Efficient Large-Scale Fleet Management via Multi-Agent Deep Reinforcement Learning

Kaixiang Lin
Michigan State University
linkaixi@msu.edu

Renyu Zhao, Zhe Xu
Didi Chuxing
{zhaorenyu,xuzhejesse}@didichuxing.com

Jiayu Zhou
Michigan State University
jiayuz@msu.edu

ABSTRACT

Large-scale online ride-sharing platforms have substantially transformed our lives by reallocating transportation resources to alleviate traffic congestion and promote transportation efficiency. An efficient fleet management strategy not only can significantly improve the utilization of transportation resources but also increase the revenue and customer satisfaction. It is a challenging task to design an effective fleet management strategy that can adapt to an environment involving complex dynamics between demand and supply. Existing studies usually work on a simplified problem setting that can hardly capture the complicated stochastic demand-supply variations in high-dimensional space. In this paper we propose to tackle the large-scale fleet management problem using reinforcement learning, and propose a contextual multi-agent reinforcement learning framework including two concrete algorithms, namely contextual deep Q-learning and contextual multi-agent actor-critic, to achieve explicit coordination among a large number of agents adaptive to different contexts. We show significant improvements of the proposed framework over state-of-the-art approaches through extensive empirical studies[1].

CCS CONCEPTS

• **Computing methodologies → Multi-agent reinforcement learning**; *Reinforcement learning*;

KEYWORDS

Multi-agent Reinforcement Learning; Deep Reinforcement Learning; Fleet Management

ACM Reference Format:
Kaixiang Lin, Renyu Zhao, Zhe Xu, and Jiayu Zhou. 2018. Efficient Large-Scale Fleet Management via Multi-Agent Deep Reinforcement Learning. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, London, UK, 10 pages. https://doi.org/10.1145/3219819.3219993

[1]Code is available at https://github.com/illidanlab/Simulator

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
KDD '18, August 19–23, 2018, London, United Kingdom
© 2018 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.
ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3219993

1 INTRODUCTION

Large-scale online ride-sharing platforms such as Uber [27] and Didi Chuxing [6] have transformed the way people travel, live and socialize. By leveraging the advances in and wide adoption of information technologies such as cellular networks and global positioning systems, the ride-sharing platforms redistribute underutilized vehicles on the roads to passengers in need of transportation. The optimization of transportation resources greatly alleviated traffic congestion and calibrated the once significant gap between transport demand and supply [13].

One key challenge in ride-sharing platforms is to balance the demands and supplies, i.e., orders of the passengers and drivers available for picking up orders. In large cities, although millions of ride-sharing orders are served everyday, an enormous number of passengers requests remain unserviced due to the lack of available drivers nearby. On the other hand, there are plenty of available drivers looking for orders in other locations. If the available drivers were directed to locations with high demand, it will significantly increase the number of orders being served, and thus simultaneously benefit all aspects of the society: utility of transportation capacity will be improved, income of drivers and satisfaction of passengers will be increased, and market share and revenue of the company will be expanded. *fleet management* is a key technical component to balance the differences between demand and supply, by reallocating available vehicles ahead of time, to achieve high efficiency in serving future demand.

Even though rich historical demand and supply data are available, using the data to seek an optimal allocation policy is not an easy task. One major issue is that changes in an allocation policy will impact future demand-supply, and it is hard for supervised learning approaches to capture and model these real-time changes. On the other hand, the reinforcement learning (RL) [24], which learns a policy by interacting with a complicated environment, has been naturally adopted to tackle the fleet management problem [10, 11, 28]. However, the high-dimensional and complicated dynamics between demand and supply can hardly be modeled accurately by traditional RL approaches.

Recent years witnessed tremendous success in deep reinforcement learning (DRL) in modeling intellectual challenging decision-making problems [17, 23] that were previously intractable. In the light of such advances, in this paper we propose a novel DRL approach to learn highly efficient allocation policies for fleet management. There are significant technical challenges when modeling fleet management using DRL:

1) *Feasibility of problem setting.* The RL framework is reward-driven, meaning that a sequence of *actions* from the policy is evaluated solely by the *reward* signal from environment [1]. The definitions

of agent, reward and action space are essential for RL. If we model the allocation policy using a centralized agent, the action space can be prohibitively large since an action needs to decide the number of available vehicles to reposition from each location to its nearby locations. Also, the policy is subject to a feasibility constraint enforcing that the number of repositioned vehicles needs to be no larger than the current number of available vehicles. To the best of our knowledge, this high-dimensional exact-constrain satisfaction policy optimization is not computationally tractable in DRL: applying it in a very small-scale problem could already incur high computational costs [20].

2) *Large-scale Agents.* One alternative approach is to instead use a multi-agent DRL setting, where each available vehicle is considered as an agent. The multi-agent recipe indeed alleviates the curse of dimensionality of action space. However, such setting creates thousands of agents interacting with the environment at each time. Training a large number of agents using DRL is again challenging: the environment for each agent is non-stationary since other agents are learning and affecting the environment at same the time. Most of existing studies [9, 14, 25] allow coordination among only a small set of agents due to high computational costs.

3) *Coordinations and Context Dependence of Action Space.* Facilitating coordination among large-scale agents remains a challenging task. Since each agent typically learns its own policy or action-value function that are changing over time, it is difficult to coordinate agents for a large number of agents. Moreover, the action space is dynamic changing over time since agents are navigating to different locations and the number of feasible actions depends on the geographic context of the location.

In this paper, we propose a contextual multi-agent DRL framework to resolve the aforementioned challenges. Our major contributions are listed as follows:

- We propose an efficient multi-agent DRL setting for large-scale fleet management problem by a proper design of agent, reward and state.
- We propose contextual multi-agent reinforcement learning framework in which two concrete algorithms: *contextual multi-agent actor-critic* (cA2C) and *contextual deep Q-learning* (cDQN) are developed. For the first time in multi-agent DRL, the contextual algorithms can not only achieve efficient explicit coordination among thousands of learning agents at each time, but also adapt to dynamically changing action spaces.
- In order to train and evaluate the RL algorithm, we developed a simulator that simulates real-world traffic activities perfectly after calibrating the simulator using real historical data provided by Didi Chuxing [6].
- Last but not least, the proposed contextual algorithms significantly outperform the state-of-the-art methods in multi-agent DRL with a much less number of repositions needed.

The rest of paper is organized as follows. We first give a literature review on the related work in Sec 2. Then the problem statement is elaborated in Sec 3 and the simulation platform we built for training and evaluation are introduced in Sec 5. The methodology is described in Sec 4. Quantitative and qualitative results are presented in Sec 6. Finally, we conclude our work in Sec 7.

