Multiple Agents Dispatch via Batch Synchronous Actor Critic in Autonomous Mobility on Demand Systems

Jiyao Li and Vicki H. Allan
Department of Computer Science, Utah State University, Logan, Utah, U.S.A

Keywords: Autonomous Mobility on Demand (AMoD) Systems, Vehicle Repositioning, Task Assignment, Multiagent Reinforcement Learning (MARL).

Abstract: Autonomous Mobility on Demand (AMoD) systems are a promising area in the emerging field of intelligent transportation systems. In this paper, we focus on the problem of how to dispatch a fleet of autonomous vehicles (AVs) within a city while balancing supply and demand. We first formulate the problem as a Markov Decision Process (MDP) of which the goal is to maximize the accumulated average reward, then propose the Multiagent Reinforcement Learning (MARL) framework. The Temporal-Spatial Dispatching Network (TSD-Net) that combines both policy and value network learns representation features facilitating spatial information with its temporal signals. The Batch Synchronous Actor Critic (BS-AC) samples experiences from the Rollout Buffer with replacement and trains parameters of the TSD-Net. Based on the state value from the TSD-Net, the Priority Destination Sampling Assignment (PDSA) algorithm defines orders' priority by their destinations. Popular destinations are preferred as it is easier for agents to find future work in a popular location. Finally, with the real-world city scale dataset from Chicago, we compare our approach to several competing baselines. The results show that our method is able to outperform other baseline methods with respect to effectiveness, scalability, and robustness.

1 INTRODUCTION

As urban populations have increased all over the world, the transportation scene in most metropolitan areas had been dominated by an ever-increasing number of private vehicles (UN, 2015). While highly convenient for individuals, a community containing an excessive number of vehicles may find itself experiencing numerous social and environmental problems, including air pollution, parking limitations, high energy consumption, and traffic congestion. The emergence of Autonomous Mobility on Demand (AMoD) systems is a promising alternative to the paradigm of private vehicles. The AMoD systems manage a fleet of autonomous vehicles (AVs) around a city. Passengers place orders through smart devices such as cellphones or tablets, and the AMoD platform dispatches AVs to provide them with fast, point-to-point travel services.

Dispatching a fleet of AVs across a city while maintaining balance between supply (AVs) and demand (orders) is one of the important operations in AMoD systems. In general, the dispatching operation comprises two primary tasks: (i) repositioning vacant AVs to locations where rider requests occur, enabling them to get closer to potential passengers, and (ii) assigning suitable orders to available AVs for delivery to the required areas through ride-matching.

However, coordinating multiple AVs in a dynamic environment is extremely difficult: (i) occurrences of orders are unknown both with respect to time and location, and the quantity of orders are constantly fluctuating over space and time; (ii) multiple AVs working under the same system may affect each other; (iii) customers may cancel the service when the waiting period exceeds their patience.

Most previous research work that applied Deep Reinforcement Learning (DRL) to solve these problems still suffers from certain limitations: (i) it takes so long to collect experiences for the AMoD system that AVs are unable to effectively interact with the environment in real-time. (ii) most neural network architectures of the AMoD system only consider spatial information from the environment but ignore temporal properties behind the spatial distribution.

To address the challenges and limitations in AMoD systems, we propose a Multiagent Reinforcement Learning (MARL) framework. Our contribu-

Li, J. and Allan, V.
Multiple Agents Dispatch via Batch Synchronous Actor Critic in Autonomous Mobility on Demand Systems.
DOI: 10.5220/0012351700003636

In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 2*, pages 198-209
ISBN: 978-989-758-680-4; ISSN: 2184-433X

tions are:

- We propose a new neural network architecture named **T**emporal-**S**patial **D**ispatching **Net**work (TSD-Net) that combines both policy network (Actor) and value network (Critic). Also, the Gated Recurrent Units (GRU) are embedded to process temporal signals behind spatial information.
- To improve the efficiency of the training procedure, we designed a **B**atch **S**ynchronous **A**ctor **C**ritic (BS-AC) algorithm. Instead of spending a long time to collecting experiences, the BS-AC samples experiences with replacement to quickly achieve effective results.
- To address the issue of insufficient supply during peak hours, we employ the **P**riority **D**estination **S**ampling **A**ssignment (PDSA) algorithm. This algorithm is utilized to select the riders with the most preferred destinations when supply is insufficient to satisfy all riders.
- To verify the effectiveness, scalability and robustness of our framework, extensive experiments are conducted with a city-scale dataset from the City of Chicago.

This paper is organized as follows: First, literature on AMoD and DRL will be reviewed in Section 2. Preliminaries and problem formulation are introduced in Section 3 and Section 4. Our proposed MARL framework is described in Section 5, and experiment results are analyzed in Section 6. Finally, conclusions are in Section 7.

2 RELATED WORK

RL in AMoD Systems. Reinforcement Learning (RL) has recently become prevalent in the field of AMoD systems. (Xu et al., 2018) learns the spatial features to identify popular and unpopular regions of a city and plans bipartite matches between vehicles and orders by Kuhn-Munkres (KM) algorithm (Munkres, 1957). However, the tabular learning method limits the dimension of the state space. The proposed holistic mechanism by (Li and Allan, 2022a), termed T-Balance, integrates Q-Learning Idle Movement (QIM) for vehicle repositioning. However, its effectiveness is hindered by a severely limited action space, resulting in a lack of flexibility. (Wen et al., 2017) applied the Deep-Q-Network (DQN) to manage the fleet of vehicles around a city to gain balance between supply and demand. (Li and Allan, 2022b) considered various degrees of nodes in the graph and designed an action mask with DQN to make action selection efficient. (Zheng et al., 2022) proposed an action sampling DQN such that vehicles at the same region can select different actions. However, none of these three works consider cooperation among vehicles. (Lin et al., 2018) and (Wang et al., 2021) utilized the Advantage Actor Critic (A2C) to balance available vehicles and orders. Both of them train the model by the same batch of experiences iteratively without importance sampling. Such a training procedure leads to bias since the A2C is an on-policy approach. (Sun et al., 2022) proposed a new network architecture with Proximal Policy Optimization for training. However, it takes so long to collect experiences that the policy network cannot be updated frequently. To handle AMoD systems across multiple cities, (Wang et al., 2018) applied the transfer learning with the DQN and (Gammelli et al., 2022) proposed graph meta-reinforcement learning to adapt to different cities, but neither consider the adaption of various settings within a city.

