INTERNATIONAL JOURNAL OF CIRCUITS, SYSTEMS AND SIGNAL PROCESSING
DOI: 10.46300/9106.2022.16.80 Volume 16, 2022

Rebalancing Autonomous Vehicles Using Deep Reinforcement Learning

Jiajie Dai*
Yancheng Institute of Technology
1 Xiwang Road, Yancheng, 224007, China

Qianyu Zhu*
Changzhou University
1 Gehu Road, Changzhou, 213164, China

Nan Jiang
Suzhou University of Science and Technology
1701 Binhe Road, Suzhou, 215011,China

Wuyang Wang
Nanjing University of Posts and Telecommunications
9 Wen Yuan Road, Nanjing, 210023, China

*: Both authors contributed to this paper equally.

Received: June 20, 2021. Revised: December 8, 2021. Accepted: January 14, 2022. Published: January 15, 2022.

Abstract—The shared autonomous mobility-on-demand (AMoD) system is a promising business model in the coming future which provides a more efficient and affordable urban travel mode. However, to maintain the efficient operation of AMoD and address the demand and supply mismatching, a good rebalancing strategy is required. This paper proposes a reinforcement learning-based rebalancing strategy to minimize passengers' waiting in a shared AMoD system. The state is defined as the nearby supply and demand information of a vehicle. The action is defined as moving to a nearby area with eight different directions or staying idle. A 4.6 × 4.4 km^2 region in Cambridge, Massachusetts, is used as the case study. We trained and tested the rebalancing strategy in two different demand patterns: random and first-mile. Results show the proposed method can reduce passenger's waiting time by 7% for random demand patterns and 10% for first-mile demand patterns.

Keywords—Deep Reinforcement Learning, Autonomous Mobility-on-Demand(AMoD), Rebalancing, Autonomous Vehicles(AVs).

I. Introduction

THE Autonomous Mobility-on-Demand (AMoD) system represents a new type of mobility services served by Autonomous Vehicles (AV) [1]. It works as current transportation network companies (e.g., Uber, Lyft, DiDi) except for that AVs show 100% compliance to the control center. The AMoD is considered a promising way to make mobility easier for passengers and profitable for operators [2]. On the one hand, the flexible scheduling of driverless cars reduces the waiting time of passengers and also enables operators to obtain more benefits. On the other hand, the learning methods used by the program, including various forms of car sharing, can effectively help consumers reduce travel costs and make travel more affordable.

However, though there are many potential benefits, one of the problems in the AMoD system is addressing the potential imbalance between demand and supply. For example, Fig. 1 shows a single side demand pattern where all passengers move from blue dot to yellow dot. With limited AVs, most of the vehicles will be centered around the yellow dot after dropping off passengers. If all vehicles stay idle when no passengers call them, there will be a high waiting time for new passengers in the blue dot because vehicles need to move a long distance to pick them up, leading to loss of services.

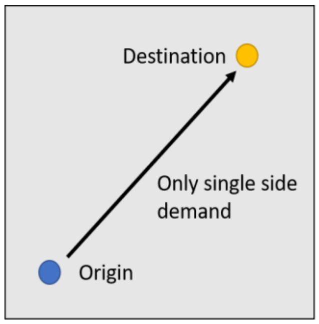

Fig. 1 example of imbalanced supply and demand

Rebalancing, which means moving idle vehicles to a specific area so as to serve the future emerging demand, is a potential way to offset the imbalance. In Fig. 1, the best rebalancing strategy is moving idle vehicles to the blue dot right after they finish a trip. However, in the real-world scenario with diverse demand patterns, how to rebalance idle vehicles remains a challenge.

This paper aims to design optimal rebalancing strategies given various demand patterns using a reinforcement learning (RL) approach to adaptively move idle vehicles. We assume that the AMoD system is operated by the government. So the objective of the control strategy is to reduce passengers' waiting time with limited supplies. The main contributions of this paper are as follows:

- We implement a deep Q network approach for the AMoD rebalance task. Based on the work by [3], we specify the action space, state space, environment, and reward function for the RL algorithm
- We design a case study with various demand patterns to test the performance of the proposed model. We find that the algorithm has better performance for scenarios with more imbalanced demand and supply.