2 RELATED WORKS

Intelligent Transportation System. Advances in machine learning and traffic data analytics lead to widespread applications of machine learning techniques to tackle challenging traffic problems. One trending direction is to incorporate reinforcement learning algorithms in complicated traffic management problems. There are many previous studies that have demonstrated the possibility and benefits of reinforcement learning. Our work has close connections to these studies in terms of problem setting, methodology and evaluation. Among the traffic applications that are closely related to our work, such as taxi dispatch systems or traffic light control algorithms, multi-agent RL has been explored to model the intricate nature of these traffic activities [2, 15, 21]. The promising results motivated us to use multi-agent modeling in the fleet management problem. In [10], an adaptive dynamic programming approach was proposed to model stochastic dynamic resource allocation. It estimates the returns of future states using a piecewise linear function and delivers actions (assigning orders to vehicles, reallocate available vehicles) given states and one step future states values, by solving an integer programming problem. In [11], the authors further extended the approach to the situations that an action can span across multiple time periods. These methods are hard to be directly utilized in the real-world setting where orders can be served through the vehicles located in multiple nearby locations.

Multi-agent reinforcement learning. Another relevant research topic is multi-agent reinforcement learning [5] where a group of agents share the same environment, in which they receive rewards and take actions. [26] compared and contrasted independent Q-learning and a cooperative counterpart in different settings, and empirically showed that the learning speed can benefit from cooperative Q-learning. Independent Q-learning is extended into DRL in [25], where two agents are cooperating or competing with each other only through the reward. In [9], the authors proposed a counterfactual multi-agent policy gradient method that uses a centralized advantage to estimate whether the action of one agent would improve the global reward, and decentralized actors to optimize the agent policy. Ryan *et al.* also utilized the framework of decentralized execution and centralized training to develop multi-agent multi-agent actor-critic algorithm that can coordinate agents in mixed cooperative-competitive environments [14]. However, none of these methods were applied when there are a large number of agents due to the communication cost among agents. Recently, few works [29, 30] scaled DRL methods to a large number of agents, while it is not applicable to apply these methods to complex real applications such as fleet management. In [18, 19], the authors studied large-scale multi-agent planning for fleet management with explicitly modeling the expected counts of agents.

Deep reinforcement learning. DRL utilizes neural network function approximations and are shown to have largely improved the performance over challenging applications [17, 22, 23]. Many sophisticated DRL algorithms such as DQN [17], A3C [16] were demonstrated to be effective in the tasks in which we have a clear understanding of rules and have easy access to millions of samples, such as video games [3, 4]. However, DRL approaches are rarely seen to be applied in complicated real-world applications, especially in those with high-dimensional and non-stationary action

space, lack of well-defined reward function, and in need of coordination among a large number of agents. In this paper, we show that through careful reformulation, the DRL can be applied to tackle the fleet management problem.

3 PROBLEM STATEMENT

In this paper, we consider the problem of managing a large set of available homogeneous vehicles for online ride-sharing platforms. The goal of the management is to maximize the gross merchandise volume (GMV: the value of all the orders served) of the platform by repositioning available vehicles to the locations with larger demand-supply gap than the current one. This problem belongs to a variant of the classical fleet management problem [8]. A spatial-temporal illustration of the problem is available in Figure 1. In this example, we use *hexagonal-grid world* to represent the map and split the duration of one day into $T = 144$ time intervals (one for 10 minutes). At each time interval, the orders emerge stochastically in each grid and are served by the available vehicles in the same grid or six nearby grids. The goal of fleet management here is to decide how many available vehicles to relocate from each grid to its neighbors in ahead of time, so that most orders can be served.

To tackle this problem, we propose to formulate the problem using *multi-agent reinforcement learning* [5]. In this formulation, we use a set of homogeneous agents with small action spaces, and split the global reward into each grid. This will lead to a much more efficient learning procedure than the single agent setting, due to the simplified action dimension and the explicit credit assignment based on split reward. Formally, we model the fleet management problem as a Markov game G for N agents, which is defined by a tuple $G = (N, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $N, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma$ are the number of agents, sets of states, joint action space, transition probability functions, reward functions, and a discount factor respectively. The definitions are given as follows:

- **Agent**: We consider an available vehicle (or equivalently an idle driver) as an agent, and the vehicles in the same spatial-temporal node are homogeneous, i.e., the vehicles located at the same region at the same time interval are considered as same agents (where agents have the same policy). Although the number of unique heterogeneous agents is always N, the number of agents N_t is changing over time.
- **State** $\mathbf{s}_t \in \mathcal{S}$: We maintain a global state $\mathbf{s}_t$ at each time t, considering the spatial distributions of available vehicles and orders (i.e. the number of available vehicles and orders in each grid) and current time t (using one-hot encoding). The state of an agent i, $\mathbf{s}_t^i$, is defined as the identification of the grid it located and the shared global state i.e. $\mathbf{s}_t^i = [\mathbf{s}_t, \mathbf{g}_j] \in R^{N\times 3+T}$, where $\mathbf{g}_j$ is the one-hot encoding of the grid ID. We note that agents located at same grid have the same state $\mathbf{s}_t^i$.
- **Action** $a_t \in \mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_{N_t}$: a *joint action* $\mathbf{a}_t = \{a_t^i\}_1^{N_t}$ instructing the allocation strategy of all available vehicles at time t. The action space $\mathcal{A}_i$ of an individual agent specifies where the agent is able to arrive at the next time, which gives a set of seven discrete actions denoted by $\{k\}_{k=1}^7$. The first six discrete actions indicate allocating the agent to one of its six neighboring grids, respectively. The last discrete action $a_t^i = 7$ means staying in the current grid. For example, the action $a_0^1 = 2$ means to relocate the 1st agent from the current grid to the second nearby grid at time 0, as shown in Figure 1. For a concise presentation, we also use $a_t^i \triangleq [\mathbf{g}_0, \mathbf{g}_1]$ to represent agent i moving from grid $\mathbf{g}_0$ to $\mathbf{g}_1$. Furthermore, the action space of agents depends on their locations. The agents located at corner grids have a smaller action space. We also assume that the action is deterministic: if $a_t^i \triangleq [\mathbf{g}_0, \mathbf{g}_1]$, then agent i will arrive at the grid $\mathbf{g}_1$ at time $t+1$.

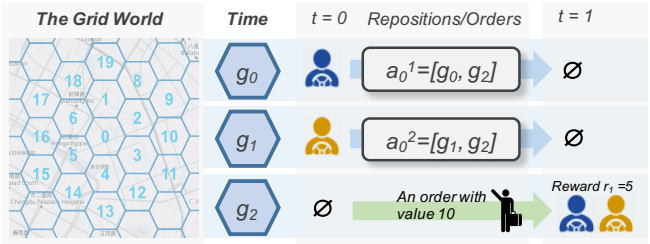


Figure 1: The grid world system and a spatial-temporal illustration of the problem setting.