Deep Reinforcement Learning (DRL). Most of the DRL can be categorized into three families: value based method, policy based method, and actor critic. In the field of the value based method (started from the TD-Gammon architecture (Tesauro et al., 1995)) the DQN (Mnih et al., 2013) is integrated with both Convolutional Neural Network (Krizhevsky et al., 2017) and Experience Replay Buffer (Lin, 1992), achieving human level control on the Atari games. However, the target value from the policy network is varied so often that the learning process is extremely difficult. (Mnih et al., 2015) conquered the problem by separating the target network from the policy network and updating it periodically. Afterwards, several augmented versions of the DQN were published: (Van Hasselt et al., 2016) utilized double Q-learning to tackle the overestimation of Q value, (Wang et al., 2016) proposed a dueling network architecture to mitigate the bias of the policy distribution, and (Schaul et al., 2015) designed a sampling mechanism where valuable experiences have priority to be learned. However, the intrinsic drawback of the DQN is that the convergence is not guaranteed. In terms of policy based method, the Monte Carlo Policy Gradient with approximation function proposed by (Sutton et al., 1999) has better convergence properties. The weakness of the method is that it can only be applied in episodic scenarios and high variance is introduced. The family of actor critic algorithms can solve this issue. (Mnih et al., 2016) proposed an online learning paradigm named Asynchronous Actor Critic where the Advantage Function can inhibit the occurrence of high variance, but it is inefficient in that experiences cannot be reused. (Schulman et al., 2017) overcomes this drawback by introducing importance sampling with clip function where the ratio is constrained within a certain range.

3 PRELIMINARIES

Several definitions of Autonomous Mobility on Demand (AMoD) systems are introduced in the following:

Map. The city is assumed to be partitioned into a grid with elements g_i. An undirected edge e_{ij} connects each pair of adjacent grid cells g_i and g_j. The map of the city can be represented by an undirected graph $M = (G, E)$, where G is the set of all grid cells $g_i \in G$, and E is the set of all edges $e_{ij} \in E$.

Task. A task τ_i is generated whenever a passenger's request occurs. A task can be represented as a tuple $\tau_i = (t_{\tau_i}, o_{\tau_i}, d_{\tau_i}, l_{\tau_i}, p_{\tau_i})$, where t_{τ_i} is the occurrence time of τ_i, o_{τ_i} and d_{τ_i} are the origin and destination of τ_i provided from the passenger, l_{τ_i} is the trip duration of τ_i estimated by the AMoD system, and p_{τ_i} is the patience duration of the passenger; in other words, the passenger will switch to alternative transportation after time $t_{\tau_i} + p_{\tau_i}$.

Agent. An agent is an autonomous vehicle (AV) in the AMoD system. An agent's information can be represented by a tuple $ag_i = (id_{ag_i}, loc_{ag_i}, stat_{ag_i})$, where id_{ag_i} is the unique identification number of agent ag_i, loc_{ag_i} is the current location of ag_i indicated by grid number g_i, and $stat_{ag_i}$ is the status of ag_i, which is either *in_service*, indicating the agent ag_i has been allocated a task, or *idle*, denoting it has no task on hand and is seeking a new one.

The framework of the AMoD system is shown in Fig.1. The AMoD Platform receives Environment Information at each time t, from which the State Information is obtained by the Spatial Encoder (SE) and the location of vehicles along with the encoded State Information from the Encoder are collected by Experience Collector appending the corresponding experience to the Rollout Buffer where training experiences are stored. The Batch Synchronous Actor Critic module (BS-AC) retrieves experiences from the Rollout Buffer, updating the parameters of the Temporal-Spatial Dispatching Network (TSD-Net) and then clearing the Rollout Buffer. The encoded State Information is also fed into the TSD-Net, which outputs the route information of repositioning for idle agents and the state value indicating future gains for Priority Destination Sampling Assignment module (PDSA). Based on the state value, the PDSA selects appropriate tasks (requests) for agents by tasks' priority such that more tasks can be served in the future. In our environment, not every order can be satisfied. We prioritize trips that maximize the total reward.

4 FORMULATION

Typically, dispatching multiple agents in the AMoD system is a sequential decision making problem that can be formulated as a Markov Decision Process (MDP). Each agent in *idle* status in the AMoD System is considered as available in the MDP model. For efficiency of computation and storage space, agents share the same Neural Network and Rollout Buffer rather than owning them independently. Sharing one buffer means all AVs share experiences together such that they can learn fast. We use the entropy to keep smaller differences among action probability, and sample the action each time. Note that agents with the same temporal-spatial property (e.g. currently in the same grid cell) are homogeneous, and they have the same policy distribution. The MDP can be represented as a tuple $\mathcal{G} = (\mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$, where $\mathcal{N}$ is the fluctuating number of available agents, $\mathcal{S}$ and $\mathcal{A}$ represent state space and action space respectively, $\mathcal{R}$ denotes the reward function, $\mathcal{P}$ indicates the state transition function and γ is a discount factor. The details of each element are given as follows.

State. The state $s_t^i \in \mathcal{S}$ is a representation of the environment that the ith available agent interacts with at time t. In our work, it can be denoted as $[\overrightarrow{T_t}, \overrightarrow{X_t}, g_t^i]$ where $\overrightarrow{T_t}$ and $\overrightarrow{X_t}$ are vectors representing spatial distribution of tasks and agents at time t respectively, and g_t^i is the location of the ith available agent at time t. It is worth noting that the length of $\overrightarrow{T_t}$ and $\overrightarrow{X_t}$ is the number of grid cells in the city, $\overrightarrow{T_t}[j]$ and $\overrightarrow{X_t}[j]$ are the number of tasks/agents in grid cell j at time t.

