Research on Edge Computing Offloading Based on Reinforcement Learning in Multi-user Scenarios

Zi-Hang Yu[1], Jian-Jun Zeng[2], Sai Liu[1], and Zhen-Jiang Zhang[1(✉)]

[1] Department of Electronic and Information Engineering, Key Laboratory of Communication and Information Systems, Beijing Municipal Commission of Education, Beijing Jiaotong University, Beijing 100044, China
{22110021,20120073,zhangzhenjiang}@bjtu.edu.cn

[2] College of Intelligence and Computing, Beijing InchTek Technology, Tianjin University, 100044 Beijing, China
jj@inchtek.ai

Abstract. Under the blessing of the new generation of information technology represented by 5G, a large number of new models and new businesses represented by smart logistics, industrial Internet, and smart transportation have emerged one after another, and the door to the intelligent interconnection of all things has officially opened. However, due to IoT sensor devices are usually responsible for data acquisition and transmission, they have certain limitations in terms of computing and storage capabilities. How to expand the performance of devices located at the edge of the network has become a focus of attention. Aiming at the problems of high energy consumption, prolonged time and high task failure rate in traditional IoT edge computing methods, this paper introduces deep reinforcement learning technology to optimize IoT edge computing offload methods. This paper models a single edge server multi-user scenario, and designs a function that comprehensively considers task delay and task failure rate as the goal of further optimization. At the same time, aiming at the problem of state space dimension explosion in traditional reinforcement learning, a computing task offloading method based on deep Q network is further proposed. Through simulation experiments, the results show that the proposed method has certain advantages in time delay and task success rate under the condition of different number of IoT devices.

Keywords: Edge computing · Reinforcement learning · Computing offload · Deep Q network

1 Introduction

With the rapid development of portable devices, the Internet of Things, and other fields, the need for more stringent requirements, computational sensitivity, and low latency has followed. With the advent of the 5G era, the network speed has once again increased rapidly, and many emerging fields and scenarios have emerged, such as Augmented

D.-J. Deng et al. (Eds.): SGIoT 2022, LNICST 497, pp. 102–112, 2023.
https://doi.org/10.1007/978-3-031-31275-5_11

Reality, Virtual Reality [1]. Not only to meet the requirements of low-latency and high-reliability communication, but also to ensure service quality. In addition, in daily life, study and work, there is a huge amount of data that needs to be processed, and these problems have an urgent need for powerful computing hardware.

In order to solve the above problems, the European Telecommunications Standardization Institute proposed mobile edge computing in 2014. Mobile edge computing is to solve the problems of computing delay of mobile cloud computing and insufficient computing equipment for big data processing. It is defined as providing cloud computing capability and IT service environment to application developers and content providers at the network edge [2]. Compared with mobile cloud computing, mobile edge computing can offload computing tasks to edge servers without going through a wide area network, thereby reducing latency and energy consumption and alleviating network bandwidth pressure [3].

Mobile edge computing also has its own shortcomings, limited by factors such as equipment and location, and the choice of task offloading strategy is particularly important in mobile edge computing [4]. When the user's own terminal cannot meet the data or business needs, need to choose a reasonable offload strategy, whether to offload this task, where to offload this task, these are all questions to consider. The goal of the offloading strategy is to optimize energy consumption, delay, cost, load balancing, etc. [5]. As for which weight is higher, it needs to be determined according to the specific application scenario. Moreover, in the actual scene, the environment is constantly changing, and the strategy cannot be static [9]. It needs to learn in the dynamic to achieve the best effect, so as to improve the service quality and user experience. Therefore, a reasonable offloading strategy can make MEC play the biggest role, utilize resources to the greatest extent [10], reduce service delay, reduce energy consumption of computing equipment, balance various loads in the system, and allow more users to obtain better user experience [11].

2 Related Work

One of the basic functions of the MEC system is to provide computing services to the edge of the network. In order to improve the quality of service (QoS), proper task offloading strategy is very important. In recent years, task offloading has received extensive attention from academia and industry. According to existing theoretical research, the task offloading problem in edge computing is a combinatorial optimization problem. From the current computing offloading task model, there are two main directions of existing research: Binary Offloading model and Partial Offloading model. In the binary offloading model, in order to simplify the problem, the tasks can no longer be divided, and the tasks can only be executed locally or on the edge server. The binary unloading model is still NP-Hard in most MEC scenarios. Dinh et al. aimed to find an offload schedule for a set of tasks between different access points and MEC service hosts to achieve the minimum goal of combining latency and energy consumption [6]. To obtain the best user experience quality as the goal, Hong et al. used approximate dynamic programming to solve it effectively [7]. Binary computing task offloading In addition to heuristic solutions, convex optimization methods are also used to solve binary offloading problems. Due to the constraints of the unloading problem, the unloading problem is usually non-convex.

Existing studies use approximation or relaxation methods to transform the non-convex problem into a convex optimization problem. Wang et al. adopted a relaxed approach to transform the joint allocation problem of cache and resources into a convex optimization problem [8]. However, with the emergence of time-sensitive tasks such as AR/VR and autonomous driving, traditional optimization algorithms cannot meet the needs well [12]. In response to the above problems, this paper integrates deep reinforcement learning into the edge computing of the Internet of Things, combines the perception ability of deep learning with the decision-making ability of reinforcement learning, and optimizes the computing offloading problem in the original edge computing of the Internet of Things.