- **Reward function** $\mathcal{R}_i \in \mathcal{R} = \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: Each agent is associated with a reward function $\mathcal{R}_i$ and all agents in the same location have the same reward function. The i-th agent attempts to maximize its own expected discounted return: $\mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}^i\right]$. The individual reward r_t^i for the i-th agent associated with the action $\mathbf{a}_t^i$ is defined as the averaged revenue of all agents arriving at the same grid as the i-th agent at time $t+1$. Since the individual rewards at same time and the same location are same, we denote this reward of agents at time t and grid $\mathbf{g}_j$ as $r_t(\mathbf{g}_j)$. Such design of rewards aims at avoiding greedy actions that send too many agents to the location with high value of orders, and aligning the maximization of each agent's return with the maximization of GMV (value of all served orders in one day). Its effectiveness is empirically verified in Sec 6.
- **State transition probability** $p(\mathbf{s}_{t+1}|\mathbf{s}_t, a_t) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$: It gives the probability of transiting to $\mathbf{s}_{t+1}$ given a joint action $\mathbf{a}_t$ is taken in the current state $\mathbf{s}_t$. Notice that although the action is deterministic, new vehicles and orders will be available at different grids each time, and existing vehicles will become off-line via a random process.

To be more concrete, we give an example based on the above problem setting in Figure 1. At time $t = 0$, agent 1 is repositioned from $\mathbf{g}_0$ to $\mathbf{g}_2$ by action a_0^1, and agent 2 is also repositioned from $\mathbf{g}_1$ to $\mathbf{g}_2$ by action a_0^2. At time $t = 1$, two agents arrive at $\mathbf{g}_2$, and a new order with value 10 also emerges at same grid. Therefore, the reward r_1 for both a_0^1 and a_0^2 is the averaged value received by agents at $\mathbf{g}_2$, which is $10/2 = 5$.

4 CONTEXTUAL MULTI-AGENT REINFORCEMENT LEARNING

In this section, we present two novel contextual multi-agent RL approaches: contextual multi-agent actor-critic (cA2C) and contextual DQN (cDQN) algorithm. We first briefly introduce the basic multi-agent RL method.

4.1 Independent DQN

Independent DQN [25] combines independent Q-learning [26] and DQN [17]. A straightforward extension of independent DQN from

small scale to a large number of agents, is to share network parameters and distinguish agents with their IDs [30]. The network parameters can be updated by minimizing the following loss function, with respect to the transitions collected from all agents:

$$\mathop{\mathbb{E}}_{\mathbf{s}_t^i, a_t^i, s_{t+1}^i, r_{t+1}^i}\left[Q(\mathbf{s}_t^i, a_t^i; \theta) - \left(r_{t+1}^i + \gamma \max_{a_{t+1}^i} Q(\mathbf{s}_{t+1}^i, a_{t+1}^i; \theta')\right)\right]^2, \tag{1}$$

where θ' includes parameters of the target Q network updated periodically, and θ includes parameters of behavior Q network outputting the action value for ϵ-greedy policy, same as the algorithm described in [17]. This method could work reasonably well after extensive tuning but it suffers from high variance in performance, and it also repositions too many vehicles. Moreover, coordination among massive agents is hard to achieve since each unique agent executes its action independently based on its action values.

4.2 Contextual DQN

Since we assume that the location transition of an agent after the allocation action is deterministic, the actions that lead the agents to the same grid should have the same action value. In this case, the number of unique action-values for all agents should be equal to the number of grids N. Formally, for any agent i where $\mathbf{s}_t^i = [\mathbf{s}_t, \mathbf{g}_i]$, $a_t^i \triangleq [\mathbf{g}_i, \mathbf{g}_d]$ and $\mathbf{g}_i \in Ner(\mathbf{g}_d)$, the following holds:

$$Q(\mathbf{s}_t^i, a_t^i) = Q(\mathbf{s}_t, \mathbf{g}_d) \tag{2}$$

Hence, at each time step, we only need N unique action-values $(Q(\mathbf{s}_t, \mathbf{g}_j), \forall j = 1, \ldots, N)$ and the optimization of Eq (1) can be replaced by minimizing the following mean-squared loss:

$$\left[Q(\mathbf{s}_t, \mathbf{g}_d; \theta) - \left(r_{t+1}(\mathbf{g}_d) + \gamma \max_{\mathbf{g}_p \in \mathrm{Ner}(\mathbf{g}_d)} Q(\mathbf{s}_{t+1}, \mathbf{g}_p; \theta')\right)\right]^2. \tag{3}$$

This accelerates the learning procedure since the output dimension of the action value function is reduced from $\mathbb{R}^{|\mathbf{s}_t|} \to \mathbb{R}^7$ to $\mathbb{R}^{|\mathbf{s}_t|} \to \mathbb{R}$. Furthermore, we can build a centralized action-value table at each time for all agents, which can serve as the foundation for coordinating the actions of agents.

Geographic context. In hexagonal grids systems, border grids and grids surrounded by infeasible grids (e.g., a lake) have reduced action dimensions. To accommodate this, for each grid we compute a *geographic context* $\mathbf{G}_{\mathbf{g}_j} \in \mathbb{R}^7$, which is a binary vector that filters out invalid actions for agents in grid $\mathbf{g}_j$. The kth element of vector $\mathbf{G}_{\mathbf{g}_j}$ represents the validity of moving toward kth direction from the grid $\mathbf{g}_j$. Denote $\mathbf{g}_d$ as the grid corresponds to the kth direction of grid $\mathbf{g}_j$, the value of the kth element of $\mathbf{G}_{\mathbf{g}_j}$ is given by:

$$[\mathbf{G}_{t,\mathbf{g}_j}]_k = \begin{cases} 1, & \text{if } \mathbf{g}_d \text{ is valid grid,} \\ 0, & \text{otherwise,} \end{cases} \tag{4}$$

where $k = 0, \ldots, 6$ and last dimension of the vector represents direction staying in same grid, which is always 1.

Collaborative context. To avoid the situation that agents are moving in conflict directions (i.e., agents are repositioned from grid $\mathbf{g}_1$ to $\mathbf{g}_2$ and $\mathbf{g}_2$ to $\mathbf{g}_1$ at the same time.), we provide a *collaborative context* $\mathbf{C}_{t,\mathbf{g}_j} \in \mathbb{R}^7$ for each grid $\mathbf{g}_j$ at each time. Based on the centralized action values $Q(\mathbf{s}_t, \mathbf{g}_j)$, we restrict the valid actions such that agents at the grid $\mathbf{g}_j$ are navigating to the neighboring grids with higher action values or staying unmoved. Therefore, the binary vector $\mathbf{C}_{t,\mathbf{g}_j}$ eliminates actions to grids with lower action values than the action staying unmoved. Formally, the kth element of vector $\mathbf{C}_{t,\mathbf{g}_j}$ that corresponds to action value $Q(\mathbf{s}_t, \mathbf{g}_i)$ is defined as follows:

$$[\mathbf{C}_{t,\mathbf{g}_j}]_k = \begin{cases} 1, & \text{if } Q(\mathbf{s}_t, \mathbf{g}_i) >= Q(\mathbf{s}_t, \mathbf{g}_j), \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

After computing both collaborative and geographic context, the ϵ-greedy policy is then performed based on the action values survived from the two contexts. Suppose the original action values of agent i at time t is $\mathbf{Q}(\mathbf{s}_t^i) \in \mathbb{R}^7_{\geq 0}$, given state $\mathbf{s}_t^i$, the valid action values after applying contexts is as follows:

$$\mathbf{q}(\mathbf{s}_t^i) = \mathbf{Q}(\mathbf{s}_t^i) * \mathbf{C}_{t,\mathbf{g}_j} * \mathbf{G}_{t,\mathbf{g}_j}. \tag{6}$$

The coordination is enabled because the action values of different agents lead to the same location are restricted to be same so that they can be compared, which is impossible in independent DQN. This method requires that action values are always non-negative, which will always hold because agents always receive nonnegative rewards. The algorithm of cDQN is elaborated in Alg 2.

