Hindawi
Security and Communication Networks
Volume 2021, Article ID 8830879, 13 pages
https://doi.org/10.1155/2021/8830879

Research Article

An Intelligent Offloading System Based on Multiagent Reinforcement Learning

Yu Weng, Haozhen Chu, and Zhaoyi Shi

College of Information Engineering, Minzu University of China, Beijing 100081, China

Correspondence should be addressed to Haozhen Chu; 595158846@qq.com

Received 18 August 2020; Revised 9 November 2020; Accepted 11 March 2021; Published 25 March 2021

Academic Editor: Luigi Coppolino

Intelligent vehicles have provided a variety of services; there is still a great challenge to execute some computing-intensive applications. Edge computing can provide plenty of computing resources for intelligent vehicles, because it offloads complex services from the base station (BS) to the edge computing nodes. Before the selection of the computing node for services, it is necessary to clarify the resource requirement of vehicles, the user mobility, and the situation of the mobile core network; they will affect the users' quality of experience (QoE). To maximize the QoE, we use multiagent reinforcement learning to build an intelligent offloading system; we divide this goal into two suboptimization problems; they include global node scheduling and independent exploration of agents. We apply the improved Kuhn–Munkres (KM) algorithm to node scheduling and make full use of existing edge computing nodes; meanwhile, we guide intelligent vehicles to the potential areas of idle computing nodes; it can encourage their autonomous exploration. Finally, we make some performance evaluations to illustrate the effectiveness of our constructed system on the simulated dataset.

1. Introduction

With the rapid development of intelligent vehicles, the vehicular network based on artificial intelligence has attracted extensive attention; its wide application has encouraged researchers all over the world to develop more applications, but there is still a problem to compute-intensive services on vehicles; as a promising solution, mobile edge computing (MEC) lets users upload services to edge computing servers (e.g., offloading), which can reduce the computing load of the terminal, just like roadside unit (RSU), building cloud, and other entities with computing [1]. Caching and network communication capabilities can become the MEC platform; they can not only reduce communication delay but also ease the workload of central BS.

However, the performance of traditional methods [2, 3] will decline sharply in the vehicular network; it is urgent to develop an effective MEC intelligent offloading solution. In recent years, machine learning is mainly used in intensive computation tasks, such as navigation and automatic driving, except in MEC; it is very difficult to build a suitable model because many vehicles are participating in the offloading system. Deep reinforcement learning (DRL) uses agents as interactive entities to learn strategies from the environment. DRL has been applied to the MEC offloading system [4], mainly for task scheduling and resource allocation and for studying DRL-based networking and caching [5, 6].

We focus on the tradeoff between the QoE of users and the profit of servers [7]; we need to schedule the corresponding edge computing nodes for intelligent vehicles; this process is similar to the order dispatch in modern taxi networks [8–10]; they provide information about passenger demand and taxi movement for finding the most appropriate pairs; some car-hailing services provide significant improvements over traditional taxi systems in terms of reducing taxi cruising time and waiting time [10, 11]; therefore, the online car-hailing is a vivid scene [8, 12–14] which can be migrated to the edge computing. Xu et al. [15] introduced reinforcement learning algorithms to have foresight, which can achieve better profits for servers and better services for users, but there is a limitation that each server pair with the user in a finite distance, it will put user in an idle state; if this state can be avoided, we can achieve

better. For this reason, we regard our goal as a decision problem for multiagent systems, which is implemented using the classical deep deterministic policy gradient (DDPG) [16] in reinforcement learning.

Furthermore, we introduce a central system with our optimization KM [17]; the algorithm performs better matching between users and servers. The positions of users are determined separately, and users cannot get any information from others at the beginning; we design a communication module referring to [18]; then, they can share information beneficial to the next decision.

The remainder of the paper is organized as follows. We provide a brief overview of the background and related works in Section 2, we will show the system architecture in Section 3, Section 4 describes improvement and algorithm description, Section 5 explains the experimental details and results, and finally, in Section 6, we summarize all the work.

2. Related Works

This paper proposes a multiagent reinforcement learning algorithm with global scheduling and applies it to the scene of intelligent offloading about edge computing. The following content introduces some related researches and the application of a multiagent decision algorithm.

An approach is proposed in [19] to find the optimal auction for computation resources of edge computing in blockchain networks; it uses a monotone transform function to anonymize prices. To decrease the number of duplicated contents in networks, a DRL method for caching in smart cities is designed in [20]; the agent in the system collects the status from MEC servers and learns to choose the optimal action to get the best policy for resource arrangement. Qi et al. [21] constructed an intelligent offloading system for vehicular edge computing by leveraging deep reinforcement learning; its communication and computation states are modeled by finite Markov chains, task scheduling, and resource allocation. The strategy is formulated as a joint optimization problem to maximize users' QoE.

Classical scheduling algorithms, such as greedy methods, are widely used in large companies, such as finding the nearest driver to serve customers [22], or using a first-in, first-out queue strategy [23]; although they are easy to dispatch, it only obtained nice profits in the short term; the spatiotemporal sequence does not match the supply-demand relationship in the long-term operation, which will lead to some suboptimal results [15]. Later, this dispatching process improved by using the central system through the taxi GPS trajectory and brute force method for the best path recommendation [24, 25], considering whether the driver took the initiative to find hot spots to provide the scheduling strategy [11] and focusing on minimizing total customer waiting time by simultaneously scheduling multiple taxis and allowing taxis to exchange their booking tasks [9], taking into account the overall benefits of a more global and far-sighted approach [26]. These methods have been put into practice and have shown valuable effects.

The vehicular network is often divided into several zones, each zone has one BS with abundant computation resources, and the BS can play the role of the central system. We refer to the work of [15] in global scheduling, which constructs a set of preference functions and calculates the corresponding different agents in each time slice with a global view. The average preference function in the region is conducive to the formulation of the reward and punishment process in the subsequent multiagent decision algorithm. We have adopted KM algorithm in order matching, which has a wide range of applications in the dispatch, network communication, system architecture, etc. [27], which is usually used for minimum weight matching [28]; however, it can also be achieved by setting up negative samples to maximize the sum of weights [29]. In this model, we use the KM algorithm, taking advantage of preference function, as a metric to form a strategy for node scheduling.

The multiagent intensive learning aim includes the learning stability of a single agent and the adaptability to the behaviors from other agents [30, 31]. Adaptability ensures that performance can be improved when other agents change their policies [32]. Some aims can be extended to dynamic games by requiring a phased satisfaction condition for all states of the dynamic game. In this case, the aim is based on the stage strategy rather than the global strategy and the expected return rather than the reward [18]. Multiagent systems have been extensively studied in various fields, such as robotics teams, resource management, distributed control, games, e-commerce [33], and several MARL algorithms that have been discussed, indicating that these algorithms combine time difference RL with game theory solvers [34, 35]. For static games generated in the state of dynamic environments [36], Alibaba also uses its system in the environment of a multiscenario e-commerce system to give priority to recommending products of interest to customers and obtain the maximum benefit of the system [37, 38].

In general, the research on multiagent reinforcement learning problems is complicated and computationally intensive. However, this model adds many idealized constraints and excludes irrelevant factor variables, such as data transfer rate and propagation delay. Simplified model is beneficial to implement our reinforcement learning algorithm.