Action. The joint action vector $\overrightarrow{a_t} \in \mathcal{A} = \mathcal{A}^1 \times \mathcal{A}^2 \times ... \times \mathcal{A}^{\mathcal{N}}$ indicates the repositioning strategy of all available agents at time t, where $\mathcal{A}^i$ is the action space of the ith available agent. It can be represented by $\overrightarrow{a_t} = \{a_t^1, a_t^2, ..., a_t^{\mathcal{N}}\}$ where $a_t^i \in \mathcal{A}^i$ is the action executed by the ith available agent at time t. The action space $\mathcal{A}^i$ of the ith available agent specifies where the agent can go next, either moving to one of the neighboring grid cells or staying in the current grid cell.

Reward. The ith available agent will receive an immediate reward $r_t^i \in \mathcal{R} \to \mathbb{R}$ from the environment after executing action a_t^i at time t. To avoid excessive agents flowing into grid cells with a high volume of tasks, each agent receives a positive reward $(+1)$ with an adjustment indicating ratio of supply and demand if it is assigned a task, as shown in Eq.1 where

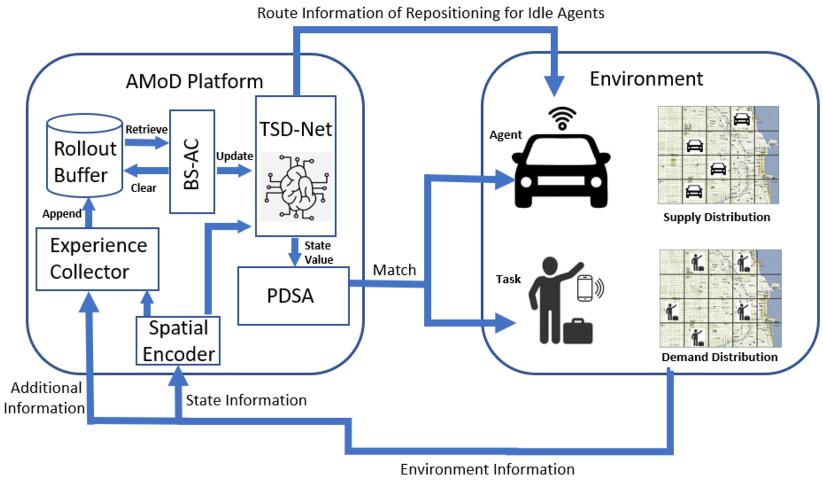


Figure 1: Autonomous Mobility on Demand (AMoD) System.

$\overrightarrow{X_t}[g_t^i]$ is the number of available agents and $\overrightarrow{T_t}[g_t^i]$ is the number of tasks at grid cell g_t^i (g_t^i is the grid cell that the ith available agent moves into at time t).

$$r_t^i = \begin{cases} 1 - \frac{\overrightarrow{X_t}[g_t^i] - \overrightarrow{T_t}[g_t^i]}{\overrightarrow{X_t}[g_t^i]} & \text{if assigned a task} \\ -1 & \text{otherwise} \end{cases} \quad (1)$$

State Transition. The state transition $s_{t+1}^i = \mathcal{P}(s_t^i, a_t^i)$ specifies what state the ith available agent will be in at time t, based on current state s_t^i and action a_t^i. In the MDP, there are two types of transitions: s_{t+1}^i will be NULL if the agent is assigned a task; otherwise $s_{t+1}^i = [\overrightarrow{T}_{t+1}, \overrightarrow{A}_{t+1}, g_{t+1}^i]$.

Discount Factor. The discount factor $\gamma \in [0,1]$ specifies the impact degree of the future rewards. If $\gamma = 0$, an agent is myopic and no future rewards are taken into account; on the other hand, all future rewards have the same weight as the current one if $\gamma = 1$.

The goal of the MDP is to maximize the long term benefits of each agent, which can be represented by $V(s^i) = \sum_{t=T_0}^{T} \gamma^{t-T_0} \cdot r_t^i$ where T_0 is the current time.

5 METHODOLOGY

5.1 Temporal-Spatial Dispatching Network

The architecture of the Temporal-Spatial Dispatching Network (TSD-Net) is shown in Fig.2. Since the spatial information has time series property, we design the Spatial Encoder (SE) with the Gated Recurrent Unit (GRU) (Cho et al., 2014) where the hidden state h_{t-1} involving previous spatial information is taken as one of inputs (here h_{t-1} represents both h_{t-1}^1 and h_{t-1}^2 in the figure). The purpose of the SE is to learn the representation of spatial distributions. To save computational resources, the TSD-Net plays the role of both policy network (Actor) and value network (Critic), sampling appropriate actions a_t^i for each available agent and outputting the scale of the state value function $V(s_t^i, \theta)$ (Sutton and Barto, 2018) at each time t.

The *Norm* function within the SE normalizes the vector of the spatial distribution, as shown in Eq.(2), where $\vec{x_t}$ is the input of the function, e.g. the spatial distribution vector $\overrightarrow{X_t}$ and $\overrightarrow{T_t}$. There are two reasons for using the *Norm* function. (*i*) During rush hours, most tasks and agents converge around downtown areas while almost none can be found in non-busy regions, meaning that many elements of vector $\vec{x_t}$ are 0, which would cause inefficiency in the learning process. The *Norm* function can help get rid of most zeros. (*ii*) The close range of the normalized vector $\vec{x_t}$ improves he sensitivity of the network. Note that the output of *Norm* is all-ones vector if $std(\vec{x_t}) = 0$.

$$Norm(\vec{x_t}) = \frac{\vec{x_t} - mean(\vec{x_t})}{std(\vec{x_t})} \quad (2)$$

Using the properties of time series of spatial data, the Gated Recurrent Unit (GRU) can combine previous corresponding information to the present task, as shown in Eq.(3) where input $\vec{y_t}$ can be seen as the output of the *Norm* function and hidden state h_{t-1} is the output of the GRU at time $t-1$. The GRU is the variant version of the LSTM (Hochreiter and Schmidhuber, 1997) which successfully solves the 'long term dependency' problem in the Recursive Neural Network (Bengio et al., 1994). When comparing the