Algorithm 1 ϵ-greedy policy for cDQN

Require: Global state $\mathbf{s}_t$
- Compute centralized action value $Q(\mathbf{s}_t, \mathbf{g}_j), \forall j = 1, \ldots, N$
- **for** $i = 1$ to N_t **do**
  - Compute action values $\mathbf{Q}^i$ by Eq (2), where $(\mathbf{Q}^i)_k = Q(\mathbf{s}_t^i, a_t^i = k)$.
  - Compute contexts $\mathbf{C}_{t,\mathbf{g}_j}$ and $\mathbf{G}_{t,\mathbf{g}_j}$ for agent i.
  - Compute valid action values $\mathbf{q}_t^i = \mathbf{Q}_t^i * \mathbf{C}_{t,\mathbf{g}_j} * \mathbf{G}_{t,\mathbf{g}_j}$.
  - $a_t^i = \mathrm{argmax}_k \mathbf{q}_t^i$ with probability $1 - \epsilon$ otherwise choose an action randomly from the valid actions.
- **end for**
- **return** Joint action $\mathbf{a}_t = \{a_t^i\}_1^{N_t}$.

Algorithm 2 Contextual Deep Q-learning (cDQN)

- Initialize replay memory D to capacity M
- Initialize action-value function with random weights θ or pre-trained parameters.
- **for** $m = 1$ to *max-iterations* **do**
  - Reset the environment and reach the initial state $\mathbf{s}_0$.
  - **for** $t = 0$ to T **do**
    - Sample joint action $\mathbf{a}_t$ using Alg. 1, given $\mathbf{s}_t$.
    - Execute a_t in simulator and observe reward $\mathbf{r}_t$ and next state $\mathbf{s}_{t+1}$
    - Store the transitions of all agents $(\mathbf{s}_t^i, a_t^i, r_t^i, \mathbf{s}_{t+1}^i, \forall i = 1, \ldots, N_t)$ in D.
  - **end for**
  - **for** $k = 1$ to M_1 **do**
    - Sample a batch of transitions $(\mathbf{s}_t^i, a_t^i, r_t^i, \mathbf{s}_{t+1}^i)$ from D, where t can be different in one batch.
    - Compute target $y_t^i = r_t^i + \gamma * \max_{a_{t+1}^i} Q(\mathbf{s}_{t+1}^i, a_{t+1}^i; \theta')$.
    - Update Q-network as $\theta \leftarrow \theta + \nabla_\theta (y_t^i - Q(\mathbf{s}_t^i, a_t^i; \theta))^2$,
  - **end for**
- **end for**

4.3 Contextual Actor-Critic

We now present the contextual multi-agent actor-critic (cA2C) algorithm, which is a multi-agent policy gradient algorithm that tailors its policy to adapt to the dynamically changing action space. Meanwhile, it achieves not only a more stable performance but also a much more efficient learning procedure in a non-stationary environment. There are two main ideas in the design of cA2C: 1) A centralized value function shared by all agents with an expected update; 2) Policy context embedding that establishes explicit coordination among agents, enables faster training and enjoys the flexibility of regulating policy to different action spaces. The centralized state-value function is learned by minimizing the following loss function derived from Bellman equation:

$$L(\theta_v) = (V_{\theta_v}(\mathbf{s}_t^i) - V_{\text{target}}(\mathbf{s}_{t+1}; \theta_v', \pi))^2, \tag{7}$$

$$V_{\text{target}}(\mathbf{s}_{t+1}; \theta_v', \pi) = \sum\nolimits_{a_t^i} \pi(a_t^i|\mathbf{s}_t^i)(r_{t+1}^i + \gamma V_{\theta_v'}(\mathbf{s}_{t+1}^i)). \tag{8}$$

where we use θ_v to denote the parameters of the value network and θ_v' to denote the target value network. Since agents staying unmoved at the same time are treated homogeneous and share the same internal state, there are N unique agent states, and thus N unique state-values ($V(\mathbf{s}_t, \mathbf{g}_j), \forall j = 1, ..., N$) at each time. The state-value output is denoted by $\mathbf{v}_t \in \mathbb{R}^N$, where each element $(\mathbf{v}_t)_j = V(\mathbf{s}_t, \mathbf{g}_j)$ is the expected return received by agent arriving at grid $\mathbf{g}_j$ on time t. In order to stabilize learning of the value function, we fix a target value network parameterized by θ_v', which is updated at the end of each episode. Note that the expected update in Eq (7) and training actor/critic in an offline fashion are different from the updates in n-step actor-critic online training using TD error [16], whereas the expected updates and training paradigm are found to be more stable and sample-efficient. Furthermore, efficient coordination among multiple agents can be established upon this centralized value network.

Policy Context Embedding. Coordination is achieved by masking available action space based on the context. At each time step, the geographic context is given by Eq (4) and the collaborative context is computed according to the value network output:

$$[\mathbf{C}_{t,\mathbf{g}_j}]_k = \begin{cases} 1, & if\ V(\mathbf{s}_t, \mathbf{g}_i) >= V(\mathbf{s}_t, \mathbf{g}_j), \\ 0, & \text{otherwise}, \end{cases} \tag{9}$$

where the kth element of vector $\mathbf{C}_{t,\mathbf{g}_j}$ corresponds to the probability of the kth action $\pi(a_t^i = k|\mathbf{s}_t^i)$. Let $\mathbf{P}(\mathbf{s}_t^i) \in \mathbb{R}_{>0}^7$ denote the original logits from the policy network output for the ith agent conditioned on state $\mathbf{s}_t^i$. Let $\mathbf{q}_{\text{valid}}(\mathbf{s}_t^i) = \mathbf{P}(\mathbf{s}_t^i) * \mathbf{C}_{t,\mathbf{g}_j} * \mathbf{G}_{\mathbf{g}_j}$ denote the valid logits considering both geographic and collaborative context for agent i at grid $\mathbf{g}_j$, where $*$ denotes an element-wise multiplication. In order to achieve effective masking, we restrict the output logits $\mathbf{P}(\mathbf{s}_t^i)$ to be positive. The probability of valid actions for all agents in the grid $\mathbf{g}_j$ are given by:

$$\pi_{\theta_p}(a_t^i = k|\mathbf{s}_t^i) = [\mathbf{q}_{valid}(\mathbf{s}_t^i)]_k = \frac{[\mathbf{q}_{valid}(\mathbf{s}_t^i)]_k}{\|\mathbf{q}_{valid}(\mathbf{s}_t^i)\|_1}. \tag{10}$$

The gradient of policy can then be written as:

$$\nabla_{\theta_p} J(\theta_p) = \nabla_{\theta_p} \log \pi_{\theta_p}(a_t^i|\mathbf{s}_t^i) A(\mathbf{s}_t^i, a_t^i), \tag{11}$$

Algorithm 3 Contextual Multi-agent Actor-Critic Policy forward

Require: The global state $\mathbf{s}_t$.
1: Compute centralized state-value $\mathbf{v}_t$
2: **for** i = 1 to N_t **do**
3: Compute contexts $\mathbf{C}_{t,\mathbf{g}_j}$ and $\mathbf{G}_{t,\mathbf{g}_j}$ for agent i.
4: Compute action probability distribution $\mathbf{q}_{valid}(\mathbf{s}_t^i)$ for agent i in grid $\mathbf{g}_j$ as Eq (10).
5: Sample action for agent i in grid $\mathbf{g}_j$ based on action probability $\mathbf{p}^i$.
6: **end for**
7: **return** Joint action $\mathbf{a}_t = \{a_t^i\}_1^{N_t}$.