3. General Architecture

The vehicle network in the city can be divided into several zones according to streets or other criteria; each zone has one central BS with abundant computation resources, as shown in Figure 1. We mainly analyze one of these areas; RSUs are equipped with MEC servers and have their signal coverage; therefore, intelligent vehicles can only upload services within a certain range of RSUs. In the case of no RSU nearby, the vehicle can upload its services to the BS directly, but in this model, we only take into account the matching relationship between vehicles and RSUs, because our goal is to schedule the edge computing nodes.

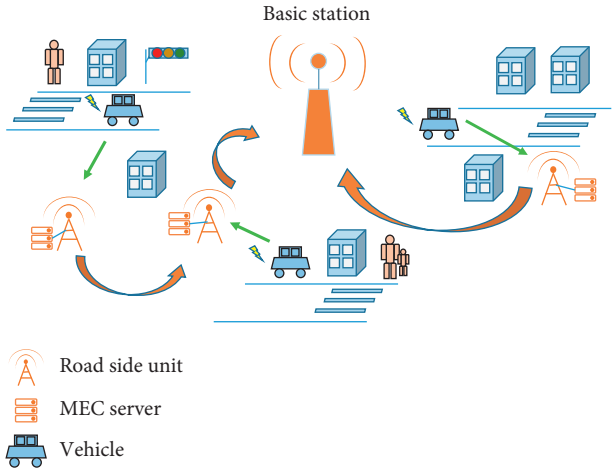

Figure 1: The architecture of MEC-based vehicular network.

Here, we introduce the model design and matching strategy of advantage algorithm at first; they are the basis of our work, but the advantage algorithm cannot cover some special situations. Inspired by [6], we have proposed our architecture to improve it.

3.1. Model Design and Matching Strategy. If we simplify the model for reinforcement learning, an intelligent vehicle can be regarded as an independent agent; its mobile strategy can be abstracted as a Markov decision process (MDP).

The model here refers to the toy example used in [18]; we did not set a more realistic vehicle movement and increase the grid size; because the actual model will consider too many details, it is difficult to reproduce a reasonable model and apply our algorithm; the algorithm compared in [18] is trained on the toy example; then, it was applied to a more complex simulation environment; we also train through a simple model and verify the advantages of our proposed algorithm. We build a simulation model (Figure 2) corresponding to each spatiotemporal state; it can show the movement strategy of the agent in each state.

Matching strategy uses a preference value as the weight between the vehicle i and the RSU j, which is calculated as the value function of the vehicle in the future state (when its service offloaded to the RSU j has finished) minus the value function of the vehicle in the current state; we call this formula of weight as the calculation of preference [17]; call it as an advantage function:

$$A_\pi(i,j) = \gamma^{\Delta t_j} V\left(s_{ij}'\right) - V\left(s_i\right) + R_\gamma(j). \tag{1}$$

Our second goal is to maximize the QoE of vehicles while guaranteeing the profits of network operators (i. e., RSUs). The network operators are in charge of the wireless access network and the mobile core network; the overall profit from finished services will be affected by the amount of transferred data and energy consumption of calculating in reality, which is not involved in our model; for the sake of simplicity, we use some random distributions to set the certain profit at each RSU instead of that; here, we use $R_\gamma(j)$ to represent.

The preference function can generate a corresponding value through (1) to determine the best matching between each vehicle and the RSU; the matching rules are

$$\begin{aligned} &\arg\max_{a_{ij}} \sum_{i=1}^{m}\sum_{j=1}^{n} A_\pi(i,j)a_{ij}, \\ &s.t. \quad \sum_{i=1}^{m} a_{ij} \le 1, \quad j = 1,2,3,\ldots,n, \\ &\qquad \sum_{j=1}^{n} a_{ij} \le 1, \quad i = 1,2,3,\ldots,m, \end{aligned} \tag{2}$$

where a_{ij} depends on the pairing relationship. If RSU j is matched with vehicle i, then a will be 1, i presents all the unmatched vehicles of this time slot, and j means that these RSUs can be allocated. During the RSU allocation phase, vehicles that are not matched to RSUs will be treated as if they were waiting to enter the next time slot. Its final effect is shown in Figure 3.

Moreover, RSU can only be dispatched to one vehicle within 2 grids (as shown in Figure 4), which can reduce the loss of profits caused by the partial allocation and also facilitate the timely response to more vehicles.

However, there is a special situation in this model. If there is no valid RSU in the area when the distribution of RSUs is too sparse, the central system will not allocate a RSU for any vehicle. This leads to the idle state, but actual vehicle can explore around to seek for some RSUs which have stronger signal source during this time.

3.2. Modules and Intermodule Connection. We design an overall architecture based on [15] to avoid the idle state; it analyzes areas having more RSUs in the future and guides a route for vehicles which are in the idle state; they can rush to potential high preference areas in advance; this way will lead to high QoE of users and the well profit ultimately. The analysis of the architecture will refer to historical memory and strategies from different agents.

To ensure that the parameters of the network can finally converge stably, our target network modules are constructed according to the DDPG algorithm. The loss function of the value network narrows the gap between the two value network modules and soft updates the parameters of the target value network. The same update method also applies to the parameter update of the target policy network.

He et al. [6] applied the reinforcement learning to the multiscene recommendation system of e-commerce [39], each scene is regarded as an agent; the different scenes are arranged by the respective strategies of the agents; its center system works as a critic to evaluate the profits from overall scenes. Our model requires adjustments to the driving path of different agents to get better overall profit; this point is very similar to our model.

Inspired by [6], we introduce actor-critic to our architecture with DDPG; an algorithm named MARDDPG (multiagent recurrent deep deterministic policy gradient) is proposed, refer to Figure 5.

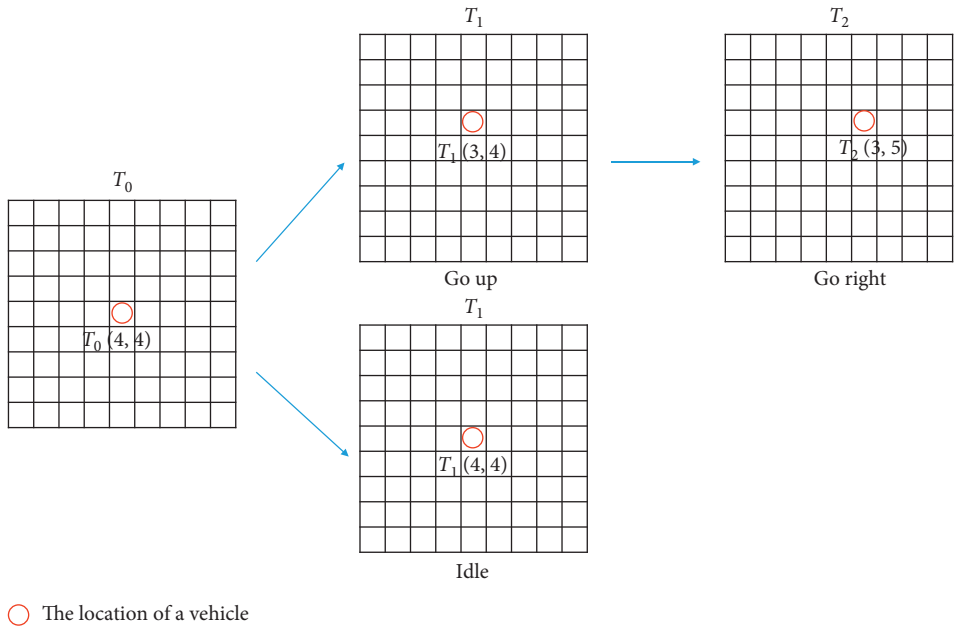

Figure 2: The 9×9 grid of the simulation model; the vehicle can choose to go in one direction or stay in place.