The rest of this article is as follows. The second section reviews related papers in the literature. Section 3 demonstrates the core methodology used in this study, including the deep Q network, definition of action and state spaces, and simulation engine. In the fourth section, we analyze the effect of the proposed algorithms in different demand patterns with a case study in Cambridge, Massachusetts, USA. Finally, the last part of the article concludes the paper and discusses future work.

II. Literature review

The development of machine learning brings various emerging algorithms. In particular, deep neural networks have achieved fruitful improvements in tasks such as computer vision, natural language processing, time series prediction, etc. Reinforcement learning, as a discipline inspired by behavioral theory in psychology, also gains more and more attention.

In 1954, [4] first proposed the concepts and terms of "reinforcement" and "reinforcement learning". In 1965, [5] also proposed this concept in control theory, describing the basic idea of learning through rewards and punishments. They all made it clear that "trial and error" is the core mechanism of reinforcement learning. In 1977, [6] proposed to adapt only to dynamic programming algorithms. In 1989, [7] proposed Q learning and further expanded the application of reinforcement learning.

In the field of urban transportation, the existing rebalancing research work mainly focuses on automobiles, rental systems [8], and public bike-sharing systems [9]. Based on the fluid model, [10] proposed an optimal rebalance model and simulates it with a 12-station AMoD system. However, their method is limited to a simplified station-based network. Besides, they does not consider the interaction between supply and demand and only focus on an ideal balance situation. [11] used a queuing theory method (i.e., Jackson network) by expanding the idea of fluid. They prove that the system is most effective when vehicles' enter and exit rates are similar at each site. The solution provides an offline optimal rebalancing strategy. If the information at each time step can be obtained in real time, their methods can be used for online application.

Recent studies have introduced machine learning approaches for the control of AMoD systems. For example, [3] proposes a reinforcement learning method for the rebalance of the AMoD system [12] used a reinforcement learning model to control the dispatch of the autonomous taxi. This study extends the work from [3] to apply a deep Q network for the rebalance of AMoD systems. Compared to [3], we conduct a more complicated scenario analysis with different demand patterns.

III. Methodology

Suppose we are providing AMoD services to a service region within a certain period of time T. Assume that the regions have been discretized into a set of disjoint zones. We further assume the service time is represented by discrete time intervals Δt.

For all idle vehicles (i.e., vehicles without passengers) at specific time intervals t, we aim to design a model to provide strategies on where the vehicle should move to. This can be done using a reinforcement learning (RL) approach. RL is a learning approach to map states to actions so as to maximize a numerical reward in an unknown and uncertain environment.

Fig. 2 shows the framework of an RL model. At time t, the RL agent receives state (environment) information s_t, and takes action a_t. The action will change the environment and generate a reward r_t to the agent. And the objective of the agent is to maximize the expected total reward:

$$V(s) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t \cdot r_t \,|s_0 = s\right] \tag{1}$$

where $0 < \gamma < 1$ is a discount factor. Equation (1) means that the reward in the future is weighted less than the immediate reward. s is the initial system state.

In a reinforcement learning model, we need to specify the state space, action space, and reward function. In the following sections, we will discuss how the model is specified.

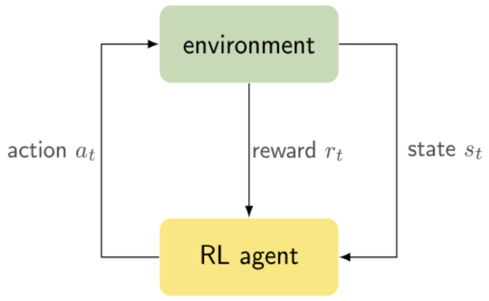

Fig. 2 illustration of reinforcement learning

A. State space

We first divide the service region into a number of equal grids. Define the set of all grids as G. For a specific grid i, the set of its neighboring grids (e.g., 5×5 centered at grid i) is defined as G_i.