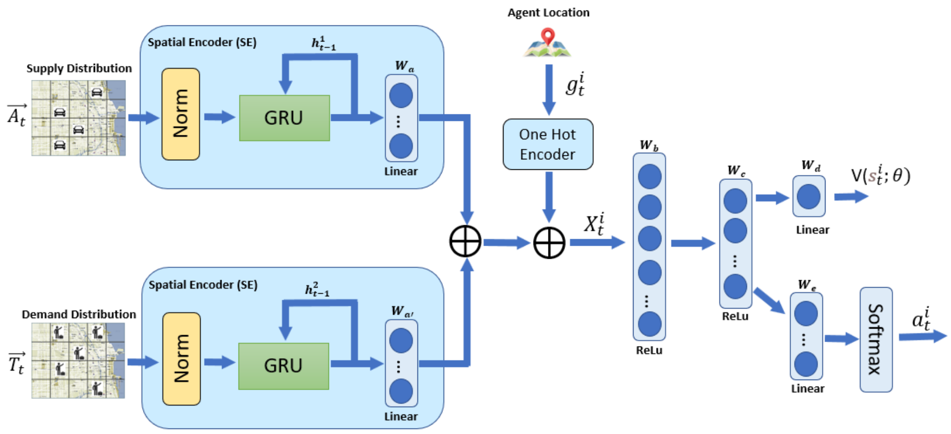


Figure 2: Temporal-Spatial Dispatching Network (TSD-Net) Architecture.

GRU with the LSTM, it is evident that the GRU exhibits a more compact model structure while retaining its desirable properties.

$$h_t = GRU(\vec{y}_t, h_{t-1}) \tag{3}$$

The overall operations of the spatial encoding is shown in Eq.(4) where W represents parameters of Linear Layer such as W_a or $W_{a'}$ and $*$ is the matrix multiplication. In this context, we have devised two distinct Spatial Encoders (SE) to independently process task-related and agent-specific information. The primary objective of these SEs is to convert spatial data into a representative vector encompassing various spatial distribution features, integrating it with the latest temporal learning outcomes through the utilization of the GRU.

$$SE(\vec{x}_t) = W * GRU(Norm(\vec{x}_t), h_{t-1}) \tag{4}$$

As shown in Fig.2, the encoded state vector X_t^i aggregates information of spatial distribution from the SE and the ith available agent location from One Hot Encoder at time t, as shown in Eq.(5) where $\oplus$ is a concatenation operator and Ω symbolizes the One Hot Encoder which converts categorical data into binary vectors with a "1" for the active category and "0"s elsewhere. The purpose of the concatenation operation is to bring together global information $(SE(\overrightarrow{X_t}), SE(\overrightarrow{T_t}))$ and local information $(\Omega(g_t^i))$. This integration provides subsequent neural network layers with a rich set of learn-able features for policy training.

$$X_t^i = SE(\overrightarrow{X_t}) \oplus SE(\overrightarrow{T_t}) \oplus \Omega(g_t^i) \tag{5}$$

The state value $V(s_t^i, \theta)$ can be computed by Eq.(6), where θ is the set of parameters of the TSD-Net. The $V(s_t^i, \theta)$ specifies the expected accumulated rewards starting from state s_t^i. It functions as a 'Critic,' evaluating whether the current state, denoted as s_t^i, is advantageous or detrimental for the agents. This assessment can be leveraged to fine-tune the policy distribution during training, and also provide additional assistance in enabling Autonomous Vehicles (AVs) to prioritize valuable tasks when supply falls short.

$$V(s_t^i, \theta) = W_d * ReLu\big(W_c * ReLu(W_b * X_t^i)\big) \tag{6}$$

Similarly, the action sampling distribution $\pi(a_t^i|s_t^i, \theta)$ can be computed by Eq.(7), where the elements of $\vec{y}$ represent the scores of each action in $\mathcal{A}^i$ and $k = |\mathcal{A}^i|$. This sampling mechanism serves a dual purpose, preventing both the repetition of the same actions by AVs in the same area and the convergence to local optima in the training policy.

$$\begin{aligned} \vec{y} &= W_e * ReLu\big(W_c * ReLu(W_b * X_t^i)\big) \\ \pi(a_t^i|s_t^i, \theta) &= \frac{e^{\vec{y}[a_t^i]}}{\sum_k e^{\vec{y}[k]}} \end{aligned} \tag{7}$$

5.2 Batch Synchronous Actor Critic

The Actor-Critic algorithm (Konda and Tsitsiklis, 1999) is a kind of Reinforcement Learning (Sutton and Barto, 2018) that integrates both policy-base and value-base approaches. The Actor learns an appropriate policy to adapt to the environment while the Critic evaluates the quality of the learning policy. In this paper, we propose the Batch Synchronous Actor Critic (BS-AC) of which the basic idea is to train the set of parameters θ of the TSD-Net with experiences from the Rollout Buffer, such that the unbiased state value function $V(s_t^i, \theta)$ (Critic) and the appropriate policy distribution (Actor) can be learned.

Unlike the Advantage Actor Critic (Mnih et al., 2016) that trains only one agent with all experiences at each iteration, the BS-AC trains multiple agents by sampling a batch of experiences from the Rollout Buffer with replacement. There are two benefits of sampling experience with replacement compared to using the sum of experiences: (*i*) States that are common among agents will take priority such that scenarios corresponding to these states can be learned by agents efficiently; (*ii*) experiences can be reproduced by sampling with replacement such that the number of learning experiences can be larger than the number of experiences in the Rollout Buffer, and sufficient experiences can help agents gain a deeper understanding of their interactions with the environment.

