1)}{p_o^j(t+1)}, \quad (1)$$

where $p_v^i(t)$ and $p_o^i(t)$ represent the probabilities of vehicle appears in grid i at time t and order appears in grid i at time t, respectively. In essence, the reward is the KL distance between vehicle distribution and order distribution as the goal of DDQN module is to minimize the distance between demand distribution and supply distribution.

- **Transmission probability** $\mathcal{P}_{\mathcal{D}}$: Explaining the probability from the current state $s_D(t)$ to the next state $s_D(t+1)$ under a specific action.
- **Reward discount factor** γ_D: γ_D is utilized to balance the current profit and future profit.

B. QRewriter MODULE

To obtain a stable and better dispatching policy, the QRewriter module employs state aggregation to reduce the state-action space and refine the dispatching policy which is obtained from DDQN module. In QRwriter module, the agents are also the idle vehicles. In detail, we use a five-tuple $\langle \mathcal{S}_{\mathcal{I}}, \mathcal{A}_{\mathcal{I}}, \mathcal{R}_{\mathcal{I}}, \mathcal{P}_{\mathcal{I}}, \gamma_I \rangle$ to describe the QRewriter module. The detailed explanation as follows,

- **State Space** $\mathcal{S}_{\mathcal{I}}$: State of agent n, $s_I^n \in \mathcal{S}_{\mathcal{I}}$. The state is defined as $(a_D^n, ID_{time}, ID_{grid})$, where a_D^n is the action which DDQN module chooses for agent n, ID_{time} is the ID of time interval and ID_{grid} is the ID of grid where the agent located. Same as DDQN module, the ID_{time} and ID_{grid} are expressed with one-hot code. The low-dimension QRewriter state obtained by aggregating the complex supply-demand distribution into a_D^n with the state aggregation (DDQN module) and we call the low-dimension QRewriter state as meta state. In detail, the core of supply-demand distribution is aggregated into a_D^n as the DDQN module makes dispatching decision a_D^n based on the complex and larger supply-demand distribution. Importantly, the small meta state dimension provides the opportunity to store Q-value with Q-table.
- **Action Space** $\mathcal{A}_{\mathcal{I}}$: Action of agent n, $a_I^n \in \mathcal{A}_{\mathcal{I}}$. Each agent owns seven actions, which is described as $\{0, 1, 2, 3, 4, 5, 6\}$. The Seven actions indicate the seven rewriting directions. In detail, $a_I^n = 2$ means that the agent will explore the second adjacent grid of the grid the agent located in.
- **Reward Space** $\mathcal{R}_{\mathcal{I}}$: Immediate reward of agent n, $r_I^n \in \mathcal{R}_{\mathcal{I}}$. The immediate reward r_I^n is depend on the improvement gain, which is expressed as

$$G = r_D(i, j) - r_D(i, k), \quad (2)$$

where $r_D(i, j)$ denotes the DDQN reward from grid i to grid j and $r_D(i, k)$ denotes the DDQN reward from grid i to grid k. Specially, grid j and grid k are the dispatching results of DDQN module and QRewriter module, respectively. The DDQN reward $r_D > 0$ means that agent is dispatched to a high-demand location and $r_D < 0$ means that agent is dispatched to a low-demand location. Consequently, the improvement gain $G > 0$ indicates the QRewriter module finds a better dispatching policy and we let the improvement reward $r_I^n = G$. When $G < 0$ indicates that the QRewriter module finds a terrible improvement dispatching policy and we let the improvement reward $r_I^n = -1000$. In summary, the improvement reward is defined as

$$r_I^n = \begin{cases} G, & G > 0 \\ -1000, & G < 0. \end{cases} \quad (3)$$

- **Transmission probability** $\mathcal{P}_{\mathcal{I}}$: Transmission probability in QRewriter module.

- **Reward discount factor** γ_I: Reward discount factor in QRewriter module.

C. EXAMPLE

To be more concrete, an example is demonstrated in Fig. 2. At time 1, agent 1 observes state $s_D^1(1)$ from environment and chooses action $a_D^1(1)$ according to the DDQN dispatching policy. Then, DDQN module receives the reward $r_D^1(1) = p_v^0(1)log\frac{p_v^0(1)}{p_o^1(1)} - p_v^1(2)log\frac{p_v^1(2)}{p_o^1(2)}$. Afterwards, the QRewriter module obtains state $S_I^1(1)$ and takes action $a_I^1(1)$ according to the improvement dispatching policy. The improvement gain is computed as $r_D(0, 1) - r_D(0, 3)$ and the improvement reward $r_I^1(1)$ is computed as equation (3). Finally, agent 1 is dispatched to grid 3 according to the improvement dispatching policy. Specially, the dispatching action of DDQN module is just a "suggest", the agents are dispatched only according to the dispatching action of QRewriter module.