Considering a vehicle in grid i at time t, the state vector is defined as:

$$s_{i,t} = \left[\left(b_{j,t}\,, d_{j,t+1}\right)\right]_{j\in G_i} \tag{2}$$

where $b_{j,t}$ is the number of available vehicles in grid j at time t, and $d_{j,t+1}$ is the expected demand (i.e., passenger requests) for grid j at time $t + 1$. Note that $b_{j,t}$ provides the current supply information, while $d_{j,t+1}$ provides the future demand information. In this study, we assume the future demand $d_{j,t+1}$ is known.

Fig 3 shows an example of how the service region is divided and the definition of neighboring grids. In the example service region, there are two vehicles (yellow and green). G_i is the set of all grids in the green (yellow) squares for the green (yellow) vehicle. This means vehicles can get the supply and demand information from nearby areas.

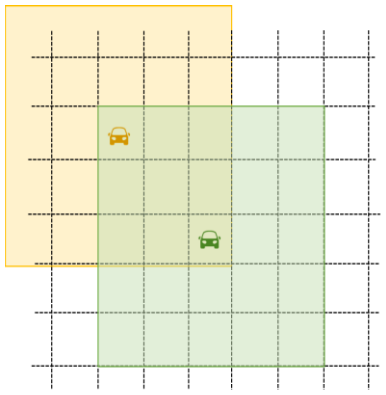

Fig. 3 illustration of state space

B. *Action space*

For an idle vehicle at grid i, we assume it can move to nearby grids or stay at grid i in the next time interval. Therefore, the action space is defined as:

$$A = \{NW, N, NE, E, SE, S, SW, W, O\} \tag{3}$$

for all time step as shown in Fig. 4, where N, W, E, S represents North, West, East, and South, respectively.

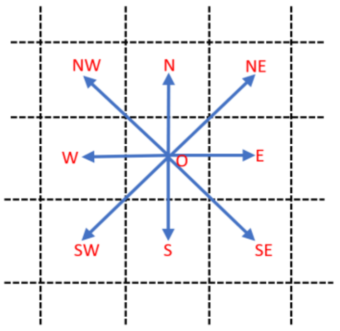

Fig. 4 illustration of action space

With the action space, a reinforcement learning model will output the best action to take given a specific state vector.

C. *Simulation engine*

To simulate the AMoD system, we use the simulation model proposed in [13] for this study. The simulation algorithm is shown in Algorithm 1.

Algorithm 1. Simulation model.

```
procedure Simulation($\boldsymbol{D}, \boldsymbol{V}$):
  initialize the system based on $\boldsymbol{D}$ and $\boldsymbol{V}$
  $\boldsymbol{t = 0}$
  while $\boldsymbol{t < T}$ do:
    generate next passenger request within $[\boldsymbol{t}, \boldsymbol{t + \Delta t}]$.
    add new requests to the request queue.
    update time $\boldsymbol{t = t + \Delta t}$
    move all vehicles up to time $\boldsymbol{t}$
    for each vehicle do:
      drop off passengers if onboard passengers reach the destination
      pick up passengers if the vehicle reaches the origin
      assign "idle" state to if the vehicle is empty
      assign "in-service" state if the vehicle has onboard passengers
    for each request do:
      if exist vehicle satisfying service criteria do:
        assign the nearest available vehicle to the request
    for each "idle" vehicle do:
      let the location of the idle vehicle be grid $\boldsymbol{i}$
      $\boldsymbol{a_{i,t}}$ = DQN($\boldsymbol{s_{i,t}}$)
      Rebalance(the vehicle, $\boldsymbol{a_{i,t}}$)
  return each passenger's waiting time
```

The model simulates the AMoD service within the time period T given the demand information D and supply information V, where V is the maximum number of vehicles in the system. D is the expected number of requests during the time period T. We assume the incoming demand is a Poisson process. That is, over a specific time period, the number of incoming requests in grid i with the destination at grid j follows the Poisson distribution with arrival rate λ_{ij}.