The Critic $V(s_t^i, \theta)$ is learned by minimizing the following loss function $L^c(\theta)$, as shown in Eq.(8) where $r_t^i + V(s_{t+1}^i, \theta)$ is the target value to be learned.

$$L^c(\theta) = \mathbb{E}\big(r_t^i + V(s_{t+1}^i, \theta) - V(s_t^i, \theta)\big)^2 \tag{8}$$

The policy of Actor $\pi(a|s, \theta)$ can be learned by maximizing the following objective function $J^a(\theta)$, as shown in Eq.(9) where $p(s)$ is the probability distribution of state s under policy π_θ and A is the Advantage function. We utilize the Advantage function rather than state value function V because V would introduce high variance, and $A = r_t^i + V(s_{t+1}^i, \theta) - V(s_t^i, \theta)$.

$$J^a(\theta) = \sum_{s \in \mathcal{S}} p(s) \sum_{a \in \mathcal{A}} \pi(a|s, \theta) \cdot A \tag{9}$$

Since our neural network architecture TSD-Net shares parameters between the policy of Actor π_θ and the state value function of Critic V, the total objective function $\mathcal{J}^{a+c}(\theta)$ should combine both $J^a(\theta)$ and $L^c(\theta)$, as shown in Eq.10. The $\mathcal{J}^{a+c}(\theta)$ is also further augmented by the policy entropy $H(\pi_\theta)$ ensuring that agents at the same grid cell can still have diversity movement. Both c_1 and c_2 are the constant value.

$$\mathcal{J}^{a+c}(\theta) = J^a(\theta) - c_1 L^c(\theta) + c_2 H(\pi_\theta) \tag{10}$$

where the policy entropy $H(\pi_\theta)$ is represented by the formula shown in Eq.11. As shown in the equation, when the action probabilities of policy π are similar, the entropy is higher, and vice versa.

$$H(\pi_\theta) = -\sum_{a=1}^{n} \pi(a, \theta) \log(\pi(a, \theta)) \tag{11}$$

The details of the BS-AC training algorithm is shown in Algo.1. First, each available agent executes its action to interact with the environment and store its transition (experience) to the Rollout Buffer $\mathcal{D}$ (line 3-10). Then a batch of transitions $\mathcal{B}$ is sampled from $\mathcal{D}$ with replacement and the accumulated objective $\mathcal{J}$ is computed from the batch (line 11-16). Finally, we update the parameters θ of the TSD-Net by the stochastic gradient ascent and clean up all transitions in $\mathcal{D}$ afterwards (line 17-20).

Algorithm 1: BS-AC Training Algorithm.

1 Initialize a set $\mathcal{D}$ as Rollout Buffer.
2 Create the TSD-Net with random parameters θ.
3 **for** *t = 1 **to** T_{max}* **do**
4   **for** *each agent ag^i **in** A* **do**
5   **if** *ag^i **in** idle mode* **then**
6   Sample an action a_t^i from the TSD-Net under state s_t^i.
7   Execute action a_t^i, observe reward r_t^i and next state s_{t+1}^i.
8   Store transition $(s_t^i, a_t^i, r_t^i, s_{t+1}^i)$ as an experience into $\mathcal{D}$.
9   **end**
10   **end**
11   Sample a batch of transitions $\mathcal{B}$ from $\mathcal{D}$ with replacement.
12     $\mathcal{J} = 0$
13   **for** *i = 1 to $|\mathcal{B}|$* **do**
14   Compute the total objective $\mathcal{J}^{a+c}(\theta)$.
15         $\mathcal{J} = \mathcal{J} + \mathcal{J}^{a+c}(\theta)$
16   **end**
17   Perform the Stochastic Gradient Ascent on $\mathcal{J}$ with respect to θ.
18   Update parameters $\theta = \theta + \alpha \frac{1}{|\mathcal{B}|} \nabla_\theta \mathcal{J}$.
19   Clean up all transitions in $\mathcal{D}$.
20 **end**

5.3 Priority Destination Sampling Assignment

To mitigate extended response times, many prior AMoD systems have employed a First Come First Serve (FCFS) task assignment strategy, particularly within the same grid cell. This approach proves effective when supply meets demand adequately. However, when supply falls short of demand, tasks with destinations in remote, low-activity areas tend to stress the limited supply. Remote destinations necessitate AVs to invest significant time in returning to high-demand regions. If waiting times surpass a customer's patience threshold, they may abandon the service. To address this challenge, we introduce the Priority Destination Sampling Assignment (PDSA) as a solution for guiding agents to select appropriate tasks when supply falls short. The fundamental concept behind PDSA is to prioritize each task based on its state value $V(s_t^i, \theta)$ derived from the TSD-Net. Tasks are subsequently sampled according to their priority values. This approach increases the likelihood of AVs selecting high-value tasks and being dispatched to destinations with high demand. Simultaneously, tasks with less favorable destinations maintain a chance of being served, thus preventing potential service gaps. The sampling distribution is constructed as following:

$$\Pr[\tau_i] = \frac{p(\tau_i)^{\mu}}{\sum_{j=1}^{\kappa} p(\tau_j)^{\mu}} \quad (12)$$

where $\Pr[\tau_i]$ is the probability that task τ_i can be sampled, function p indicates priority of τ_i, which is determined by the order rank of the state value, κ specifies the total number of tasks in grid cell where τ_i locates, and $\mu \in [0,1]$ is a factor suggesting the degree of the priority used. The advantage of the PDSA is that tasks with 'hot' destinations have better chances to be served while tasks with 'cold' destinations will still not be ignored completely.

The details of the PDSA algorithm are shown in Algo.2. First, we still follow the FSFC strategy if the number of agents is greater than tasks (Line 1-2). If not, the priority of each task is computed by its order rank based on the state value from the Critic of the TSD-Net (Line 4-5). Finally, an appropriate task is sampled by probability $\Pr[\tau_i]$ and assigned to an agent (Line 6-9).

Algorithm 2: PDSA Algorithm.