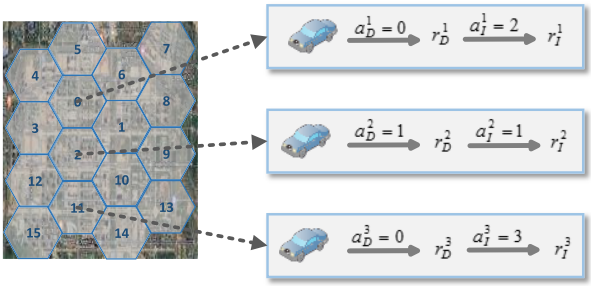

FIGURE 2. Example problem.

III. PROPOSED ALGORITHMS FOR MULTI-DRIVER DISPATCHING

A. BACKGROUND OF RL ALGORITHM AND DRL ALGORITHM

RL algorithm [16] is a ML algorithm that relies solely on rewards. A RL agent, the action executor, interacts with environment and obtains a numerical reward based on the current state and action. Normally, a five-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$ is used to define the model of RL:

- $\mathcal{S}$: State space, which includes all states of RL agents ($s \in \mathcal{S}$);
- $\mathcal{A}$: Action space, which includes all actions of RL agents ($a \in \mathcal{A}$);
- $\mathcal{R}$: Reward space, which includes all immediate reward $r_{s,a}$ of RL agents ($r_{s,a} \in \mathcal{R}$);
- $\mathcal{P}$: Transmission probability;
- γ: Reward discount factor.

In detail, at time t, a RL agent observes a state s_t from the environment and takes an action a_t according to the action policy π, which is a series of consecutive actions. After taking a_t, the environment gives an immediate reward r_{s_t,a_t} to the agent. The RL model uses the existing information to compute the state-action value (Q-value) $Q^\pi(s_t, a_t)$,

$$Q^\pi(s_t, a_t) = r_{s_t,a_t} + \gamma Q^\pi(s_{t+1}, a_{t+1}), \quad (4)$$

where s_{t+1} and a_{t+1} represent the state and action of the next time, respectively. Then, the state-action value $Q^\pi(s_t, a_t)$ will be stored into the brain of RL model, Q-table. On the whole, RL model aims to obtain an optimal action policy $\pi^\star$ and maximize the state-action value $Q^{\pi^\star}(s_t, a_t)$,

$$Q^{\pi^\star}(s_t, a_t) = E[r_{s_t,a_t} + \gamma \max_{a_{t+1}} Q^\pi(s_{t+1}, a_{t+1}|s_t, a_t)]. \quad (5)$$

Obviously, equation (5) is a Bellman optimality problem, and dynamic programing is an effective method to solve the Bellman optimality problem.

The RL model is able to obtain the optimal action policy with accurate Q-values. Therefore, Q-value is a vital element in the RL model. In RL model, Q-table is used to store and output Q-values. Nevertheless, when the RL model has substantial states, the computing complexity cannot be managed and Q-table no longer an effective approach. Fortunately, DRL model employs neural networks instead the RL's Q-table to estimate (output) the Q-values and solves the substantial states problem.

B. DDQN MODULE ALGORITHM FOR MULTI-DRIVER DISPATCHING

DDQN algorithm, a DRL algorithm, is composed of ϵ-greedy action policy, evaluation Q-network, target Q-network and action advantage function. In the following, each part is explained in detail.

1) ϵ-GREEDY ACTION POLICY

At each DDQN training step, the brain of DDQN chooses and takes an action. In theory, the brain should choose the current optimal action. However, the current optimal action in training stage is not the real optimal action since some action values not be updated. Thus, always taking the current optimal action will cause DDQN algorithm fall into local optimum. Fortunately, ϵ-greedy action policy [17] is an effective approach to solve the local optimum problem. The core of ϵ-greedy action policy is exploration-exploitation dilemma. In detail, the brain of DDQN will choose the current optimal action with probability $1-\epsilon$ and choose a random action with probability ϵ,

$$a = \begin{cases} current\ optimal\ action: \arg\max_a Q, & \alpha > \epsilon \\ random\ action, & \alpha < \epsilon, \end{cases} \quad (6)$$

where ϵ is the exploring probability and α is a random float number belongs to [0, 1]. Generally, the ϵ-greedy action policy provides the opportunity to jump out from the local optimum and explore new actions.