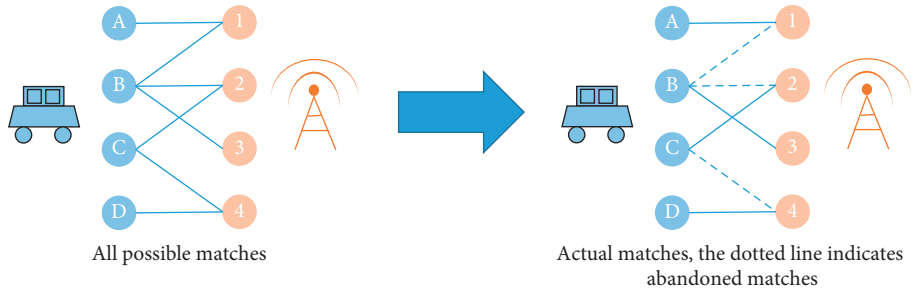

Figure 3: Matching the vehicle to the RSU by KM algorithm.

There are three important modules designed to the collaboration of multiple agents, respectively, a global critic module, independent agents, and a communication mechanism.

3.2.1. Actor Module. Each agent actor$_i$ in the group module called Actor is a separate actor module that accepts the local observation o_t^i such as the location and the shared message m_{t-1} from the previous moment and chooses one action. The behaviors of vehicles are a finite set of discrete actions: up, down, left, and right; we define the behavior variable as a preference probability of different behaviors at that moment:

$$p_i = \left(w_1^i, \ldots, w_{n_i}^i\right), \; p_i \in R_i^n,$$
$$\text{s.t.} \quad \sum_{j=1}^{n_i} w_j^i = 1. \tag{3}$$

Therefore, each behavior is a n_i dimension vector; this vector will eventually select the corresponding behavior with the highest probability of preference as the next behavior of the agent.

Inspired by DDPG-related work, a method of determining the strategy is used instead of a random strategy. The actor$_i$ of each agent corresponds to the function $\mu_i(s_t; \theta_i)$, where the parameter is θ_i, and the function maps a state to a behavior. At time t, the agent decides its behavior according to the actor$_i$ network:

$$a_t^{i_t} \approx \mu^{i_t}\left(m_{t-1}, o_t^{i_t}; \theta_{i_t}\right), \tag{4}$$

where $s_t \approx \{m_{t-1}, o_t\}$ represents the approximate global state; the behavior of actor$_i$ depends on both the message m_{t-1} and its current observation o_t.

To be exact, each agent here corresponds to the vehicle in the idle state of the forecast period, and the period during which the vehicle whose service is still being computed will not be taken into account; each agent i will be according to its strategy $\mu_i(s_t)$ each time the behavior a_t^i is chosen, and then an immediate reward $r_t^i = r(s_t, a_t^i)$ was obtained from the environment. After all the observations and rewards of the agent are accumulated, the state changes from $s_t$to update to s_{t+1}.

3.2.2. Critic Module. The purpose of multiple agents is to achieve the global maximum profits. We pass a global critic

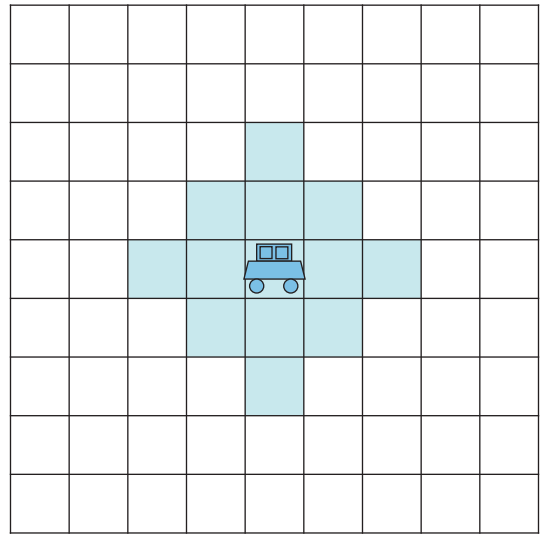

Figure 4: The request range (grid covered by blue) from one smart vehicle.

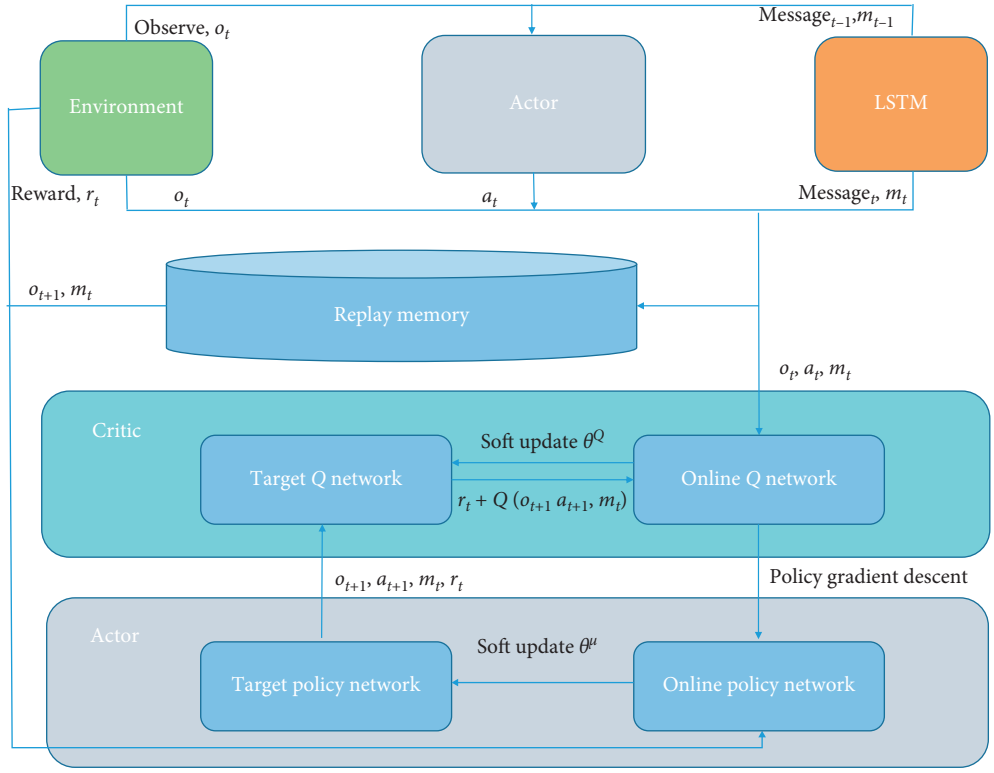

Figure 5: An illustration of the proposed architecture.

behavior value function $Q(s_t, a_t^1, a_t^2, \ldots, a_t^n)$ to evaluate the overall profit; each agent performs a local behavior after obtaining local observations.

A critic network is designed to fit the behavioral value function, which is used to assess the impact of overall behavior on future expectations when taking actions. Because all agents share message, we use a global evaluation function for

$$Q = \sum_{j=1}^{n} q_j, q_j = f\left(s_t, a_t^j; \varphi\right). \tag{5}$$

Each q_j represents the assessment of the behavior of each agent in the global state by the critic network, which ultimately needs to be summed to form a total assessment of agents. The details of the actor-critic network inside an agent are shown in Figure 6.