Input: The agent set $A(g_i)$ and the task set $T(g_i)$ at grid g_i

1 **if** $|A(g_i)| \geq |T(g_i)|$ **then**
2   **return**
3 **else**
4   Sort tasks in $T(g_i)$ based on the state value of their destination by descending order.
5   Compute each task's priority by its rank in the sorted set, $p(\tau_i) = \frac{1}{rank(\tau_i)}$.
6   **for** *each agent* ag_i *in* $A(g_i)$ **do**
7   Sample task $\tau_i \sim \Pr[\tau_i]$.
8   Assign task τ_i to agent ag_i.
9   **end**
10 **end**

6 EXPERIMENT

6.1 Experimental Setup

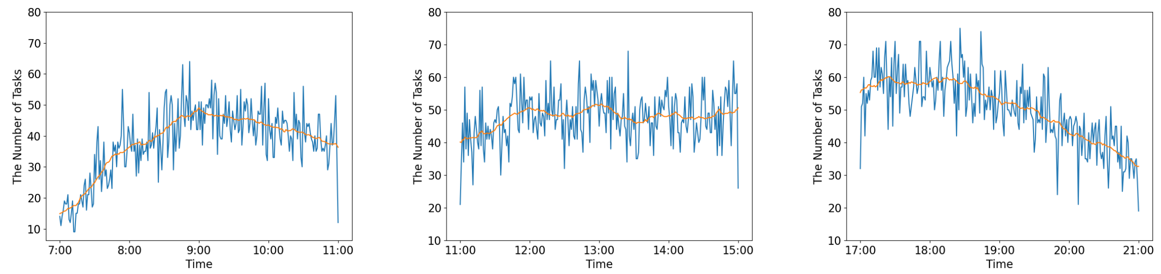
Evaluation Data. Our experiments are based on a real world city-scale dataset from the City of Chicago during September in 2019 (Chicago, 2018), which contains about 1.3 million records. Each record represents as a task in our AMoD System contains a number of attributes: record ID, starting time, origin, destination, trip time and total trip fee.

To evaluate the effectiveness, scalability and robustness of our approach, three time periods are selected, and their time temporal patterns are shown in Fig.3. In the morning (Fig.3a), we observe that the average number of tasks per minute increases steadily to 50 at 9:00AM, peaking at more than 60 tasks per minute, and fluctuating around 45 tasks per minute afterwards. At noon (Fig.3b), the average of the number of tasks seems relatively stable, going up and down between 40 and 50. The evening period is more challenging (Fig.3c). The peak of tasks per minute is greater than 70, and then the amount decreases dramatically after 19:00PM.

The spatial patterns for these three time periods are depicted in Figure 4. In the city's hotspots (Figure 4a), zones 8, 28, 32, and 33 represent the downtown areas, while zone 76 corresponds to the airport. Typically, across downtown areas, the volume of tasks during noon and evening notably surpasses that of the morning, with the exception of zone 28, where the task volume in the morning is slightly higher. This phenomenon is primarily because few businesses and activities occur in downtown areas during the morning. A similar trend is observed at the airport. Conversely, in the city's common areas (Figure 4b), the differences in task distribution across the three time periods within each zone are less pronounced. During the morning, tasks are more evenly distributed. This is because residents from various neighborhoods request AVs in the morning, leading to a more balanced task distribution across the city.

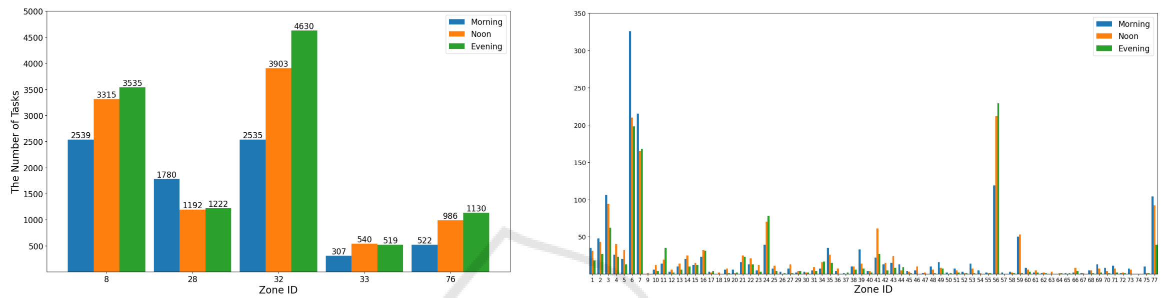
Comparison Methods. Our approach is compared with the following baselines in terms of Service Rate, Response Time, and Repositioning Time.

- **Random.** The available agents stochastically select one of the neighboring grid cells or current grid cell for their relocation. This method serves as the lower bound of the comparison.
- **Soft S-D Heuristic.** Based on the information of task and agent distribution, the Soft S-D heuristic directs AVs to locations which are generally popular, without regard to time of day. This method is robust to the environmental parameter change.
- **AS-DDQN (Zheng et al., 2022).** Using the direction scores obtained from the Double Deep-Q-Network (DDQN), agents sample an appropriate direction to execute. Directions with higher scores are more likely to be chosen. This method can discourage all AVs who are in the same places from making the same decision.



(a) Morning. (b) Noon. (c) Evening.

Figure 3: Temporal patterns of three time periods. The blue line represents real time tasks amount tracked by every minute while the orange line specifies the average tasks amount within 15 minutes.



(a) Downtown and airport areas. (b) Other areas.

Figure 4: Spatial patterns of three time periods. The X-axis represents zone ID of the city, while Y-axis stands for the total tasks amount in each time period separately.

- **Soft cDQN (Lin et al., 2018).** Agents make decisions with output from the Deep-Q-Network (DQN) and the context information of the environment and the deployment of other agents. This method can reduce the dimension of agents' action space and make the learning process effective.
- **Ours.** Our approach utilizes a novel neural network framework, an improved learning process, and a considerable task assignment mechanism to make agents more sensitive and adaptive to the dynamic properties of the environment.

Deployment. The city is partitioned into 77 cells. The simulation cycle is set to be 1 minute and the patience duration of passengers is assumed to be 10 minutes during the comparison. To further challenge the simulation, the initial locations of all agents are remote areas where tasks rarely occurs. In terms of In our approach, the batch size is set to be 1024. The discount factor $\gamma = 0.99$. The learning rate is $\alpha = 0.0005$, and constant value $c_1 = 0.5, c_2 = 0.001$.

6.2 Performance Measurements

In this work, the performance of the AMoD System is evaluated by the following measurements.

Measurement 1: Service Rate. The Served Rate measures the proportion of tasks that can be served by agents in the AMoD System successfully.