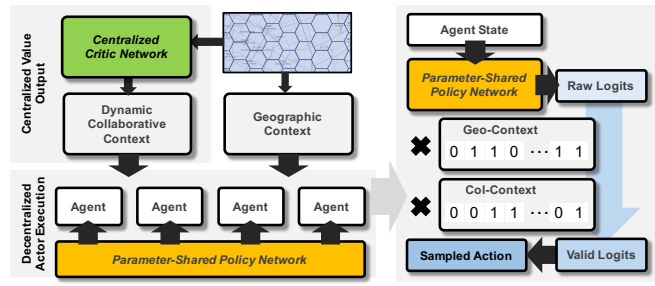


Figure 2: Illustration of contextual multi-agent actor-critic. The left part shows the coordination of decentralized execution based on the output of centralized value network. The right part illustrates embedding context to policy network.

where θ_p denotes the parameters of policy network and the advantage $A(\mathbf{s}_t^i, a_t^i)$ is computed as follows:

$$A(\mathbf{s}_t^i, a_t^i) = r_{t+1}^i + \gamma V_{\theta_v'}(\mathbf{s}_{t+1}^i) - V_{\theta_v}(\mathbf{s}_t^i). \tag{12}$$

The detailed description of cA2C is summarized in Alg 4.

5 SIMULATOR DESIGN

Unlike the standard supervised learning problems where the data is stationary to the learning algorithms and can be evaluated by the training-testing paradigm, the interactive nature of RL introduces intricate difficulties on training and evaluation. One common solution in traffic studies is to build simulators for the environment [15, 21, 28]. In this section, we introduce a simulator design that models the generation of orders, procedure of assigning orders and key driver behaviors such as distributions across the city, on-line/off-line status control in the real world. The simulator serves as the training environment for RL algorithms, as well as their evaluation. More importantly, our simulator allows us to calibrate the key performance index with the historical data collected from a fleet management system, and thus the policies learned are well aligned with real-world traffics.

The Data Description The data provided by Didi Chuxing includes orders and trajectories of vehicles in the center area of a City (Chengdu) in four consecutive weeks. The city is covered by a hexagonal grids world consisting of 504 grids. The order information includes order price, origin, destination and duration. The trajectories contain the positions (latitude and longitude) and status (on-line, off-line, on-service) of all vehicles every few seconds.

Algorithm 4 Contextual Multi-agent Actor-Critic Algorithm for N agents

Initialization:
Initialize the value network with fixed value table.
for $m = 1$ to max-iterations **do**
  Reset environment, get initial state $\mathbf{s}_0$.
  Stage 1: Collecting experience
  for $t = 0$ to T **do**
    Sample actions $\mathbf{a}_t$ according to Alg 3, given $\mathbf{s}_t$.
    Execute $\mathbf{a}_t$ in simulator and observe reward r_t and next state $\mathbf{s}_{t+1}$.
    Compute value network target as Eq (8) and advantage as Eq (12) for policy network and store the transitions.
  end for
  Stage 2: Updating parameters
  for $m_1 = 1$ to M_1 **do**
    Sample a batch of experience: $\mathbf{s}_t^i, V_{target}(\mathbf{s}_t^i; \theta_v', \pi)$
    Update value network by minimizing the value loss Eq (7) over the batch.
  end for
  for $m_2 = 1$ to M_2 **do**
    Sample a batch of experience: $\mathbf{s}_t^i, a_t^i, A(\mathbf{s}_t^i, a_t^i), \mathbf{C}_{t,g_j}, \mathbf{G}_{g_j}$, where t can be different one batch.
    Update policy network as $\theta_p \leftarrow \theta_p + \nabla_{\theta_p} J(\theta_p)$.
  end for
end for

Timeline Design. In one time interval (10 minutes), the main activities are conducted sequentially, also illustrated in Figure 4.

- *Vehicle status updates:* Vehicles will be set offline (i.e. off from service) or online (i.e. start working) following a distribution learned from real data using a maximum likelihood estimation.
- *Order generation:* The new orders generated at the current time step are bootstrapped from real orders occurred in the same time interval. Since the order will naturally reposition vehicles in a wide range, this procedure keeps the reposition from orders similar to the real data.
- *Interact with agents:* This step computes state as input to fleet management algorithm and applies the allocations for agents.
- *Order assignments:* All available orders are assigned through a two-stage procedure. In the first stage, the orders in one grid are assigned to the vehicles in the same grid. In the second stage, the remaining unfilled orders are assigned to the vehicles in its neighboring grids. This two-stage procedure is essential to stimulate the real world activities.

Calibration. The effectiveness of the simulator is guaranteed by calibration against the real data regarding the most important performance measurement: the gross merchandise volume (GMV). As shown in Figure 3, after the calibration procedure, the GMV in the simulator is very similar to that from the ride-sharing platform. The r^2 between simulated GMV and real GMV is 0.9331 and the Pearson correlation is 0.9853 with p-value $p < 0.00001$.

6 EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the effectiveness of our proposed method.

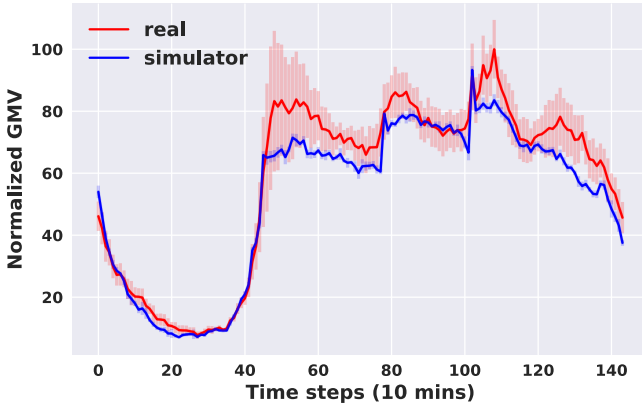


Figure 3: The simulator calibration in terms of GMV. The red curves plot the GMV values of real data averaged over 7 days with standard deviation, in 10-minute time granularity. The blue curves are simulated results averaged over 7 episodes.