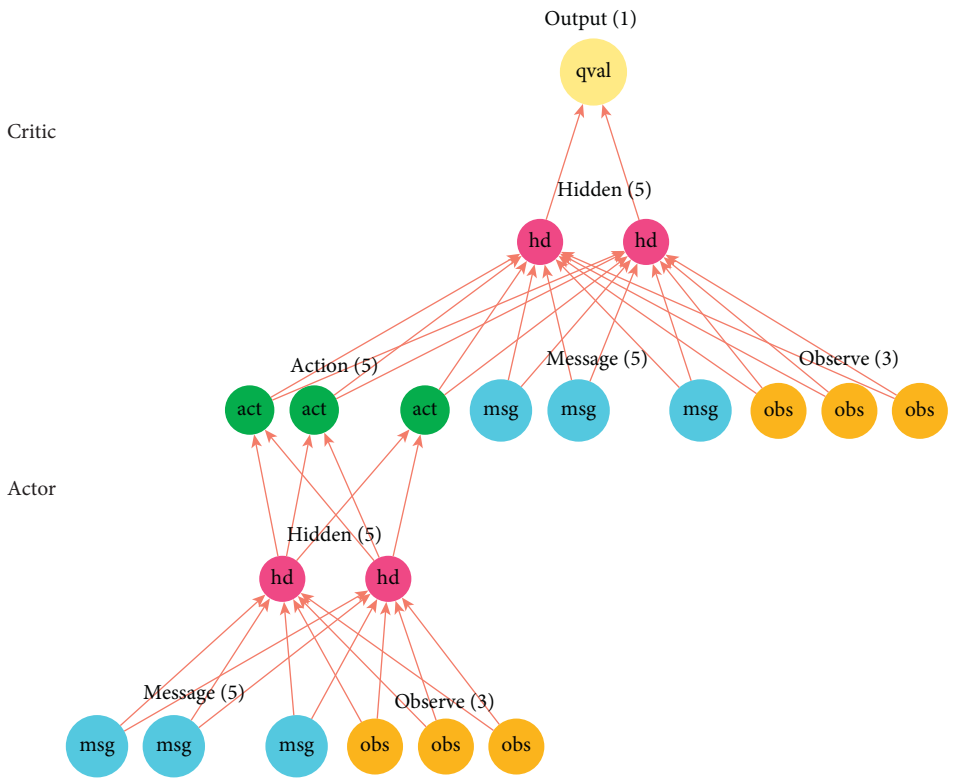

Figure 6: An actor-critic network inside an agent.

The local observation corresponds to a three-dimensional vector, the five-dimensional memory information vector of the previous moment is used as input, and the five-dimensional motion vector is output and combined with local observation as input, finally the critic network outputs one-dimensional evaluation vector, the middle hidden layers are all consists of 5 neurons.

3.2.3. Communication Mechanism. In our communication mechanism based on long-short-time memory network (LSTM), we refer to the setting of LSTM in [18]; the author applies it to the multiscenario ranking task, different scenarios are regarded as an agent, the LSTM encodes all local observations and actions of all agents into a message vector, and the message will be sent between agents. We input all observations and behaviors of all agents into the same communication module simultaneously; it will generate a message vector that memorizes the global information of the current moment. The vector will be sent to the actor module of the different agents at the next moment and form a new input together with their observation information, which has an effect of cooperating. The message m_{t-1} is updated in the communication module, which memorizes the observation o_t and behavior a_t of different agents at the same time.

Because of this mechanism, each agent's decision is based not only on its state and previous behavior but also on the state and behavior of other agents. This communication mechanism gives agents the ability to approximate the state of the global environment, allowing them to make more global and long-term decisions. Then, the o_t, a_t, and m_{t-1} of all the agents at each moment are stored in the playback memory area and transmitted to the global critic module Critic. The online behavior value function $Q(m_{t-1}, o_t, a_t)$ evaluates the effect of the behavior a_t when each agent accepts the message m_{t-1} and the local state o_t.

4. Improvements and Algorithm Description

4.1. Optimized Preference Function. The preference function (1) is only used as a weight for matching. We will calculate the average preference value from one region as improved preference function; it reflects the appearance probability of valid RSUs. A larger value means that this region will have more services which can produce higher overall profits; the system will recommend vehicles rush to this region in advance. This behavior recommendation will reduce the amount of vehicles which are in the idle state; even if there is no valid RSUs, the matching strategy cannot be performed; the agent will actively seek for RSUs. The formula of improved preference is as follows:

$$F_{x,y} = \sum_{j=1}^{n_i} \frac{A_\pi\left(\text{order}_j, \text{car}_i\right)}{n_i}, \tag{6}$$

where n_i is the number of RSUs that had finished the services within a certain distance of the agent i; A_π is based on a preference function (1). Later, we call $F_{x,y}$ as the preference density.

Replay buffer stores data about (m_t, o_{t+1}, r_t) from every episode at each time step; the policy network randomly extracts the sample data from the buffer in the train phase,

and thereby the agent can choose far-sighted action by the past message from other agents, observation, and reward; this way can remove correlations in the observation sequence and smooth over changes in the data distribution. Value function in the target network is calculated by the sum of the immediate return r_t. r_t represents the reward and punishment value; they are obtained by the interaction between the agents and the environment; we define it as the future potential comprehensive income of an agent; whether it is high or not is totally judged by the average value of the preference function; r_t^i is defined as follows:

$$r_t^i = \begin{cases} -1, & F_{x_t,y_t} > F_{x_{t+1},y_{t+1}}, \\ 0, & F_{x_t,y_t} = F_{x_{t+1},y_{t+1}}, \\ 1, & F_{x_t,y_t} < F_{x_{t+1},y_{t+1}}. \end{cases} \tag{7}$$

The preference density of this new area is lower than the original, and a penalty value of −1 is assigned to discourage the exploration of the region in the state s_t; similarly, if the preference density does not change, then to reduce the unnecessary exploration of the agent, which is better than the case of exploring the low preference density, we assign a value of 0, namely, neither encourage nor suppress it; Ultimately, exploring high-density areas is our desired aim with a reward value of 1.

4.2. Optimized Matching for Vehicle Dispatching

4.2.1. Problem Description. The traditional-KM algorithm is suitable for the exact matching of bipartite graphs. Because our model only allocates within a certain range, there is likely a mismatch to the existing bipartite graphs (exact matching: bipartite graph in the case where the left node can match the right node one by one), for example, there is a single area overlap phenomenon (as shown in Figure 7). If only this RSU exceeds the request range of vehicles, it cannot be matched; the traditional-KM algorithm has been modified, because we do not know the lower limit of the worst case of $A(i, j)$ in reinforcement learning; we cannot simply set the negative weight to 0; we need a considerable negative weight to identify the pairing situation in which the RSU is unable to match the vehicle; in the extreme case, the RSU with negative weight cannot be discarded; otherwise, it will affect the satisfaction of users.

A new problem arises with negative weight, when the number of matching on both sides is inconsistent; according to the classical KM algorithm, the nodes with a small number are preferentially matched to ensure that they can be matched, but in the actual matching process, if a considerable negative weight is taken into account in the matching, a lot of time about updating will be wasted; we need to avoid it as much as possible, but it will leave some cases (Figure 8): the edges of the negative weight (non-responsive) are omitted in the figure. When there is a situation as shown in Figure 8, the matching between vehicle c and 2 is better, but in the traditional-KM algorithm, since vehicle b cannot find a new matching in a short time, a loop is stuck in a long calculation.