There are two states of a vehicle. If a vehicle is assigned to travelers, it has the state of "in-service". Otherwise, it has the state of "idle" and is available to be rebalanced. For each request, we will evaluate the number of available vehicles using the following service criteria:1) the vehicle has available capacity, 2) the estimated waiting time for the passenger is not too large, 3) the detour rate for on-board passengers is not too large. Passenger wait time is defined as the time difference between a traveler sending out the request and s/he is picked up by a vehicle. When passengers in the queue reach the maximum waiting time, we assume they will forgive the AMoD service and use other modes. In this case, we define an indicator "service rate" to describe the proportion of demands that are successfully served.

Vehicle rebalancing is run every Δt time. The rebalancing action is obtained from the deep Q network (DQN), which we will elaborate on later.

D. Reward function

As mentioned before, we assume the AMoD system is operated by the government with reducing waiting time as their objective. Consider an idle vehicle at grid i, the reward function at time step t is defined as:

$$r_t \coloneqq R(s_{i,t}, a_t) = -[SIM(a_t, s_{i,t}) - SIM(\text{“}O\text{”}, s_{i,t})] - c_r(a_t) \tag{4}$$

where $a_t \in A$ is the action at time t. $R(s,a)$ is the immediate reward function for the vehicle after taking action a under state s. $SIM(a,s)$ is the simulation engine that can return passengers' waiting time for an episode. “O” is the action of no rebalancing. $c_r(a_t)$ is a fixed cost of taking rebalancing action, defined as:

$$c_r(a_t) = \begin{cases} 0, & a_t = \text{"}O\text{"} \\ c, & \text{Otherwise} \end{cases} \tag{5}$$

we use $c_r(a_t)$ to discourage empty-running of rebalancing distance and limit the operational cost.

Eq. 4 indicates that immediate reward is calculated as the waiting time of passengers under DQN strategy minus the waiting time of passengers without rebalancing. Note that when the action of DQN is “O” (i.e., $a_t = \text{"}O\text{"}$), the vehicle remains idle during the rebalancing period, the immediate reward is 0.

The reason for using waiting time difference as the reward, rather than the waiting time itself, is that waiting time itself can be affected by many reasons (e.g., the origin location), we want to extract the “true” effect of rebalancing on reducing waiting time. Hence, $SIM(\text{“}O\text{”}, s_{i,t})$ plays a role of a base model where no rebalancing is used.

E. Deep Q network

The DQN (Deep Q-Network) algorithm was developed by [14]. It was proposed to solve a wide range of games by combining RL and deep neural networks. The algorithm was developed by enhancing a classic RL algorithm called Q-Learning with deep neural networks to represent the Q function.

Q-learning [15] is a typical reinforcement learning method for agents to learn how to act optimally in controlled Markovian domains. In the Q-learning algorithm, the Q-value represents the expected utility of an agent executing the action in the current state. The RL agent intends to select the action with the maximal value of the utility, i.e., the Q-value. The Q-value function for the particular action and state is updated according to the reward received from the environment after executing the action in the current state. For each state and action pair (s,a), the Q-value is initialized as 0 at the beginning of the learning process.

Considering that the current station is s and the vehicle takes action a, it moves to a new grid (or no-move) with a new state s$'$. Then the Q-value is updated based on the following equations:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left(R(s,a) + \gamma \cdot \max_{a' \in A} Q(s',a')\right) \tag{6}$$

where $\alpha \in (0,1)$ is the learning rate.

We define the optimal Q-value ($Q^*(s,a)$) as the maximum return that can be obtained starting from state s, taking action a. It can be shown that this update will make $Q(s,a)$ converges to the optimal Q-value function $Q^*(s,a)$ [14]. And the optimal policy for the vehicle given state s is

$$\pi^*(s) = \underset{a \in A}{argmax}\, Q^*(s,a) \tag{7}$$

For most problems, it is impractical to represent the Q-function as a table containing values for each combination of s and a because of the large number of possible combinations. However, in the DQN, instead, we train a function approximator, a neural network with parameters θ, to estimate the Q-vales (see Fig. 5).