6.1 Experimental settings

In the following experiments, both of training and evaluation are conducted on the simulator introduced in Sec 5. For all the competing methods, we prescribe two sets of random seed that control the dynamics of the simulator for training and evaluation, respectively. Examples of dynamics in simulator include order generations, and stochastically status update of all vehicles. In this setting, we can test the generalization performance of algorithms when it encounters unseen dynamics as in real scenarios. The performance is measured by GMV (the total value of orders served in the simulator) gained by the platform over one episode (144 time steps in the simulator), and order response rate, which is the averaged number of orders served divided by the number of orders generated. We use the first 15 episodes for training and conduct evaluation on the following ten episodes for all learning methods. The number of available vehicles at each time in different locations is counted by a pre-dispatch procedure. This procedure runs a virtual two-stage order dispatching process to compute the remaining available vehicles in each location. On average, the simulator has 5356 agents per time step waiting for management. All the quantitative results of learning methods presented in this section are averaged over three runs.

6.2 Performance comparison

In this subsection, the performance of following methods are extensively evaluated by the simulation.

- **Simulation:** This baseline simulates the real scenario without any fleet management. The simulated results are calibrated with real data in Sec 5.
- **Diffusion:** This method diffuses available vehicles to neighboring grids randomly.
- **Rule-based:** This baseline computes a $T \times N$ value table $\mathbf{V}_{rule}$, where each element $\mathbf{V}_{rule}(t, j)$ represents the averaged reward of an agent staying in grid $\mathbf{g}_j$ at time step t. The rewards are averaged over ten episodes controlled by random seeds that are different with testing episodes. With the value table, the agent

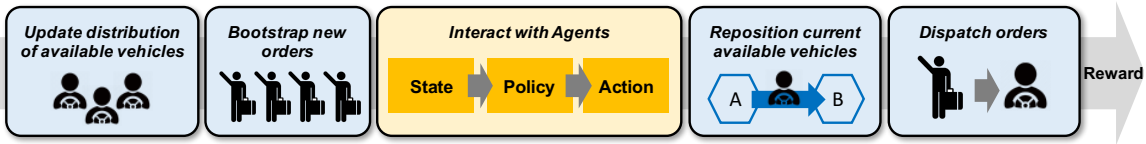


Figure 4: Simulator time line in one time step (10 minutes).

samples its action based on the probability mass function normalized from the values of neighboring grids at the next time step. For example, if an agent located in $\mathbf{g}_1$ at time t and the current valid actions are $[\mathbf{g}_1, \mathbf{g}_2]$ and $[\mathbf{g}_1, \mathbf{g}_1]$, the rule-based method sample its actions from $p(a_t^i \triangleq [\mathbf{g}_1, \mathbf{g}_j]) = \mathbf{V}_{rule}(t+1, j)/(\mathbf{V}_{rule}(t+1, 2) + \mathbf{V}_{rule}(t+1, 1)), \forall j = 1, 2$.

- **Value-Iter:** It dynamically updates the value table based on policy evaluation [24]. The allocation policy is computed based on the new value table, the same used in the rule-based method, while the collaborative context is considered.
- **T-Q learning**: The standard independent tabular Q-learning [24] learns a table $\mathbf{q}_{tabular} \in \mathbb{R}^{T \times N \times 7}$ with ϵ-greedy policy. In this case the state reduces to time and the location of the agent.
- **T-SARSA**: The independent tabular SARSA [24] learns a table $\mathbf{q}_{sarsa} \in \mathbb{R}^{T \times N \times 7}$ with same setting of states as T-Q learning.
- **DQN**: The independent DQN is currently the state-of-the-art as we introduced in Sec 4.1. Our Q network is parameterized by a three-layer ELUs [7] and we adopt the ϵ-greedy policy as the agent policy. The ϵ is annealed linearly from 0.5 to 0.1 across the first 15 training episodes and fixed as $\epsilon = 0.1$ during the testing.
- **cDQN**: The contextual DQN as we introduced in Sec 4.2. The ϵ is annealed the same as in DQN. At the end of each episode, the Q-network is updated over 4000 batches, i.e. $M_1 = 4000$ in Alg 2. To ensure a valid context masking, the activation function of the output layer of the Q-network is ReLU + 1.
- **cA2C**: The contextual multi-agent actor-critic as we introduced in Sec 4.3. At the end of each episode, both the policy network and the value network are updated over 4000 batches, i.e. $M_1 = M_2 = 4000$ in Alg 2. Similar to cDQN, The output layer of the policy network uses ReLU + 1 as the activation function to ensure that all elements in the original logits $\mathbf{P}(\mathbf{s}_t^i)$ are positive.

Except for the first baseline, the geographic context is considered in all methods so that the agents will not navigate to the invalid grid. Unless other specified, the value function approximations and policy network in contextual algorithms are parameterized by a three-layer ReLU [12] with node sizes of 128, 64 and 32, from the first layer to the third layer. The batch size of all deep learning methods is fixed as 3000, and we use AdamOptimizer with a learning rate of $1e{-}3$. Since performance of DQN varies a lot when there are a large number of agents, the first column in the Table 1 for DQN is averaged over the best three runs out of six runs, and the results for all other methods are averaged over three runs. Also, the centralized critics of cDQN and cA2C are initialized from a pre-trained value network using the historical mean of order values computed from ten episodes simulation, with different random seeds from both training and evaluation.

To test the robustness of proposed method, we evaluate all competing methods under different numbers of initial vehicles, and the results are summarized in Table 1. The results of *Diffusion* improved the performance a lot, possibly because the method sometimes encourages the available vehicles to leave the grid with high density of available vehicles, and thus the imbalanced situation is alleviated. The *Rule-based* method that repositions vehicles to the grids with a higher demand value, improves the performance of random repositions. The *Value-Iter* dynamically updates the value table according to the current policy applied so that it further promotes the performance upon *Rule-based*. Comparing the results of *Value-Iter*, *T-Q learning* and *T-SARSA*, the first method consistently outperforms the latter two, possibly because the usage of a centralized value table enables coordinations, which helps to avoid conflict repositions. The above methods simplify the state representation into a spatial-temporal value representation, whereas the DRL methods account both complex dynamics of supply and demand using neural network function approximations. As the results shown in last three rows of Table 1, the methods with deep learning outperforms the previous one. Last but not least, the contextual algorithms (cA2C and cDQN) largely outperform the independent DQN (DQN), which is the state-of-the-art among large-scale multi-agent DRL method and all other competing methods.