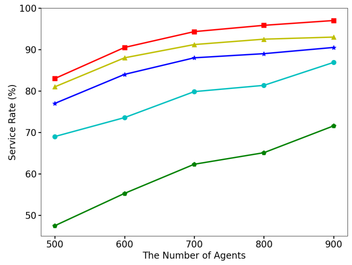
Measurement 2: Response Time. The Response Time estimates the average time duration between rider request and pickup by an AV.

Measurement 3: Repositioning Time. The Repositioning Time gauges the average relocation time between when an agent completes one task and finds another to serve.

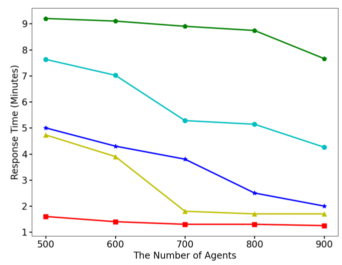
6.3 Performance Comparison

To verify the effectiveness, scalability and robustness of our proposed approach, we compare it with other competitive baseline methods in terms of Service Rate, Response Time and Repositioning Time along with various numbers of agents within three time periods: Morning(7AM$\sim$11AM), Noon(11AM$\sim$3PM) and Evening(5PM$\sim$9PM), as shown in Fig.5. Generally, more agents means more tasks can be handled, less tasks' responsive time while more repositioning time for agents themselves. In particular, several observations are made from Fig.5 as following:

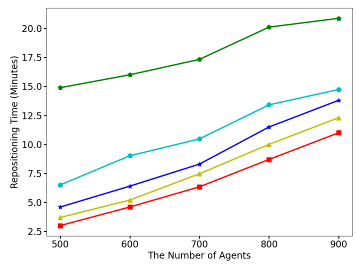
1. Compared to the Random method, the Soft SD-Heuristic attains improved results over all performance measurements. This is because the Soft SD-Heuristic can intentionally diffuse agents to local areas where supply is insufficient for demand.



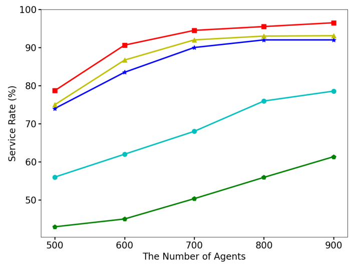
(a) Service Rate (Morning).



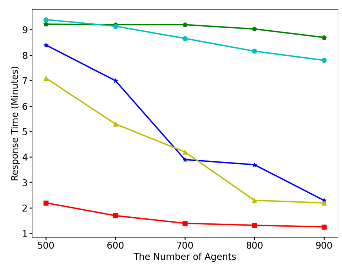
(b) Response Time (Morning).



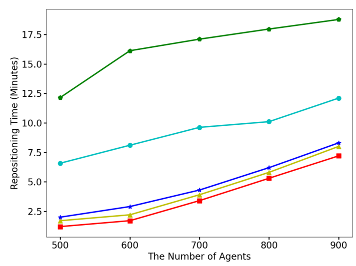
(c) Repositioning Time (Morning).



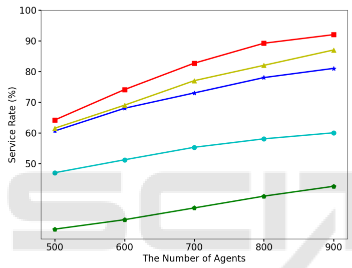
(d) Service Rate (Noon).



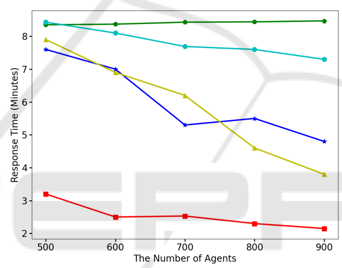
(e) Response Time (Noon).



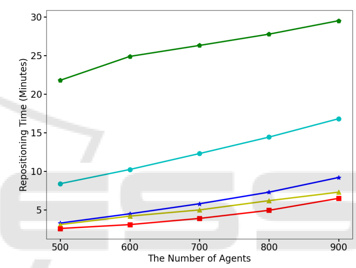
(f) Repositioning Time (Noon).



(g) Service Rate (Evening).

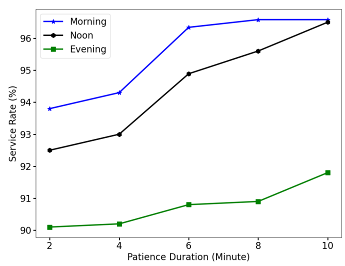


(h) Response Time (Evening).

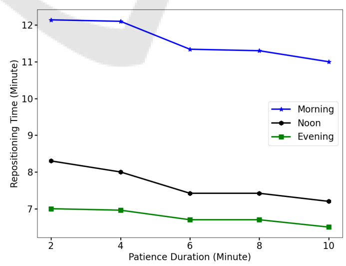


(i) Repositioning Time (Evening).

Figure 5: Performance Comparisons of competitive approaches w.r.t. Service Rate, Response Time and Repositioning Time along with various number of agents within three time periods: Morning(7AM∼11AM), Noon(11AM∼3PM) and Evening(5PM∼9PM). The graph key is as follows: Random:, Soft SD-Heuristic:, AS-DDQN:, Soft cDQN:, Ours:.



(a) Service Rate.



(b) Repositioning Time.

Figure 6: The robustness tests of the Service Rate and the Repositioning Time along with various patience duration of passengers.

2. The learning based methods make significant improvement upon the heuristic method because the learning approaches seek to optimize the long term benefit while the heuristic maintains a myopic focus on the current benefit. Also, the learning based methods consider the spatial distribution of agents and tasks globally while the heuristic only cares about supply and demand locally.

3. With respect to the learning based method, the series algorithms of actor critic outperform the DQN series. There are two reasons for this: (*i*) the Q value approximator of the DQN is not guaranteed to converge while the policy approximator of the

actor critic has better convergence properties; (*ii*) the actor critic algorithms use policy entropy to ensure the diversity of action samples each time, but the DQN will eventually tend to select the action with highest Q value, even though the softmax function is used.