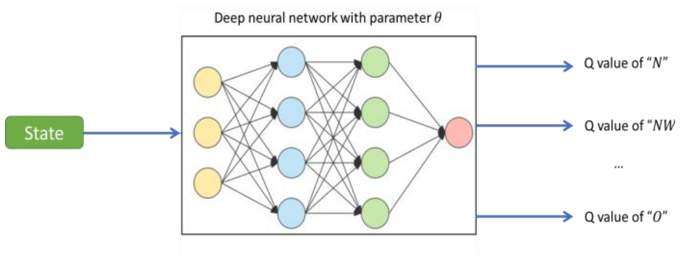

Fig. 5 architecture of DQN

The loss function when training the DQN is defined as:

$$L(\theta) = \sum_{(s,a,r,s') \in \Omega_i} \left(r + \gamma \cdot \max_{a' \in A} Q(s',a') - Q(s,a)\right)^2 \tag{8}$$

where Ω_i is the set of replay buffers at the i-th training epoch. After the training, this Q-value function is used to guide the rebalancing actions of all idle vehicles in the simulation algorithm.

To train a DQN, the simulation engine is run for a period of time. At every time t, the vehicles are rebalanced with a ε-greedy method: with probability ε, vehicles take action $a \in A$ randomly. And with probability $1-\varepsilon$, vehicles take action $\underset{a \in A}{argmax}\, Q(s,a)$, where $Q(s,a)$ is the currently estimated Q-values based on DQN. All the past experience is stored by the user in replay buffers Ω_i at the i-th training epoch. And the DQN is updated with the gradient descent on the above loss function (Eq. 8). The detailed training process of the DQN can be found in [14].

IV. Case study and results

A. *Case study setting*

We choose a 4.6 $\times$ 4.4 km2 region in Cambridge as the case study (Fig. 6). We divide the map into 10 × 10 grids. The AV fleet size is set as 20. The total demand is set as 60 trips/hour. The rebalancing cost is set as $c = 5$. The DQN is trained with a three-layer neural network beforehand with 250 training epochs, using ε-greedy behavior policy and a replay memory of 100 most recent steps in batches of size 32. The learning rate is set to be 0.001 with no decay for the Adam optimizer. Every training process is conducted with 10 replications and the average results are reported. The codes used in this study are modified from [3].

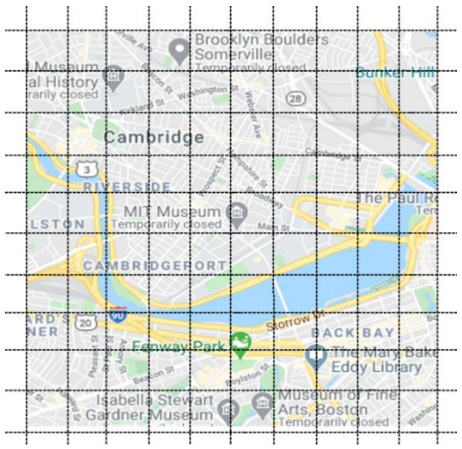
Brooklyn Boulders Somerville
Museum al History
Bunker Hill
Cambridge
RIVERSIDE
MIT Museum
The Paul Re
CAMBRIDGEPORT
BACK BAY
Fenway Park
The Mary Baker Eddy Library
Isabella Stewart Gardner Museum
Museum of Fine Arts, Boston

Fig. 6 case study region.

B. *Simulation scenes design*

We consider two scenarios with respect to different demand patterns: random demand and first-mile (FM) demand (Fig. 6). The random demand is generated by randomly selecting fifteen OD pairs and each OD pair is assigned a flow proportion uniformly ranging from 0 to 1. It represents a uniformly distributed demand pattern. The FM demand is generated by setting a fixed destination in the center of the map (i.e., a subway station) and randomly selecting fifteen origins. Each OD pair is assigned the same flow proportion (i.e., 1/15). Note that the FM demand is considered because AMoD is considered to be implemented as a first-mile feeding service to public transit [16], [17].

These two demand patterns used in this study are shown in Fig. 7. Obviously, FM demand has a larger imbalance between demand and supply because vehicles are easily centered to the destination area (see Fig. 7). Therefore, passengers in FM demand scenarios may have a higher waiting time, and we expect the control strategy works more effectively in the FM demand pattern.