2) EVALUATION Q-NETWORK

Each DDQN agent obtains an immediate reward from environment after taking an action according to ϵ-greedy action policy. The long-term accumulated reward is composed of the immediate reward,

$$r_D(t)+\gamma_D r_D(t+1)+\gamma_D^2 r_D(t+2)+\cdots+\gamma_D^{T-t} r_D(T), \quad (7)$$

where γ_D denotes the reward discount factor in DDQN module. Specially, the except value of long-term accumulated reward is the Q-value. In DDQN module, we use $Q_D(s_D(t), a_D(t))$ for denoting the Q-value,

$$Q_D(s_D(t), a_D(t)) = E[\sum_{k=0}^{T} \gamma_D^k r_D(t+k) | s_D(t), a_D(t)]. \quad (8)$$

The goal of DDQN model is to maximize Q-value. However, the optimal Q-value of next time is needed when calculate the optimal Q-value of current time. Consequently, Q-value can be expressed as,

$$\begin{aligned} &Q_D(s_D(t), a_D(t)) \\ &= (1-\Gamma_D) Q_D(s_D(t), a_D(t)) \\ &\quad + \Gamma_D (r_D(t) + \gamma_D \max_{a_D(t+1)} Q_D(s_D(t+1), a_D(t+1))), \end{aligned} \quad (9)$$

where Γ_D is learning rate in the DDQN module. Obviously, a precise sate-action value is vital for DDQN model to seek out the optimal action policy and maximize state-action value. For DDQN, the model utilizes neural network to estimate state-action value, and we call the neural network as evaluation Q-network. Afterwards, the Q-value is expressed as,

$$\begin{aligned} &Q_D(s_D(t), a_D(t); \theta) \\ &= (1-\Gamma_D) Q_D(s_D(t), a_D(t)) \\ &\quad + \Gamma_D (r_D(t) + \gamma_D \max_{a_D(t+1)} Q_D(s_D(t+1), a_D(t+1); \theta)), \end{aligned} \quad (10)$$

where θ represents the parameters of evaluation Q-network. Besides, we update the parameters of evaluation Q-network by minimizing the loss $J(\theta)$,

$$\begin{aligned} J(\theta) = E[(Q_D(s_D(t), a_D(t); \theta) - (r_D(t) \\ + \gamma_D \max_{a_D(t+1)} Q_D(s_D(t+1), a_D(t+1); \theta)))^2], \end{aligned} \quad (11)$$

with gradient descent method [18].

3) TARGET Q-NETWORK

To improve the stability of DDQN, another independent neural network, target Q-network, is introduced into DDQN model. The core of target Q-network is to use the output of target Q-network at time $t+1$ instead of the output of evaluation Q-network at time $t+1$ when compute the Q-value at time t. Therefore, the equation (10) is turned into,

$$\begin{aligned} &Q_D(s_D(t), a_D(t); \theta) \\ &= (1-\Gamma_D) Q_D(s_D(t), a_D(t)) \\ &\quad + \Gamma_D (r_D(t) + \gamma_D \max_{a_D(t+1)} Q_D(s_D(t+1), a_D(t+1); \theta')), \end{aligned} \quad (12)$$

where θ' represents the parameters of target Q-network. In addition, we update the parameters of target Q-network with the following equation periodically,

$$\theta' = \beta\theta + (1-\beta)\theta', \quad (13)$$

where β is the update rate. To reduce the relevance of training samples, the memory replay is introduced into the DDQN model. At each training step, a four-tuple $\langle s_D(t), a_D(t), r_D(t), s_D(t+1) \rangle$ is stored into memory replay. Afterwards, a mini-batch of samples are selected randomly from memory replay to train the parameters of evaluation Q-network.

4) ACTION ADVANTAGE FUNCTION

Action advantage function allows agents reference the average action value when choose action and improve the stability and robustness of the DDQN model. After introducing the action advantage function into DDQN model, the equation (10) is expressed as,

$$\begin{aligned} Q_D(s_D(t), a_D(t); \theta) = V(s_D(t); \theta) + A(s_D(t), a_D(t); \theta) \\ - \frac{1}{|A_D|} \sum_{a(t) \in A_D} A(s_D(t), a(t); \theta), \end{aligned} \quad (14)$$

where $V(s_D(t))$ represents the state value of state $s_D(t)$, $A(s_D(t), a_D(t))$ represents the action value of state $s_D(t)$ under the action $a_D(t)$, $A(s_D(t), a_D(t); \theta) - \frac{1}{|A_D|} \sum_{a(t) \in A_D} A(s_D(t), a(t); \theta)$ represents action advantage value of action $a_D(t)$ and $|A_D|$ represents the dimension of DDQN module's action space. The action advantage function expresses the advantage of an action a relative to the average value of state s. From the perspective of quantitative relationship, it is the deviation of random variable relative to mean value. In detail, if action a is better than the mean value, then action advantage value is positive; otherwise, it is negative. Consequently, a relatively stable Q-value can be guaranteed by selecting the action according to action advantage function.