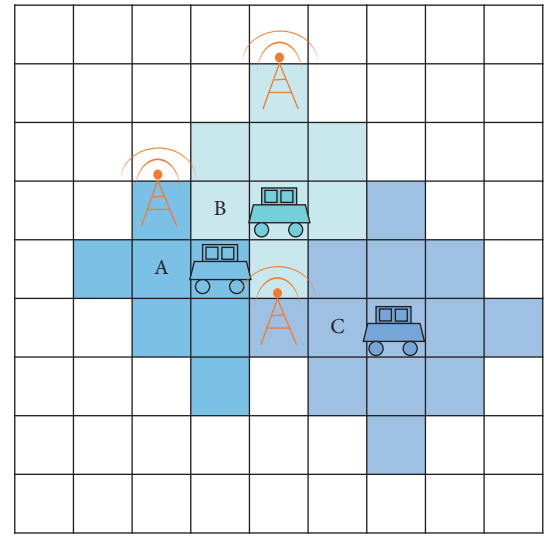

Figure 7: The overlapped answer range of three vehicles.

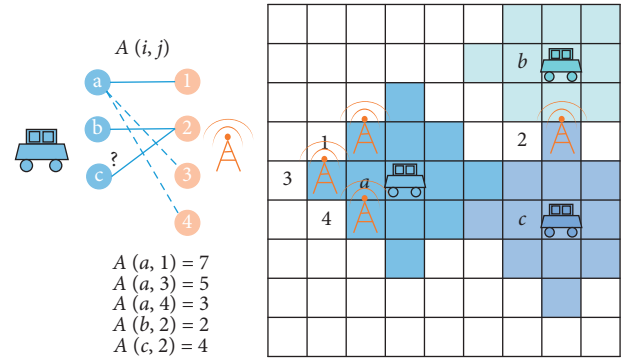

Figure 8: Potential matching situation; it may imply a better match as a to 3 or a to 4.

There will be a more suitable matching; nodes with a higher sum of overall pairing weights will be excluded from the existing matching situation. This requires us to further optimize the existing matching strategy.

We introduce the concept of the loser; we call it the loser-KM algorithm. Once the situation in Figure 8 appears, the vehicle will be eliminated immediately, and then all the vehicle nodes that are eliminated will be collected to Loser$\{\text{loser}_0, \ldots, \text{loser}_n\}$; loser_i will be given one more chance to challenge the vehicle who has already matched the RSU j. If it is greater, the corresponding RSUs will be collected in the master node set Challenger$\{\text{challenger}_0, \ldots, \text{challenger}_n\}$, and finally, the process of attacking will use the classical KM algorithm. The loser-KM algorithm can be used to deal with bipartite graphs that need to consider negative weight matching, and there are some unmatchable conditions. See Algorithm 1 for details.

4.3. Multiagent Recurrent Deep Deterministic Policy Algorithm.

By calculating the regional preference function, the agent i chooses an action by its behavior decision based on its approximate global state s_t; it moves to a new area;

Input: calculate all vehicle-RSU pair matches using KM algorithm and the weight of vehicles pairing with RSUs' weight; record the vehicle nodes that are not matched as the loser set.
Output: updated pairs match
(1) Collect matched orders $j \in [1, \ldots n]$
(2) Initialize a directory challenge, arrays attacker, and challenger
(3) Travel across all the weights between Loser i and RSU j:
(4) If weight [Loser [i]] [j] > weight [match [j]] [j], then
(5) Challenge [Loser [i]] [j] = weight [Loser [i]] [j]]−weight [match [j]] [j]
(6) If Loser [i] is not in attacker, then
(7) Attacker push (Loser [i])
(8) End if
(9) If j is not in attacker, then
(10) Challenger push (Loser [i])
(11) End if
(12) End if
(13) Initialize a two-dimension array chaWeight
(14) Check if challenger j is in the challenge list of attacker i:
(15) If challenger [j] is not in Challenge [attacker [i]], then
(16) Challenge [attacker [i]] [challenger [j]] = 0
(17) End if
(18) chaWeight [i][j] = Challenge [attacker [i]] [challenger [j]]
(19) Calculate attacker-challenger pairs chaMatch with chaWeight by using KM algorithm
(20) For i = 1:challenger, do
(21) Match [challenger [i]] = attacker [chaMatch [i]]
(22) End for

Algorithm 1: Loser Kuhn–Munkres Algorithm.

central system uses the loser-KM matching algorithm to determine a set of scheduling strategies. Since the regional preference takes historical information into account, it encourages vehicles to go to the area where the RSU likely takes services in advance; both the response rate and the total profits have improved in the end.

Then, we define the network update rule; we try to minimize the difference between the calculated behavior value and old value in online Q-network; correspondingly, due to the characteristics of the DDPG network, the value of weight in target Q-network is the value in the last training. Because of the existence of the target network, the update of weight is delayed; it makes the training more convergent and the network more stable. We refer to the settings of the loss function in [6].

The detailed process is shown in Algorithm 2.

5. Experiments

We designed our experiments to investigate the following questions:

(1) How to reflect the role of the improved km algorithm and improve the performance of matching?

(2) MARDDPG is related to prior methods but makes several changes; how does it compare with others when applied to the same simulated environments, with our experimental metric?

To answer (1), we compare the performance on the average profits in a round of matching between the traditional and improved km algorithm. About (2), we show that both the results of traditional and MARDDPG under the same RSU and different numbers of vehicles, it is used to compare different algorithms and highlight the performance under different sparsity levels of the agent. The use of different metrics is to show the performance of each algorithm, it is difficult to finish this matching problem in limited time, it is obvious that each algorithm has its limitations and merits.

5.1. Implementation Details. To verify the reliability of the algorithm, we designed the entire dispatch process to be executed in a 9×9 grid, and for 20-time steps, uniform finite time and small space can effectively reduce the interference of external noise, such as cross-regional and cross-day information transmission; we simplify the behavior of vehicles, each vehicle can only stay in one-time spot, or do a horizontal/vertical move.

While setting the dispatch distance to 2, the vehicle can only upload services to the RSU not exceeding this distance range. If a RSU has not been requested to any vehicle for a long time, it will be canceled, and the cancellation time is set to a range from 0 to 5 of the truncated Gaussian function; its average is 2.5, and the standard deviation is 2.

The RSU generation model also simulates the morning and peak traffic patterns of commuting and the situation with the residential area; the RSU position uses a two-component mixed Gaussian to generate x- and y-axis coordinates and truncates them into integers in the grid. The initial positions of vehicles and RSUs are generated by a discrete uniform distribution function.