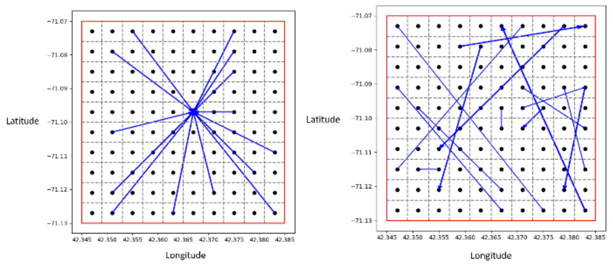

Fig.7: Demand patterns. (Left: FM demand. Right: Random demand)

C. *Results*

Fig. 8 shows the results of the training process for the FM scenario. The average rewards become stable after around 100 episodes (irregular ups and downs are due to randomness in training), showing that the DQN control strategy has converged. The converged reward is greater than 0. Since the reward is defined as the difference in waiting time (see Eq 4), we conclude that the DQN has learned better control strategies than the no-rebalance bassline.

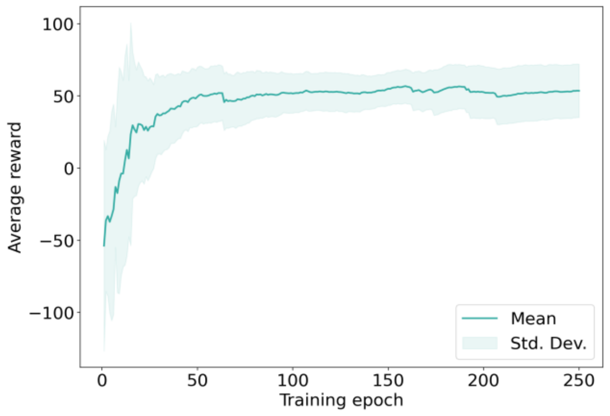

Fig. 8 training process

Tab. 1 compares the performance of the rebalancing methods (DQN vs. No rebalance) under two demand patterns. Besides waiting time, we also report the service rate (i.e., the proportion of passengers being served) and rebalance distance (i.e., the travel distance of AVs for rebalancing purposes). For the FM demand scenario, compared to no-rebalance, the DQN can reduce the waiting time from 6.87 minutes to 6.16 minutes (-10.33%). For the random demand scenario, rebalancing can reduce the waiting time from 5.77 minutes to 5.37 minutes (-6.9%). Both of the scenarios show that the proposed DQN rebalancing strategies can effectively reduce passenger waiting time. It is worth noting that, due to the reduction in passenger waiting time, we also improve the service rate of the system, which implies that more passengers are served with rebalancing strategies given the limited supply. However, the system improvement also brings higher rebalance travel distance because vehicles need to travel more to pick up passengers.

Tab. 1 results comparison

Demand pattern	**Rebalance**	**Avg waiting time (min)**	**Service rate (%)**	**Rebalance distance (km)**
First-mile	DQN	**6.16** (0.24)	**53.8** (1.4)	8.93 (1.4)
	No-rebalance	6.87 (0.01)	53.4 (0.0)	0 (0)
Random	DQN	**5.37** (0.19)	**57.4** (0.7)	7.36 (1.4)
	No-rebalance	5.77 (0.01)	56.8 (0.0)	0 (0)

(The numbers in brackets represent standard deviation across different experiment replications).

V. Conclusion and future research

This paper develops a reinforcement learning-based rebalancing strategy in order to minimize passengers' waiting time in an AMoD system. We use the city of Cambridge in Massachusetts as a case study. Results show that, with the rebalancing using a deep Q network, passengers' waiting time can be reduced, the waiting time reduction is more significant when the demand and supply show higher imbalance (such as the first-mile demand).

Future research can be conducted in the following directions: 1) Due to lack of data, this study does not have real-world demand information. Future studies can test the model using actual demand and supply scenarios to enrich the results. The actual demand can be estimated from models using license plate recognition data [18], [19]. 2) As there are many vehicles in the system, future studies may consider using multi-agent reinforcement learning to discover potential cooperative behavior among different vehicles. 3) The DQN can be compared with other control strategies, such as simulation-based optimization [20]. 4) This study only focuses on reducing waiting time. Future studies can test the model performance in different metrics, such as maximizing social welfare or total profit.