6.3 Consideration of reposition costs

In reality, each reposition comes with a cost. In this subsection, we consider such reposition costs and estimated them by fuel costs. Since the travel distance from one grid to another is approximately 1.2km and the fuel cost is around 0.5 RMB/km, we set the cost of each reposition as $c = 0.6$. In this setting, the definition of agent, state, action and transition probability is same as we stated in Sec 3. The only difference is that the repositioning cost is included in the reward when the agent is repositioned to different locations. Therefore, the GMV of one episode is the sum of all served order value substracted by the total of reposition cost in one episode. For example, the objective function for DQN now includes the reposition cost as follows:

$$\mathbb{E}\left[Q(\mathbf{s}_t^i, a_t^i; \theta) - \left(r_{t+1}^i - c + \gamma \max_{a_{t+1}^i} Q(\mathbf{s}_{t+1}^i, a_{t+1}^i; \theta')\right)\right]^2, \quad (13)$$

where $a_t^i \triangleq [\mathbf{g}_o, \mathbf{g}_d]$, and if $\mathbf{g}_d = \mathbf{g}_o$ then $c = 0$, otherwise $c = 0.6$. Similarly, we can consider the costs in cA2C. However, it is hard to apply them to cDQN because the assumption, that different actions that lead to the same location should share the same action value, which is not held in this setting. Therefore we compared two deep learning methods that achieve best performances in the previous section. As the results shown in Table 2, the DQN tends to reposition more agents while the cA2C achieves better performance in terms of both GMV and order response rate, with lower cost. The training procedures and the network architecture are the same as described in the previous section.

Table 1: Performance comparison of competing methods in terms of GMV and order response rate. For a fair comparison, the random seeds that control the dynamics of the environment are set to be the same across all methods.

	100% initial vehicles		90% initial vehicles		10% initial vehicles	
	Normalized GMV	Order response rate	Normalized GMV	Order response rate	Normalized GMV	Order response rate
Simulation	100.00 ± 0.60	81.80% ± 0.37%	98.81 ± 0.50	80.64% ± 0.37%	92.78 ± 0.79	70.29% ± 0.64%
Diffusion	105.68 ± 0.64	86.48% ± 0.54%	104.44 ± 0.57	84.93% ± 0.49%	99.00 ± 0.51	74.51% ± 0.28%
Rule-based	108.49 ± 0.40	90.19% ± 0.33%	107.38 ± 0.55	88.70% ± 0.48%	100.08 ± 0.50	75.58% ± 0.36%
Value-Iter	110.29 ± 0.70	90.14% ± 0.62%	109.50 ± 0.68	89.59% ± 0.69%	102.60 ± 0.61	77.17% ± 0.53%
T-Q learning	108.78 ± 0.51	90.06% ± 0.38%	107.71 ± 0.42	89.11% ± 0.42%	100.07 ± 0.55	75.57% ± 0.40%
T-SARSA	109.12 ± 0.49	90.18% ± 0.38%	107.69 ± 0.49	88.68% ± 0.42%	99.83 ± 0.50	75.40% ± 0.44%
DQN	114.06 ± 0.66	93.01% ± 0.20%	113.19 ± 0.60	91.99% ± 0.30%	103.80 ± 0.96	77.03% ± 0.23%
cDQN	115.19 ± 0.46	94.77% ± 0.32%	**114.29** ±0.66	94.00% ± 0.53%	105.29 ± 0.70	79.28% ± 0.58%
cA2C	**115.27** ±0.70	94.99% ± 0.48%	113.85 ± 0.69	93.99% ± 0.47%	**105.62** ±0.66	79.57% ± 0.51%

Table 2: The effectiveness of contextual multi-agent actor-critic considering dispatch costs.

	Normalized GMV	Order response rate	Repositions
cA2C	112.70 ± 0.64	94.74% ± 0.57%	408859
DQN	110.81 ± 0.68	92.50% ± 0.50%	606932

Table 3: The effectiveness of averaged reward design. The performance of methods using the raw reward (second column) is much worse than the performance of methods using the averaged reward.

	Proposed methods	Raw Reward
	Normalized GMV/Order response rate	Normalized GMV/Order response rate
cA2C	115.27 ±0.70/94.99% ± 0.48%	105.75 ± 1.17/88.09% ± 0.74%
cDQN	115.19 ± 0.46/94.77% ± 0.32%	108.00 ± 0.35 /89.53% ± 0.31%

Table 4: The effectiveness of context embedding.

	Normalized GMV/Order response rate	Repositions
	Without reposition cost	
cA2C	115.27 ± 0.70/94.99% ± 0.48%	460586
cA2C-v1	114.78 ± 0.67/94.52% ± 0.49%	704568
cA2C-v2	111.39 ± 1.65/92.12% ± 1.03%	846880
	With reposition cost	
cA2C	112.70 ± 0.64/94.74% ± 0.57%	408859
cA2C-v3	110.43 ± 1.16/93.79% ± 0.75%	593796

6.4 The effectiveness of averaged reward design

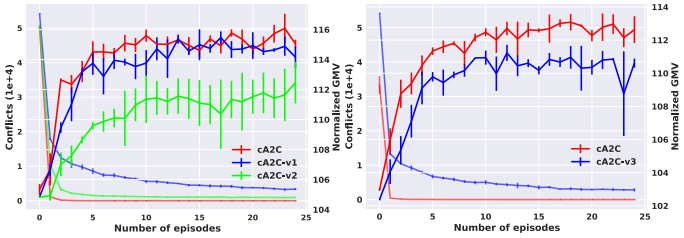
In multi-agent RL, the reward design for each agent is essential for the success of learning. In fully cooperative multi-agent RL, the reward for all agents is a single global reward [5], while it suffers from the credit assignment problem for each agent's action. Splitting the reward to each agent will alleviate this problem. In this subsection, we compare two different designs for the reward of each agent: the averaged reward of a grid as stated in Sec 3 and the total reward of a grid that does not average on the number of available vehicles at that time. As shown in table 3, the methods with averaged reward (cA2C, cDQN) largely outperform those using total reward, since this design naturally encourages the coordinations among agents. Using total reward, on the other hand, is likely to reposition an excessive number of agents to the location with high demand.

6.5 Ablations on policy context embedding

In this subsection, we evaluate the effectiveness of context embedding, including explicitly coordinating the actions of different agents through the collaborative context, and eliminating the invalid actions with geographic context. The following variations of proposed methods are investigated in different settings.

- cA2C-v1: This variation drops collaborative context of cA2C in the setting that does not consider reposition cost.
- cA2C-v2: This variation drops both geographic and collaborative context of cA2C in the setting that does not consider reposition cost.
- cA2C-v3: This variation drops collaborative context of cA2C in the setting that considers reposition cost.

Figure 5: Convergence comparison of cA2C and its variations without using context embedding in both settings, with and without reposition costs. The X-axis is the number of episodes. The left Y-axis denotes the number of conflicts and the right Y-axis denotes the normalized GMV in one episode.



(a) Without reposition cost (b) With reposition cost

The results of above variations are summarized in Table 4 and Figure 5. As seen in the first two rows of Table 4 and the red/blue curves in Figure 5 (a), in the setting of zero reposition cost, cA2C achieves the best performance with much less repositions (65.37%) comparing with cA2C-v1. Furthermore, collaborative context embedding achieves significant advantages when the reposition cost is considered, as shown in the last two rows in Table 4 and Figure 5 (b). It not only greatly improves the performance but also accelerates the convergence. Since the collaborative context largely narrows down the action space and leads to a better policy solution in the sense of both effectiveness and efficiency, we can conclude that coordination based on collaborative context is effective. Also, comparing the performances of cA2C and cA2C-v2 (red/green curves in Figure 5 (a)), apparently the policy context embedding (considering both geographic and collaborative context) is essential to performance, which greatly reduces the redundant policy search.