Input: the environment
Output: $\theta = \{\theta^1, \ldots, \theta^N\}$
(1) Initialize the parameters $\theta = \{\theta^1, \ldots, \theta^N\}$ for the N actor network and ϕ for the critic network
(2) Initialized the replay memory M
(3) For training step = 1:all steps, do
(4) m_0^0=initialized message, $t = 0$
(5) While $t < T$, do
(6) For $i = 1{:}N$, do
(7) Select the action $a_t^i = \mu_t^i(m_{t-1}, o_t^i)$ for agent i_t
(8) Receive reward r_t^i which is calculated by regional average preference function
(9) Receive observation o_{t+1}^i
(10) Update message by $m_{t-1}^i = \mathrm{LSTM}(m_{t-1}^{i-1}, [o_t^i, a_t^i])$
(11) End for
(12) $m_t^0 = m_{t-1}^N$
(13) $t = t + 1$
(14) End while
(15) Store episode $\{m_0, a_1, r_1, m_1, o_2, a_2, \ldots\}$ in M
(16) Sample a random minibatch of episodes from replay memory M
(17) Each episode and each time, we do
(18) Update the critic, actor, and LSTM network by minimizing the loss
(19) Soft-update the target critic and target actor network
(20) End for

Algorithm 2: Multiagent recurrent deep deterministic policy algorithm (MARDDPG).

5.2. Improved Performance on Matching. Our algorithm is compared with the distance priority algorithm, the profit priority greedy algorithm, and the advantage algorithm. Each algorithm uses the original KM version and the loser-KM version to match these RSUs so that we can observe the impact of existing algorithm factors; in the bipartite graph, the weight of the distance-first algorithm is based on the distance between the RSUs and the vehicles; the weight of the profit priority is based on the calculation of services with the highest profit, and the algorithms combine with the two preference matching strategies; we call them as advantage algorithm whose name comes from advantage function and MARDDPG.

We use average profits at first, namely, the profit obtained by the RSU on each vehicle as a metric for the improvement of the strategy algorithm, because traditional KM in the advantage algorithm uses the preference function as the weight, compared with our improved KM matching algorithm, which has become an adaptation of MARDDPG; it takes the regional preference as the weight, which means that once the vehicle reaches the area with a higher preference value, the profit obtained will be higher so that average profits will be higher because it focuses on long-term returns compared to advantage algorithm.

As shown in Figure 9, each bar represents the average content of 20 independent experiments. Here, we uniformly take the ratio of RSUs to vehicles, which is 100 : 25, as the experiment environment. We are not listing the distance algorithm because it is unrelated to the matching strategy. With using the loser-KM algorithm, the average profits are higher than the traditional-KM algorithm, which is also due to the loser-KM which is based on the principle of the traditional algorithm; its improvement of performance is made of the change according to the actual problem of

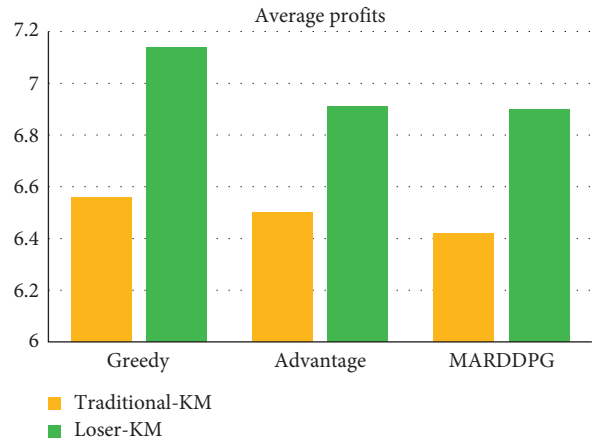

Figure 9: The comparison between traditional-KM and loser-KM in average profits.

negative weight with an unknown lower limit; therefore, the loser-KM algorithm used in the RSU allocation process contributes to certain performance improvement.

Following that, we use four strategies compared with four metrics (average profits, overall profits, pickup distance, and response rate) under the condition of 100 RSUs and the number of vehicles is 25, 50, or 75, respectively, with ten independent experiments.

We compare the performance of average profit in Figure 10; when the ratio of RSUs to vehicles is lowest, the average profits tend to be high; because the overall number of finished services is relatively small, average profits will decrease first and then slowly rise with the ratio increasing. The performance of distance algorithm in the average profit is not outstanding, when the vehicles density increases; its

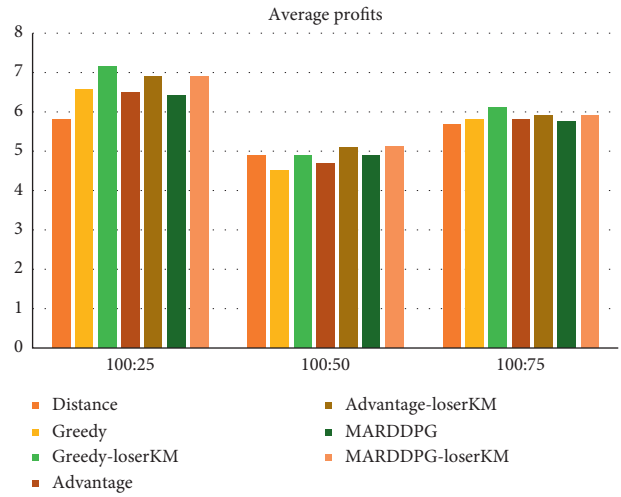

Figure 10: The comparison in average profits under three magnitudes of vehicles.

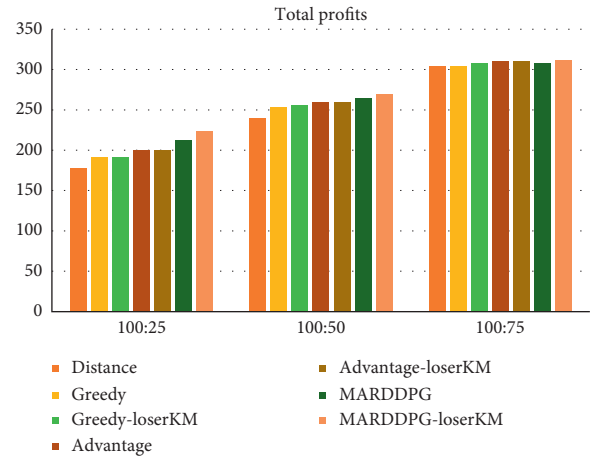

Figure 11: The comparison in total profits under three magnitudes of vehicles.

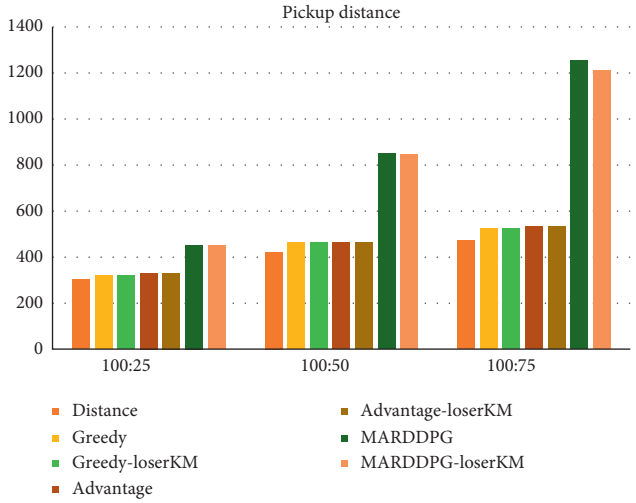

Figure 12: The comparison in pickup distance under three magnitudes of vehicles.

performance also rallies, because the distance between vehicles and RSUs is not too long at this situation; RSU can quickly finish computing services; then, go to next. The greedy algorithm has the best performance in the average profit due to being profit-driven, but it has decreased at a ratio of 100 : 50; we think this is caused by an extreme situation in which the driving distance is too long. The average profit of advantage algorithm will lag behind MARDDPG in some cases, but the experiment shows that the performances of advantage and MARDDPG are almost same in average profit; although we think MARDDPG takes regional preference into account in a period, vehicles will rush to the RSU with high information transmission. Our time step which is only 20 is relatively short. When RSU receives the service from one vehicle, it will finish it without interference; they cannot respond to a new service; therefore, this improvement is weakened.