References

[1] B. Mo, Q. Y. Wang, J. Moody, Y. Shen, and J. Zhao, "Impacts of subjective evaluations and inertia from existing travel modes on adoption of autonomous mobility-on-demand," *Transp. Res. Part C Emerg. Technol.*, vol. 130, p. 103281, 2021.

[2] B. Mo, Z. Cao, Z. Hongmou, Y. Shen, and J. Zhao, "Competition between Shared Autonomous Vehicles and Public Transit: A Case Study in Singapore," *Transp. Res. Part C Emerg. Technol.*, 2021.

[3] J. Wen, J. Zhao, and P. Jaillet, "Rebalancing shared mobility-on-demand systems: A reinforcement learning approach," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 220–225.

[4] M. L. Minsky, *Theory of neural-analog reinforcement systems and its application to the brain-model problem*. Princeton University, 1954.

[5] M. Waltz and K. Fu, "A heuristic approach to reinforcement learning control systems," *IEEE Trans. Automat. Contr.*, vol. 10, no. 4, pp. 390–398, 1965.

[6] P. Werbos, "Advanced forecasting methods for global crisis warning and models of intelligence," *Gen. Syst. Yearb.*, pp. 25–38, 1977.

[7] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[8] B. Boyacı, K. G. Zografos, and N. Geroliminis, "An integrated optimization-simulation framework for vehicle and personnel relocations of electric carsharing systems with reservations," *Transp. Res. Part B Methodol.*, vol. 95, pp. 214–237, 2017.

[9] M. Dell, E. Hadjicostantinou, M. Iori, and S. Novellani, "The bike sharing rebalancing problem : Mathematical formulations and benchmark instances The bike sharing rebalancing problem : Mathematical formulations and benchmark instances," *Omega*, vol. 45, no. January 2018, pp. 7–19, 2013.

[10] M. Pavone, S. L. Smith, and D. Rus, "Robotic Load Balancing for Mobility-on-Demand Systems ∗," pp. 0–25, 2012.

[11] R. Zhang, "Control of Robotic Mobility-On-Demand Systems : a Queueing-Theoretical Perspective," *Int. J. Rob. Res.*, 2016.

[12] C. Mao, Y. Liu, and Z.-J. M. Shen, "Dispatch of autonomous vehicles for taxi services: A deep reinforcement learning approach," *Transp. Res. Part C Emerg. Technol.*, vol. 115, p. 102626, 2020.

[13] J. Wen, Y. X. Chen, N. Nassir, and J. Zhao, "Transit-oriented autonomous vehicle operation with integrated demand-supply interaction," *Transp. Res. Part C Emerg. Technol.*, vol. 97, pp. 216–234, 2018.

[14] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

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

[16] B. Mo, Y. Shen, and J. Zhao, "Impact of Built Environment on First- and Last-Mile Travel Mode Choice," *Transp. Res. Rec.*, 2018.

[17] Y. Shen, B. Mo, X. Zhang, and J. Zhao, "Built Environment and Autonomous Vehicle Mode Choice: A First-Mile Scenario in Singapore," 2019.

[18] B. Mo, R. Li, and J. Dai, "Estimating dynamic origin–destination demand: A hybrid framework using license plate recognition data," *Comput. Civ. Infrastruct. Eng.*, vol. 35, no. 7, pp. 734–752, 2020.

[19] B. Mo, R. Li, and X. Zhan, "Speed profile estimation using license plate recognition data," *Transp. Res. Part C Emerg. Technol.*, vol. 82, pp. 358–378, 2017.

[20] B. Mo, Z. Ma, H. N. Koutsopoulos, and J. Zhao, "Calibrating Path Choices and Train Capacities for Urban Rail Transit Simulation Models Using Smart Card and Train Movement Data," *J. Adv. Transp.*, vol. 2021, p. 5597130, 2021.

Contribution of individual authors to the creation of a scientific article

- Jiajie Dai implemented the deep Q network, conducted the case study, and wrote the manuscript.
- Qianyu Zhu proposed the reinforcement learning model and revised the manuscript.
- Nan Jiang implemented the simulation model.
- Wuyang Wang generated the synthetic demand data.