The details of DDQN module algorithm is demonstrated in Algorithm 1.

C. QRewriter MODULE ALGORITHM FOR MULTI-DRIVER DISPATCHING

Since the complex supply-demand distribution is aggregated in current DDQN action and we obtain the small meta state space (QRewriter state space). In particular, the small meta state dimension provides the opportunity to store Q-value with Q-table. Consequently, in this module, we propose RL algorithm, Q-learning algorithm, to improve the action policy which obtained from DDQN module. At each Q-learning training step, each agent observes a state $s_I(t)$ from environment and takes an action $a_I(t)$ which also according to the ϵ-greedy action policy. Afterwards, the Q-learning model checks if the state $s_I(t)$ already exists in the Q-table. Q-table is shown in Table 1, where $Q_I(s_I(t), a_I^1(t))$ denotes the Q-value under the state $s_I(t)$ with taking the first action $a_I^1(t)$ in the action space. If the state already exists in the Q-table, we update the Q-value $Q_I(s_I(t), a_I(t))$ at the special location in the Q-table according to the following equation,

$$\begin{aligned} &Q_I(s_I(t), a_I(t)) \\ &= (1-\Gamma_I) Q_I(s_I(t), a_I(t)) \\ &\quad + \Gamma_I (r_I(t) + \gamma_I \max_{a_I(t+1)} Q_I(s_I(t+1), a_I(t+1))), \end{aligned} \quad (15)$$

where Γ_I denotes the learning rate in the QRewriter module. We add a new line to Q-table to store the information

TABLE 1. The detailed information of Q-table.

$Q_I(s_I(t), a_I^1(t))$	$Q_I(s_I(t), a_I^2(t))$	...	$Q_I(s_I(t), a_I^7(t))$
$Q_I(s_I(t+1), a_I^1(t+1))$	$Q_I(s_I(t+1), a_I^2(t+1))$	...	$Q_I(s_I(t+1), a_I^7(t+1))$
$Q_I(s_I(t+2), a_I^1(t+2))$	$Q_I(s_I(t+2), a_I^2(t+2))$	...	$Q_I(s_I(t+2), a_I^7(t+2))$
⋮	⋮	⋮	⋮

Algorithm 1 DDQN Module Algorithm for Multi-Driver Dispatching Problem

Input: Initial distribution of available vehicles and initial distribution of orders.

for $t = t_\triangle, 2t_\triangle, \ldots, 144t_\triangle$ **do**
  for *each available vehicle* **do**
    Taking an action according to ϵ-greedy action policy ;
    Storing $(s_D(t), a_D(t), r_D(t), s_D(t+1))$ into the memory replay;
    Sampling some samples from memory replay;
    Updating the evaluation Q-network with $\theta \leftarrow \theta + \nabla_\theta J(\theta)$;
    Updating the target Q-network with $\theta' \leftarrow \beta\theta + (1-\beta)\theta'$, periodically;

Output: The dispatching actions for available vehicles.

Algorithm 2 QRewriter Module Algorithm for Multi-Driver Dispatching Problem

Input: The dispatching actions of available vehicles.

for $t = t_\triangle, 2t_\triangle, \ldots, 144t_\triangle$ **do**
  for *each available vehicle* **do**
    Taking an action according to ϵ-greedy action policy ;
    Checking the current state if already exists in the Q-table;
    If yes, update Q-table with equation (15);
    If no, add a new line into Q-table and update Q-value;

Output: The improvement dispatching actions for available vehicles.

(Q-value) of state $s_I(t)$, otherwise. Since the limited and managed state-action dimension allows Q-learning model gain an integral Q-table, we are capable of getting an integral Q-table through continuous training and updating Q-table. Moreover, stable and precise Q-values are able to be extracted from Q-table when the algorithm convergences. Therefore, an optimal improvement action policy will be obtained with the stable and precise Q-values. The QRewriter module algorithm is shown in Algorithm 2.

D. ANALYSIS OF THE PROPOSED ALGORITHM

1) COMPLEXITY OF THE PROPOSED ALGORITHM

For the DDQN module, the complexity is $O(|S_D|^2|A_D|)$, where $|S_D|$ and $|A_D|$ represent the state space dimension and action space dimension of DDQN module, respectively. For the QRewriter module, the complexity is $O(|S_I|^2|A_I|)$, where $|S_I|$ and $|A_I|$ denote the state space dimension and action space dimension of QRewriter module, respectively. Consequently, the complexity of our proposed algorithm is $O(|S_D|^2|A_D| + |S_I|^2|A_I|)$. Since $|A_D|$ is equal to $|A_I|$, the complexity can be simplified as $O(|A_I|(|S_D|^2 + |S_I|^2))$. Obviously, the state space dimension of DDQN module and QRewriter module dominate the complexity of our proposed algorithm. In addition, $|S_I| < |S_D|$ since the available vehicles distribution and order distribution are aggregated into the QRewriter state.