Figure 11 shows the comparison of overall profits; it measures the degree of different matches. We can find that it does not have a regular relationship with the average profit. The overall performance of the distance algorithm is not outstanding because its weights are independent of the preference function. The greedy algorithm performs generally in terms of overall profits; although it prefers a higher average profit, it does not consider the performance of time spent, which makes it unable to achieve enough profits in a limited time slice, resulting in performance degradation. The advantage algorithm is more inclined to match RSUs with better overall profits; with the ratio rising, its overall profits are almost equal to our algorithm. MARDDPG will actively seek for RSUs because it takes into account the idle state; this will have a considerable improvement when the ratio is low. As the ratio increases, a large number of RSUs can enter the effective matching distance of vehicles, which reduces the probability of idle state.

We show the result of pickup distance in Figure 12; pickup distance represents the total distance that a vehicle moves; it can reflect the activity for exploring the area. The distance algorithm shines here because its weight is based on shorter distances. The performances of the greedy algorithm and advantage are also taken into account; as ratio increases, there will not be a large change in performance, because this will not produce extra mobile behavior. MARDDPG has the "worst" pickup distance because it will actively break the idle state and rush to the potential area. It can be truly proved with the ratio increasing. You can see that the distance has greatly increased in the real world, and it can be worked as a route recommendation because the vehicle can also rely on experience to rush to areas with more timely RSUs.

Until the end of one cycle, we evaluate the response rate from the RSUs (the number of the response/the total number of responses), as shown in Figure 13; it serves as a feeling of QoE and can also indirectly reflect the number of finished services. The distance algorithm has a good response rate, it prioritizes distance, and vehicles can quickly rush to the nearly RSUs; then, RSUs begin to compute the services from vehicles. The greedy algorithm is not good at responses, because the higher the profit is, the longer the calculation time is, and the more times RSU cannot respond,

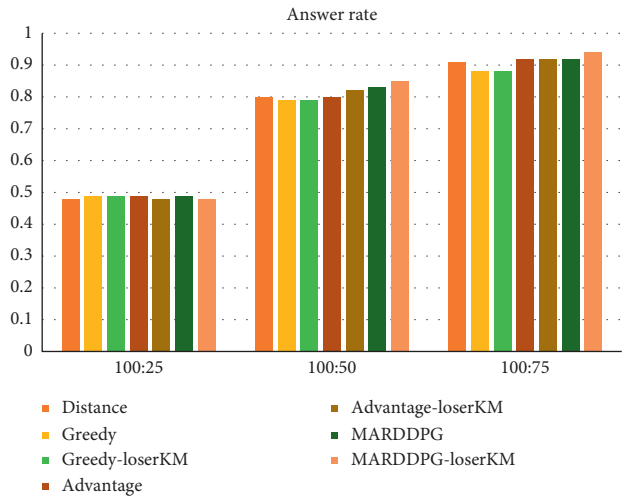

Figure 13: The comparison in answer rate under three magnitudes of vehicles.

leading to the lower response rate. Advantage algorithm is based on the weight of overall profits; it balances the tradeoff between time and distance; it also performs exceptionally in terms of response rate. MARDDPG reduces the time that vehicle stays in idle state and encourages them to actively seek more potential free RSUs, so that it can respond immediately. As ratio increases, this advantage is increasingly manifested; if one area is lacking free RSUs, it will guide vehicles to go to other areas.

5.3. Results Comparison and Analysis. The greedy algorithm is the most profitable because it is profit-driven; however, it also results in a lower response rate. The distance-driven algorithm has the smallest distance and its response rate is improved relative to the greedy algorithm because RSU can respond in time when the previous services are completed. The advantage algorithm has excellent performance, but there may be a situation where the agent may be vacant, so it is at a disadvantage in response rate and average profits.

MARDDPG gives the vehicles the ability to actively explore based on the advantages of retaining the preference matching algorithm, under the definition of the reward function; the vehicle is encouraged to go to the high preference area by central scheduling system, where it is more likely to encounter the free RSUs; its response rate and profits achieve better performance; correspondingly, this exploration also leads to more distances per vehicle, so it can be used as the path recommendation but not a signal source wizard when the MEC servers are sparse in the actual application for the reference to vehicle. As the ratio of RSUs to vehicles decrease, it seems that the performance gain is not large.

When the RSU density is high, the RSU coverage is close to the entire area, and the exploration ability will no longer affect. There are also some cases where the response rate and the total profits are not high, because the high-profit transmission needs to be transported for a long time, and the RSU cannot answer other vehicles during this transportation phase. When the running of the simulation model reaches the time T, the result statistics will be summarized. Upload services that are not completed because of a limited period will affect the relevant data.

These figures unify the performance of the various algorithms mentioned above under different metrics; it helps us to further analyze the experiment from a general point of view. We can find that the distance cost of MARDDPG is relatively large. The overall profit becomes flat with an increase of the ratio. We can find using the loser-KM and the preference function benefit total profits before the ratio of 100 : 75 because the system considers the future profit. The ratio of 100 : 50 is a kind of ideal environment; it has improved the total profits by 60% and approximately 100% response rate compared to the ratio of 100 : 25. We can also see that comparing the region preference with algorithms other than being profit-driven, it takes into account the factors of average return and response rate. The total profits from all servers and the total response rate which is indirectly related to the individual profits are used as the tradeoff metrics; they can embody the robustness and foresight of an algorithm when the RSUs exist quite far from the vehicles.

6. Conclusions and Future Work

In this paper, we introduce the multiagent reinforcement learning algorithm into the agent exploration and combine it with using the loser-KM algorithm in matching. According to the comparison of the traditional algorithms, we can find that it can perform better and satisfy the requirements of high response rate and total profits optimization. As shown in the figures, this algorithm is more suitable for the medium density, and it may lead a long way based on the prediction, so it had better be applied to the recommendation system with sparse RSUs; in large density areas, its merits are not obvious. The related algorithms of these comparisons are all achieved by ourselves, and the details mentioned in the referenced papers are restored as much as possible. In every experiment, each series of comparison algorithms use the same dataset generated by the RSUs generation models, but there are some extreme cases of experiment because a gap between theory and reality exists.

We will try to regard this reinforcement learning algorithm as a decision model in more fields and overcome the defect in the distance; it is related to the nature of the algorithm, whose prediction is a trial and error that leads to more distances. These may become the focus of our future work.

Data Availability

Performance evaluations illustrate the effectiveness of our constructed system on the simulated dataset. The data used to support the findings of this study are available within the article.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant no. 61772575, in part by the National Key R&D Program of China under Grant no. 2017YFB1402101, and in part by the Independent Research Projects of Minzu University of China.

References

[1] H. Gao, C. Liu, Y. Li, and X. Yang, "V2VR: reliable hybrid-network-oriented V2V data transmission and routing considering RSUs and connectivity probability," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–14, 2020.

[2] R. Deng, R. Lu, C. Lai, and T. H. Luan, "Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing," in *Proceedings of the 2015 IEEE International Conference on Communications (ICC'15)*, pp. 3909–3914, London, UK, June 2015.