4. In terms of the DQN series, the Soft cDQN is somewhat better than the AS-DDQN, the main reason being that the AS-DDQN uses the rank as a priority that is rigid for the action distribution such that it is difficult to adapt to the dynamic of the environments.
5. We observe that our approach outperforms other baselines.This is mainly because the TSD-Net utilizes the GRU to process the spatial information with temporal signals such that accurate representation features can be learned; moreover, the PDSA can effectively allocate agents to suitable areas by prioritizing tasks with destinations in high-demand areas. This arrangement ensures that agents are consistently in proximity whenever a high-demand task becomes available.

With 900 agents and our approach, the performances of the Service Rate and the Repositioning Time applying various patience durations of customers are shown in Fig.6. In terms of the Service Rate, we observe that the fluctuation in the morning and noon is within 3% while in the evening no more than 1%. On the other hand, for the Repositioning Time, the performance changes over all three time periods are within 1 minute. From this perspective, our approach is robust and adaptive with various patience duration settings.We also observe that Autonomous Vehicles (AVs) spend more time repositioning during the morning hours in comparison to the noon and evening periods. This disparity can be attributed to the following factors: (*i*) In hotspot areas, task demand during noon and evening is notably higher than in the morning, as illustrated in Figure 4a. Consequently, most AVs working during these periods do not need to invest significant time in repositioning, as they continue to have ample opportunities primarily within the downtown and airport regions. (*ii*) Unlike the concentrated task distribution in hotspot areas during noon and evening, task distribution in the morning tends to be more evenly dispersed across the city, as depicted in Figure 4. AVs are compelled to allocate additional time during the morning to reach customers before being assigned tasks. Conversely, tasks in business districts are more prevalent during the noon and evening hours and are typically concentrated in downtown areas. This concentration allows AVs to reduce the time needed to reach these tasks, thereby minimizing repositioning requirements.

7 CONCLUSION

In this paper, there are four contributions to AMoD systems. First, the TSD-Net combines both policy and value networks to save computational cost. It also facilitates the temporal signals behind spatial information to learn representation features. Second, to decrease time needed to collect experiences, the BS-AC algorithm samples experience from the Rollout Buffer with replacement and utilizes Stochastic Gradient Ascent to train the parameters of TSD-Net. Thirdly, based on the state value from the Critic of the TSD-Net, the PDSA algorithm defines priorities of each task and samples appropriate ones for agents. Finally, performance comparisons are conducted to verify the effectiveness, scalability and robustness of our approach.

While the proposed Multiagent Reinforcement Learning (MARL) framework has demonstrated significant performance improvements in the field of AMoD systems, several limitations remain that need to be addressed in the future:

- The TSD-Net considers correlations in terms of temporal patterns but overlooks interactions concerning spatial patterns. For instance, certain grid cells adjacent to high-demand activity areas such as downtown or airports may exhibit sparse demand themselves. However, deploying agents around these areas might prove to be a strategic choice. Leveraging Graph Neural Networks (GNNs) (Sanchez-Lengeling et al., 2021) presents a promising approach to processing spatial information within a graph framework. Future research endeavors should focus on exploring methods to integrate GNNs into the existing TSD-Net, enabling a more comprehensive consideration of spatial aspects.
- In regions with low supply where agents seldom operate, their decision-making can suffer due to insufficient experience to train the policy network. Consequently, satisfying demands in these areas becomes challenging, potentially leading to inefficient behavior by agents. Future research aims to investigate methods enabling machines to generate experiences automatically corresponding to low-supply areas. This exploration intends to facilitate proper training of the policy network, ultimately enhancing the decision-making process in underrepresented regions.

REFERENCES

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Chicago (2018). *Chicago Data Portal: Taxi Trips*. https://data.cityofchicago.org/Transportation/Taxi-Trips/wrvz-psew.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In Moschitti, A., Pang, B., and Daelemans, W., editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Gammelli, D., Yang, K., Harrison, J., Rodrigues, F., Pereira, F. C., and Pavone, M. (2022). Graph meta-reinforcement learning for transferable autonomous mobility-on-demand. *arXiv preprint arXiv:2202.07147*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Konda, V. and Tsitsiklis, J. (1999). Actor-critic algorithms. *Advances in neural information processing systems*, 12.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

Li, J. and Allan, V. H. (2022a). T-balance: A unified mechanism for taxi scheduling in a city-scale ride-sharing service. In *ICAART (2)*, pages 458–465.

Li, J. and Allan, V. H. (2022b). Where to go: Agent guidance with deep reinforcement learning in a city-scale online ride-hailing service. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1943–1948. IEEE.

Lin, K., Zhao, R., Xu, Z., and Zhou, J. (2018). Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1774–1783.

Lin, L.-J. (1992). *Reinforcement learning for robots using neural networks*. Carnegie Mellon University.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

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

Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38.

Sanchez-Lengeling, B., Reif, E., Pearce, A., and Wiltschko, A. B. (2021). A gentle introduction to graph neural networks. *Distill*. https://distill.pub/2021/gnn-intro.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sun, J., Jin, H., Yang, Z., Su, L., and Wang, X. (2022). Optimizing long-term efficiency and fairness in ride-hailing via joint order dispatching and driver repositioning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3950–3960.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12.

Tesauro, G. et al. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.

UN, D. (2015). World urbanization prospects: The 2014 revision. *United Nations Department of Economics and Social Affairs, Population Division: New York, NY, USA*, 41.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2094–2100.

Wang, G., Zhong, S., Wang, S., Miao, F., Dong, Z., and Zhang, D. (2021). Data-driven fairness-aware vehicle displacement for large-scale electric taxi fleets. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1200–1211. IEEE.

Wang, Z., Qin, Z., Tang, X., Ye, J., and Zhu, H. (2018). Deep reinforcement learning with knowledge transfer for online rides order dispatching. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 617–626. IEEE.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR.

Wen, J., Zhao, J., and Jaillet, P. (2017). Rebalancing shared mobility-on-demand systems: A reinforcement learning approach. In *2017 IEEE 20th international conference on intelligent transportation systems (ITSC)*, pages 220–225. Ieee.

Xu, Z., Li, Z., Guan, Q., Zhang, D., Li, Q., Nan, J., Liu, C., Bian, W., and Ye, J. (2018). Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In *Proceedings of the 24th*

ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 905–913.

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