2) CONVERGENCE OF THE PROPOSED ALGORITHM

The essential problems of the DDQN module and QRewriter module are the Bellman problems. For the Bellman equation, we let

$$F(s,a) = r + \gamma \max Q(s,a) - Q^\star(s,a). \tag{16}$$

Afterwards, we can obtain

$$\begin{aligned} E[F(s,a)|F] &= \sum_{s\in S} p^a_{s,s'}[r + \gamma \max Q(s,a) - Q^\star(s,a)] \\ &= (HQ)(s,a) - Q^\star(s,a) \\ &= (HQ)(s,a) - (HQ^\star)(s,a) \\ &\le \gamma ||d||_\infty, \end{aligned} \tag{17}$$

where $d = Q(s,a) - Q^\star(s,a) = (1-\Gamma)Q(s,a) - \Gamma(r + \max_{a'} Q(s',a') - Q^\star(s,a))$. According to [19] and [20], the bounded DDQN reward and the bounded improvement reward ensure that d convergences to zero when $\sum_t \Gamma_t = 1$ and $\sum_t \Gamma_t^2 < \infty$. Therefore, our proposed algorithm is able to convergence to the optimal Q-value and the efficient multi-driver dispatching policy can be obtained.

IV. EXPERIMENT RESULTS

A. EXPERIMENT SETTING

1) SIMULATOR DESIGN

Generally, simulators are built to research the traffic problems [21]–[23]. In our experiment, we set up a simulator to train and test the performance of our proposed algorithm. To prove the effectiveness of our simulator, the simulator parameters are calibrated with the real world history data. Consequently, the traffic behaviors of our simulator are same as the real world traffic behaviors, and this is proved in our previous work [24].

In the simulator, the given area is spilt into many grids and one day is divided into 144 time intervals (each time interval is $t_\triangle = 10$ *mins*). During each time interval, the orders are generated according to the real world data firstly. Then,

TABLE 2. Performance comparison of comparative methods in terms of order response rate.

Vehicle initialization	100%	90%	70%
Methods	Order response rate	Order response rate	Order response rate
Simulation	62.57%	59.00%	52.57%
Random	65.58%	62.57%	55.14%
Multi-agent DDQN	77.00%	72.71%	62.14%
QRewriter-DDQN	**80.86%**	**75.71%**	**63.43%**

the vehicle distribution is generated according to the vehicle state (start working and stop working), which is estimated with Maximum Likelihood (ML) method. Afterwards, all idle vehicles are dispatched to different grids according to QRewriter-DDQN algorithm. Lastly, matching available vehicles and orders (order dispatching).

2) DATA DESCRIPTION

Our real world data is provided by DiDi Chuxing, and the data includes the vehicle data and order data in the core area of Haikou. In detail, the vehicle data is composed of unique vehicle ID, location latitude, location longitude and vehicle state (start working or stop working). Order data is made up of unique order ID, pick-up location (latitude and longitude), drop-off location (latitude and longitude) and pick-up time. Furthermore, the real world data is for three consecutive weeks, the first two weeks' data (training week) and the third week's data (testing week) are used to train the proposed QRwriter-DDQN algorithm and test the performance of QRewriter-DDQN algorithm, respectively.

3) EVALUATION METRIC

The goal of the proposed QRewriter-DDQN algorithm is to serve more orders with dispatching available vehicles to the grids with high demand. Consequently, the evaluation metric is order response rate, which is defined as

$$\frac{N_{so}}{N_{to}} \times 100\%, \tag{18}$$

where N_{so} and N_{to} represent the number of served orders of one day and the number of total orders of one day, respectively.

B. COMPARATIVE METHODS

In order to analysis the proposed QRwriter-DDQN algorithm performance, three comparative methods are used as baseline methods. Moreover, the order response rates of QRewriter-DDQN algorithm and baseline methods are the average values of three runs with random seeds. The detailed information of all methods is described as follow,

- **Simulation**: Simulation method means that vehicles serve orders without any dispatching policy.
- **Random**: In this method, the available vehicles in the same grid are uniformly dispatched to a random grid.
- **Multi-agent DDQN**: DDQN is a method of DRL, and multi-agent DDQN is a method of multi-agent DRL. Specially, multi-agent DDQN/DRL indicates that there are multiple agents in the DDQN/DRL model. For multi-agent DDQN, there are three layers in the neural network. The three layers owns 128, 64, 32 neurons, respectively. ReLU [25] is employed as the activation function. For ϵ-greedy action policy, the ϵ is set as 0.3. For memory replay, the sizes of memory replay and random mini-batch are 10000 and 2000, respectively. Besides, RMSProp algorithm [26] is used to update the parameters of evaluation Q-network. The reward discount factor and learning rate are set as 0.9 and 0.01, respectively.
- **QRewriter-DDQN**: The parameter setting of DDQN module is same as the multi-agent DDQN method. The reward discount factor and learning rate of QRewriter module are also set as 0.9 and 0.01, respectively. Besides, the influence of ϵ value on QRewriter-DDQN algorithm performance will be discussed in the Experiment Analysis section.