(a) At 01:50 am. (b) At 06:40 pm.

Figure 6: Illustration on the repositions nearby the airport at 1:50 am and 06:40 pm. The darker color denotes the higher state value and the blue arrows denote the repositions.

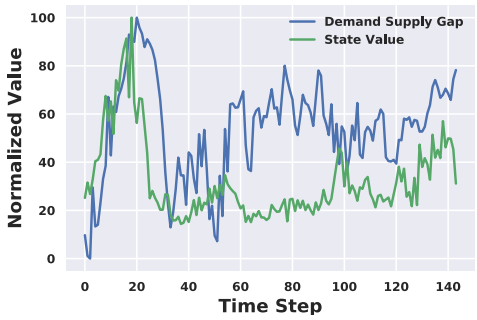


Figure 7: The normalized state value and demand-supply gap over one day.

6.6 Qualitative study

In this section, we analyze whether the learned value function can capture the demand-supply relation ahead of time, and the rationality of allocations. To see this, we present a case study on the region nearby the airport. The state value and allocation policy is acquired from cA2C that was trained for ten episodes. We then run the well-trained cA2C on one testing episode, and qualitatively exam the state value and allocations under the unseen dynamics. The sum of state values and demand-supply gap (defined as the number of orders minus the number of vehicles) of seven grids that cover the CTU airport is visualized. As seen in Figure 7, the state value can capture the future dramatic changes of demand-supply gap. Furthermore, the spatial distribution of state values can be seen in Figure 6. After the midnight, the airport has a large number of orders, and less available vehicles, and therefore the state values of airport are higher than other locations. During the daytime, more vehicles are available at the airport so that each will receive less reward and the state values are lower than other regions, as shown in Figure 6 (b). In Figure 6 and Figure 7, we can conclude that the value function can estimate the relative shift of demand-supply gap from both spatial and temporal perspectives. It is crucial to the performance of cA2C since the coordination is built upon the state values. Moreover, as illustrated by blue arrows in Figure 6, we see that the allocation policy gives consecutive allocations from lower value grids to higher value grids, which can thus fill the future demand-supply gap and increase the GMV.

7 CONCLUSIONS

In this paper, we first formulate the large-scale fleet management problem into a feasible setting for deep reinforcement learning. Given this setting, we propose contextual multi-agent reinforcement learning framework, in which two contextual algorithms cDQN and cA2C are developed and both of them achieve the large scale agents' coordination in fleet management problem. cA2C enjoys both flexibility and efficiency by capitalizing a centralized value network and decentralized policy execution embedded with contextual information. It is able to adapt to different action space in an end-to-end training paradigm. A simulator is developed and calibrated with the real data provided by Didi Chuxing, which served as our training and evaluation platform. Extensive empirical studies under different settings in simulator have demonstrated the effectiveness of the proposed framework.

ACKNOWLEDGMENTS

This material is based in part upon work supported by the National Science Foundation under Grant IIS-1565596, IIS-1615597, IIS-1749940 and Office of Naval Research N00014-14-1-0631, N00014-17-1-2265.

REFERENCES

[1] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866* (2017).
[2] Bram Bakker, Shimon Whiteson, Leon Kester, and Frans CA Groen. 2010. Traffic light control by multiagent reinforcement learning systems. In *Interactive Collaborative Information Systems*. Springer, 475–510.
[3] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The Arcade Learning Environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)* 47 (2013), 253–279.
[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI gym. *arXiv preprint arXiv:1606.01540* (2016).
[5] Lucian Busoniu, Robert Babuska, and Bart De Schutter. 2008. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, And Cybernetics-Part C: Applications and Reviews, 38 (2), 2008* (2008).
[6] Didi Chuxing. [n. d.]. ([n. d.]). http://www.didichuxing.com/en/
[7] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).
[8] Pierre J Dejax and Teodor Gabriel Crainic. 1987. Survey paper - a review of empty flows and fleet management models in freight transportation. *Transportation science* 21, 4 (1987), 227–248.
[9] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2017. Counterfactual Multi-Agent Policy Gradients. *arXiv preprint arXiv:1705.08926* (2017).
[10] Gregory A Godfrey and Warren B Powell. 2002. An adaptive dynamic programming algorithm for dynamic fleet management, I: Single period travel times. *Transportation Science* 36, 1 (2002), 21–39.
[11] Gregory A Godfrey and Warren B Powell. 2002. An adaptive dynamic programming algorithm for dynamic fleet management, II: Multiperiod travel times. *Transportation Science* 36, 1 (2002), 40–54.
[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *ECCV*. Springer, 630–645.
[13] Z Li, Y Hong, and Z Zhang. 2016. *Do on-demand Ride-Sharing Services Affect Traffic Congestion? Evidence from Uber Entry.* Technical Report. Working paper, available at SSRN: https://ssrn. com/abstract= 2838043.
[14] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *arXiv preprint arXiv:1706.02275* (2017).
[15] Michał Maciejewski and Kai Nagel. 2013. The influence of multi-agent cooperation on the efficiency of taxi dispatching. In *PPAM*. Springer, 751–760.
[16] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *ICML*. 1928–1937.
[17] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.
[18] Duc Thien Nguyen, Akshat Kumar, and Hoong Chuin Lau. 2017. Collective multiagent sequential decision making under uncertainty. *AAAI* (2017).

[19] Duc Thien Nguyen, Akshat Kumar, and Hoong Chuin Lau. 2017. Policy gradient with value function approximation for collective multiagent planning. *NIPS* (2017).

[20] Tu-Hoa Pham, Giovanni De Magistris, and Ryuki Tachibana. 2017. OptLayer-Practical Constrained Optimization for Deep Reinforcement Learning in the Real World. *arXiv preprint arXiv:1709.07643* (2017).

[21] Kiam Tian Seow, Nam Hai Dang, and Der-Horng Lee. 2010. A collaborative multiagent taxi-dispatch system. *IEEE T-ASE* 7, 3 (2010), 607–616.

[22] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[23] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *Nature* 550, 7676 (2017), 354.

[24] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.

[25] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. 2017. Multiagent cooperation and competition with deep reinforcement learning. *PloS one* 12, 4 (2017), e0172395.

[26] Ming Tan. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *ICML*. 330–337.

[27] Uber. [n. d.]. ([n. d.]). https://www.uber.com/

[28] Chong Wei, Yinhu Wang, Xuedong Yan, and Chunfu Shao. 2017. Look-ahead Insertion Policy for A Shared-taxi System Based on Reinforcement Learning. *IEEE Access* (2017).

[29] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. 2018. Mean Field Multi-Agent Reinforcement Learning. *ICML* (2018).

[30] Lianmin Zheng, Jiacheng Yang, Han Cai, Weinan Zhang, Jun Wang, and Yong Yu. 2017. MAgent: A Many-Agent Reinforcement Learning Platform for Artificial Collective Intelligence. *arXiv preprint arXiv:1712.00600* (2017).