[3] R. Deng, Z. Yang, J. Chen, N. R. Asr, and M.-Y. Chow, "Residential energy consumption scheduling: a coupled-constraint game approach," *IEEE Transactions on Smart Grid*, vol. 5, no. 3, pp. 1340–1350, 2014.

[4] P. Dong, X. Wang, J. J. P. C. Rodrigues, F. Xia, and Z. Ning, "Deep reinforcement learning for vehicular edge computing: an intelligent offloading system," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 6, 2019.

[5] L. T. Tan and R. Q. Hu, "Mobility-aware edge caching and computing in vehicle networks: a deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 10190–10203, 2018.

[6] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: a deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 44–55, 2018.

[7] X. Ma, H. Gao, H. Xu, and M. Bian, "An IoT-based task scheduling optimization scheme considering the deadline and cost-aware scientific workflow for cloud computing," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, p. 249, 2019.

[8] B. Chen and H. H. Cheng, "A review of the applications of agent technology in traffic and transportation systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 2, pp. 485–497, 2010.

[9] K.-T. Seow, N. H. Dang, and D.-H. Lee, "A collaborative multiagent taxi-dispatch system," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 3, pp. 607–616, 2010.

[10] L. Zhang, T. Hu, M. Yue et al., "A taxi order dispatch model based on combinatorial optimization," in *Proceedings of the 2017 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2151–2159, New York, NY, USA, August 2017.

[11] B. Li, D. Zhang, L. Sun et al., "Hunting or waiting? Discovering passenger-finding strategies from a large-scale real-world taxi dataset," in *Proceedings of the 2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pp. 63–68, IEEE, Seattle, WA, USA, March 2011.

[12] Y. Tong, Y. Chen, Z. Zhou et al., "The simpler the better: a unified approach to predicting original taxi demands based on large-scale online platforms," in *Proceedings of the 2017 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1653–1662, Halifax, Nova Scotia, Canada, August 2017.

[13] C. Yang, L. Bai, C. Zhang, Y. Quan, and J. Han, "Bridging collaborative filtering and semi-supervised learning: a neural approach for POI recommendation," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1245–1254, New York, NY, USA, August 2017.

[14] J. Liu, L. Sun, Li Qiao, J. Ming, Y. Liu, and H. Xiong, "Functional zone based hierarchical demand prediction for bike system expansion," in *Proceedings of the 2017 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 957–966, Halifax Nova Scotia Canada, August 2017.

[15] Z. Xu, Z. Li, Q. Guan et al., "Large-scale order dispatch in on-demand ride-hailing platforms: a learning and planning approach," in *Proceedings of the 2018 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, London, UK, July 2018.

[16] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 2014 31st International Conference on Machine Learning*, Beijing, China, June 2014.

[17] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[18] J. Feng, H. Li, M. Huang et al., "Learning to collaborate: multi-scenario ranking via multi-agent reinforcement learning," in *Proceedings of the 2018 International World Wide Web Conference*, Lyon, France, April 2018.

[19] N. C. Luong, Z. Xiong, P. Wang, and D. Niyato, "Optimal auction for edge computing resource management in mobile blockchain networks: a deep learning approach," in *Proceedings of the IEEE International Conference on Communications (ICC'18)*, Kansas City, MO, USA, May 2018.

[20] Y. He, F. Richard Yu, N. Zhao, C. Victor, M. Leung, and H. Yin, "Software-defined networks with mobile edge computing and caching for smart cities: a big data deep reinforcement learning approach," *IEEE Communications Magazine*, vol. 55, no. 12, pp. 31–37, 2017.

[21] Q. Qi, J. Wang, Z. Ma et al., "Knowledge-driven service offloading decision for vehicular edge computing: a deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 4192–4203, 2019.

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

[23] R. Zhang and M. Pavone, "Control of robotic mobility-on-demand systems: a queueing-theoretical perspective," *The International Journal of Robotics Research*, vol. 35, no. 1, pp. 186–203, 2016.

[24] M. Qu, H. Zhu, J. Liu, G. Liu, and H. Xiong, "A cost-effective recommender system for taxi drivers," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 45–54, New York, NY, USA, August 2014.

[25] S. Rahili, B. Riviere, S. Olivier, and S. Chung, "Optimal routing for autonomous taxis using distributed reinforcement

learning," in *Proceedings of the 2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 556–563, Singapore, November 2018.

[26] H. T. Wai, Z. Yang, Z. Wang et al., "Multi-agent reinforcement learning via double averaging primal-dual optimization," in *Proceedings of the Proceedings of the 32nd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, December 2018.

[27] Y. Han, X. Zhou, L. Yang, and S. Li, "A bipartite matching based user pairing scheme for hybrid vlc-rf noma systems," in *Proceedings of the 2018 International Conference on Computing, Networking and Communications (ICNC)*, pp. 480–485, Maui, HI, USA, March 2018.

[28] T. Yuan, X. Huang, M. Ma, and J. Yuan, "Balance-based SDN controller placement and assignment with minimum weight matching," in *Proceedings of the 2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, Kansas City, MO, USA, May 2018.

[29] H. Sun, Z. Chen, S. Yan, and L. Xu, "MVP matching: a maximum-value perfect matching for mining hard samples, with application to person Re-identification," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 6737–6747, Seoul, South Korea, October 2019.

[30] C. A. S. De Witt, J. N. Foerster, G. Farquhar et al., "Multi-agent common knowledge reinforcement learning," 2018, http://arxiv.org/abs/1810.11702.

[31] R. Raileanu, E. Denton, A. Szlam et al., "Modeling others using oneself in multi-agent reinforcement learning," in *Proceedings of the 2018 Machine Learning Research*, Stockholm, Sweden, July 2018.

[32] P. Mannion, K. Mason, S. Devlin, and J. Duggan, "Multi-objective dynamic dispatch optimisation using multi-agent reinforcement learning," in *Proceedings of the 15th International Conference On Autonomous Agents And Multiagent Systems*, Singapore, March 2016.

[33] H. Gao, W. Huang, and Y. Duan, "The cloud-edge-based dynamic reconfiguration to service workflow for mobile ecommerce environments," *ACM Transactions on Internet Technology*, vol. 21, no. 1, pp. 1–23, 2021.

[34] R. Raileanu, E. Denton, A. Szlam et al., "Modeling others using oneself in multi-agent reinforcement learning," in *Proceedings of the 35th International Conference on Machine Learning*, Stockholmsmässan, Stockholm, Sweden, July 2018.

[35] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, Sao Paulo, Brazil, May 2017.

[36] A. Nowé, P. Vrancx, and Y. M. D. Hauwere, "Game theory and multi-agent reinforcement learning," *Reinforcement Learning: State-of-the-Art*, Springer, New York, NY, USA, 2012.

[37] D. Qing and A.-X. Zeng, *Reinforcement Learning beyond Games: To Make a Difference in Alibaba*, Electronic Industry Press, Beijing, China, 2018.

[38] H. Gao, L. Kuang, Y. Yin, B. Guo, and K. Dou, "Mining consuming behaviors with temporal evolution for personalized recommendation in mobile marketing apps," *Mobile Networks and Applications*, vol. 25, no. 4, pp. 1233–1248, 2020.

[39] X. Yang, S. Zhou, and M. Cao, "An approach to alleviate the sparsity problem of hybrid collaborative filtering based recommendations: the product-attribute perspective from user reviews," *Mobile Networks and Applications*, vol. 25, no. 2, pp. 376–390, 2020.