Remark: The QRewriter-DDQN algorithm will fall into local optimum and obtain poor performance when ϵ value too big. However, the ϵ value too small will cause the QRewriter-DDQN algorithm always choose random actions and obtain a terrible dispatching policy. Consequently, a proper ϵ value is vital for the QRewriter-DDQN algorithm performance.

C. EXPERIMENT ANALYSIS

To test the effectiveness and robustness of our proposed QRewriter-DDQN algorithm in the idle vehicle dispatching environment, three experiments are demonstrated with different vehicle initializations. The experiment results are shown in Table 2, and the results in the table are the average order response rate over three runs. For all methods, the initial vehicles are random generated with the random seeds and orders are generated according to the real world data provided by DiDi Chuxing.

As demonstrated in Table 2, the QRewriter-DDQN algorithm outperforms the state-of-the-art algorithm like multi-agent DDQN and other baselines in terms of order response rate. Although multi-agent DDQN method is capable of learning the complex dynamic variations between demand and supply, the state-action dimension of multi-agent DDQN is too large to obtain sterling and stable dispatching performance. The proposed QRewriter-DDQN algorithm is able to reduce the state-action space by using state aggregation and improve the dispatching performance. In QRewriter-DDQN, the small state-action dimension of the QRewriter module allows the module get Q-value from simple and efficient Q-table. Consequently, the QRewriter module elevates the DDQN module action policy with the stable and accurate Q-value. Furthermore, the multi-agent DDQN method, which was proposed in the previous work [24], corresponds to the DDQN module in our proposed QRewriter-DDQN

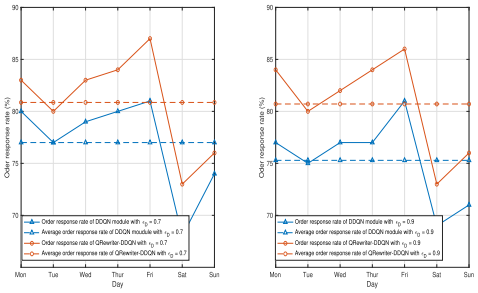

FIGURE 3. The effectiveness and robustness of the proposed QRewriter-DDQN algorithm under different DDQN module performances (different DDQN module's ϵ_D values) under 100% vehicle initialization.

algorithm, the comparison between multi-agent DDQN method and QRewriter-DDQN method indicates the advantages and effectiveness of QRewriter module. Finally, the improvement space of QRewriter-DDQN algorithm decreases as the initialized number of vehicles decreases. It is because that the available vehicles which can be dispatched are less when the initialized number of vehicles is small.

Fig. 3. indicates the effectiveness and robustness of the proposed QRewriter-DDQN algorithm with different DDQN module performances (different DDQN module's ϵ_D values) under 100% vehicle initialization. According to the figure, even under poor DDQN module performance, QRewriter-DDQN is capable of achieving the stable and sterling performance. In detail, the order response rate of $\epsilon_D = 0.7$ is higher about 2% than the order response rate of $\epsilon_D = 0.9$. However, the same excellent performance can be achieved when apply the QRewriter module to both of the two cases. It is because that the QRewriter module can effectively and efficiently abate the effect of the larger supply-demand dimension and improve the performance improvement space of DDQN module. Moreover, the QRewriter module employs the steady Q-table to realize the improvement policy. Consequently, QRewriter-DDQN algorithm is able to obtain stable results even in a poor environment (poor DDQN module performance).

Fig. 4. shows the order response rate of each day in the testing week with different QRewriter module's ϵ_I values under 100% vehicle initialization. The goal of this figure is to investigate how ϵ_I effects the performance of QRewriter-DDQN algorithm. Specially, ϵ_I determines how agents balance between exploration and exploitation. Clearly, $\epsilon_I = 0.8$ surpasses the performance with other ϵ_I values according to the figure. The lager ϵ_I values, like $\epsilon_I = 1$ and $\epsilon_I = 0.9$, let the available vehicles always take the current optimal actions. In the training steps, the current optimal actions are not the real optimal actions. Therefore, the QRewriter-DDQN algorithm will fall into the local optimum and obtain poor dispatching performance. The small ϵ_I values, like $\epsilon_I = 0.7$, let idle vehicles take random dispatching actions with larger probability and attenuate the dispatching performance. Consequently, the proper ϵ_I value is vital for the QRewriter-DDQN algorithm and we apply $\epsilon_I = 0.8$ in the QRewriter module to obtain better dispatching

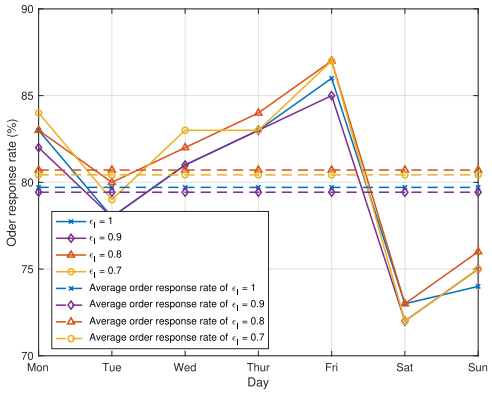

FIGURE 4. The relationship between QRewriter module's ϵ_I values and QRewriter-DDQN algorithm performance under 100% vehicle initialization.

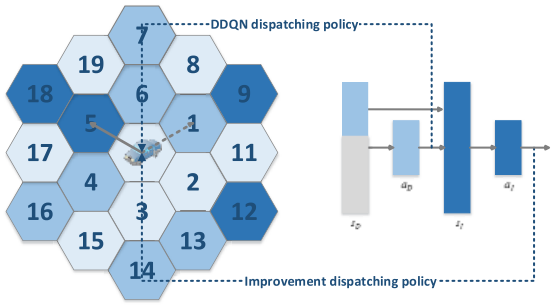

FIGURE 5. Case study: the dispatching process for a specific idle vehicle.

policy. This phenomenon is also confirmed by the insights provided in **Remark**.

D. CASE STUDY

Fig. 5. demonstrates the dispatching process for a specific idle vehicle. The given area is split into many grids, and the darker the color is, the more orders there are. As we can see, DDQN module wants dispatch the idle vehicle to a grid with less orders (grey dotted arrow). Nevertheless, the QRewriter module finds that there is an adjacent grid with high demand and dispatches the idle vehicle to the high demand grid (grey solid arrow). In detail, the DDQN module obtains state s_D from environment and chooses action a_D according to the DDQN dispatching policy. The DDQN module suggests the idle vehicle move to grid 1. However, the DDQN module only employs reward which produced by action a_D to update its parameters and the idle vehicle will not be really dispatched to grid 1. Afterwards, the QRewriter module gets state s_I from a_D and part of s_D. The improvement dispatching action a_I is taken with the improvement dispatching policy and the idle vehicle is dispatched to grid 5. At last, the QRewriter module uses the reward produced by action a_I to update its own parameters.

V. CONCLUSION

In this article, we proposed the QRewriter-DDQN algorithm to dispatch available vehicles in ahead to the locations with high demand to serve more orders. The QRewriter-DDQN algorithm factorized into a DDQN module and a QRewriter

module, which were parameterized by neural networks and Q-table with RL methods, respectively. Particularly, DDQN module employed the KL distribution distance between available vehicles and orders as excitation to capture the complex dynamic variations of supply-demand. Afterwards, the QRewriter module learned to improve the DDQN dispatching policy with the streamlined and effective Q-table in RL. Importantly, the higher performance improvement space of the DDQN dispatching policy can be obtained by aggregating QRewriter state into low-dimension meta state. At last, the experiment results demonstrated the significant improvement of QRewriter-DDQN in terms of order response rate.

REFERENCES

[1] Didi Chuxing. [Online]. Available: http://www.didichuxing.com/en/
[2] Uber. [Online]. Available: https://www.uber.com/
[3] G. Xu, J. Wang, G. Q. Huang, and C.-H. Chen, "Data-driven resilient fleet management for cloud asset-enabled urban flood control," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 6, pp. 1827–1838, Jun. 2018.
[4] M. Dicorato, G. Forte, M. Trovato, C. B. Munoz, and G. Coppola, "An integrated DC microgrid solution for electric vehicle fleet management," *IEEE Trans. Ind. Appl.*, vol. 55, no. 6, pp. 7347–7355, Nov. 2019.
[5] Q. Cui, Y. Wang, K.-C. Chen, W. Ni, I.-C. Lin, X. Tao, and P. Zhang, "Big data analytics and network calculus enabling intelligent management of autonomous vehicles in a smart city," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2021–2034, Apr. 2019.
[6] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takac, "Reinforcement learning for solving the vehicle routing problem," in *Proc. Neural Inf. Process. Syst. (NeurIPS)*, 2018, pp. 9839–9849.
[7] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," 2016, *arXiv:1611.09940*. [Online]. Available: http://arxiv.org/abs/1611.09940
[8] J. Jin, M. Zhou, W. Zhang, M. Li, Z. Guo, Z. Qin, Y. Jiao, X. Tang, C. Wang, J. Wang, G. Wu, and J. Ye, "CoRide: Joint order dispatching and fleet management for multi-scale ride-hailing platforms," 2019, *arXiv:1905.11353*. [Online]. Available: http://arxiv.org/abs/1905.11353
[9] 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," 2019, *arXiv:1911.11260*. [Online]. Available: http://arxiv.org/abs/1911.11260
[10] G. Guo and Y. Xu, "A deep reinforcement learning approach to ride-sharing vehicles dispatching in autonomous mobility-on-demand systems," *IEEE Intell. Transp. Syst. Mag.*, early access, Apr. 1, 2020, doi: 10.1109/MITS.2019.2962159.
[11] J. J. Q. Yu, W. Yu, and J. Gu, "Online vehicle routing with neural combinatorial optimization and deep reinforcement learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 10, pp. 3806–3817, Oct. 2019.
[12] T. Oda and C. Joe-Wong, "MOVI: A model-free approach to dynamic fleet management," in *Proc. IEEE INFOCOM-IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 2708–2716.
[13] A. O. Al-Abbasi, A. Ghosh, and V. Aggarwal, "DeepPool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 12, pp. 4714–4727, Dec. 2019.
[14] 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 *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 905–913.
[15] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2016, pp. 1995–2003.
[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, vol. 1. Cambridge, MA, USA: MIT Press, 1998.
[17] A. S. Mignona and R. L. A. Rochaa, "An adaptive implementation of ϵ-greedy in reinforcement learning," *Procedia Comput. Sci.*, vol. 109, pp. 1146–1151, 2017, doi: 10.1016/j.procs.2017.05.431.
[18] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*. [Online]. Available: http://arxiv.org/abs/1609.04747
[19] F. S. Melo, "Convergence of Q-learning: A simple proof," Inst. Syst. Robot., Tech. Rep., 2001, pp. 1–4.
[20] T. Jaakkola, M. I. Jordan, and S. P. Singh, "Convergence of stochastic iterative dynamic programming algorithms," in *Proc. Neural Inf. Process. Syst. (NeurIPS)*, 1994, pp. 703–710, 1994.
[21] X. Liang, X. Du, G. Wang, and Z. Han, "A deep reinforcement learning network for traffic light cycle control," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1243–1253, Feb. 2019.
[22] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2016, pp. 1928–1937.
[23] K. T. Seow, N. H. Dang, and D. Lee, "A collaborative multiagent taxi-dispatch system," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 3, pp. 607–616, Jul. 2010.
[24] W. Zhang, Q. Wang, J. Li, and D. Shi, "Dynamic vehicle dispatching based on minimum fleet a deep reinforcement learning method," in *Proc. Neural Inf. Process. Syst. (NeurIPS, Deep Reinforcement Learn. Workshop)*, 2019.
[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[26] N. Ketkar, "Stochastic gradient descent," in *Deep Learning With Python*. Berkeley, CA, USA: Apress, 2017, doi: 10.1007/978-1-4842-2766-4_8.

WENQI ZHANG received the B.S. degree from the Beijing University of Posts and Telecommunications, China, in 2016. She is currently pursuing the Ph.D. degree with the National Engineering Laboratory for Mobile Network Security, Wireless Technology Innovation Institute, Beijing University of Posts and Telecommunications. Her research interests include dynamic fleet management, reinforcement learning, and UAV-assisted networks.

QIANG WANG (Member, IEEE) received the B.S. degree in material science and engineering from the Hefei University of Technology, Hefei, China, in 2003, and the Ph.D. degree in communication engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 2008. Since 2008, he has been working with the School of Information and Communication Engineering, where he is currently an Associate Professor. He has participated in many national projects, such as NSFC, 863 and so on. His research interests include information theory, wireless communications, VLSI, and statistical inference.

JINGJING LI is currently pursuing the master's degree with the National Engineering Laboratory for Mobile Network Security and the Wireless Technology Innovation Institute, Beijing University of Posts and Telecommunications. Her research interests include rebalancing bike sharing systems and reinforcement learning.

CHEN XU is currently pursuing the master's degree with the National Engineering Laboratory for Mobile Networks Security and Wireless Technology Innovation Institute, Beijing University of Posts and Telecommunications. His research interests include estimated time of arrival and reinforcement learning.