3 System Model

3.1 Network Model

This article discusses the MEC system consisting of an Access Point (AP) and multiple mobile devices. The wireless AP can be a small cell base station or a Wi-Fi AP. In addition to the conventional AP, it has an MEC server. The MEC network model is shown in the Fig. 1.

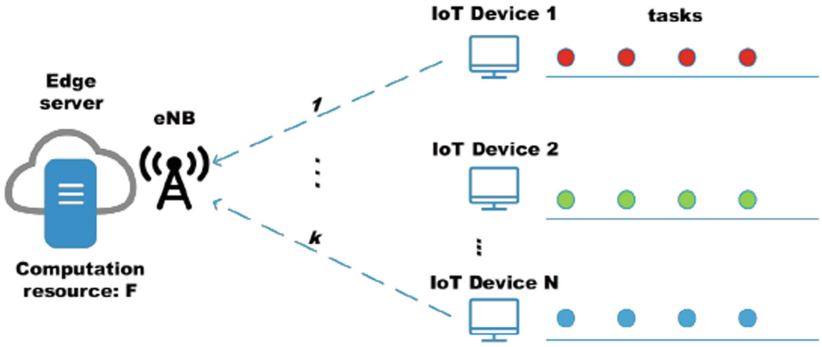

Fig. 1. MEC network model diagram

In the MEC network, there is one edge server and N IoT devices, and the edge server can provide computing services for the IoT devices within its coverage. Assuming that each IoT device has multiple computing tasks, these tasks can be selected to be executed locally or offloaded to the MEC server for execution. A collection of IoT devices can be represented as $\{1, 2, \ldots N\}$. The total computing resources of the MEC server are F. Divide time into multiple slots,the length of each slot is set to τ. Assume that the computing tasks of IoT devices follow a Poisson process. Its speed is λ.

Get a parameter as p, where $p = \lambda t_0$. This process is a Bernoulli process with parameter p. That is, the arrival time of each device computing task is an independent random variable.

The arriving task is defined by a triple $task_{i,j} = \left(b_{i,j}, c_{i,j}, \tau\right)$. Where $b_{i,j}$ represents the size of the input computational task. $c_{i,j}$ represents the number of CPU cycles

required to complete the computing task. τ represents the deadline for completing the calculation task, used for delay constraints. The variable $a_{i,j} \in \{0, 1\}$ is used to represent the offloading decision of the computing task. If $a_{i,j} = 0$, it means that the computing task is executed locally, if $a_{i,j} = 1$, it means that the computing task is offloaded to the edge server for execution.

3.2 Communication Model

Since this paper pays more attention to the computational offloading decision problem of the MEC server shared by multiple users, the wireless communication part is simplified, and the situation of users working on orthogonal channels is considered. Therefore, users do not suffer from multi-user interference with each other, which is a common situation in communication systems such as LTE (Long Term Evolution) at present. Since the wireless arrival state changes over time and is affected by the mobility of the terminal equipment, when a task arrives and the channel state is not good, it is chosen to be processed locally instead of waiting for better channel conditions. Suppose $h_{i,j}$ is the small-scale channel gain of the th terminal device to the MEC server in the j-th slot. The channel transmission rate calculated by Shannon's theorem is shown in the formula (1):

$$R_{i,j} = log\left(1 + \frac{d_{i,j}^{-\alpha}\left|h_{i,j}\right|^2 P_{i,j}}{\sigma^2}\right), i, j = 1, 2, \cdots, N \tag{1}$$

where d_i represents the distance from the device to the MEC server, α represents the path loss index, P_i represents the transmit power of the device, σ^2 represents the noise power of the MEC server.

3.3 Computational Model

According to the result of computing offloading, the computing tasks executed locally are waiting to be executed in the local buffer queue, and the computing tasks that are offloaded to the MEC server are waiting for computing in the buffer of the server.

Local Computing Model. If you choose to execute $\text{task}_{\text{i,j}}$ locally, the calculation formula of the processing delay $t_{i,j}^l$ of the task is as follows:

$$t_{i,j}^l = c_{i,j}/f_i^l \tag{2}$$

Among them, $c_{i,j}$ represents the amount of computation that the task needs to complete, and f_i^l represents the computing capability of the local device. At the same time, taking into account the time limit of task processing, propose a penalty $\beta_{i,j}$ for not completing the task within the deadline, the calculation formula is as (3):

$$\beta_{i,j} = 1_{\{t_{i,j}^l > \tau\}} \tag{3}$$

where $1_{\{\mu\}}$ is an indicator function whose value is 1 only when the condition is true, and 0 otherwise.

Edge Computing Model. The process of offloading tasks to the edge server mainly includes three processes. First, the terminal device uploads the input data to the edge server through the wireless network, then the edge server executes the calculation task, and finally returns the calculation result to the corresponding terminal device. The delay calculation formula of the unloading process is as follows:

$$t_{i,j}^{off} = t_{i,j}^{trans} + t_{i,j}^{comp} \tag{4}$$

$$t_{i,j}^{trans} = b_{i,j}/R_{i,j} \tag{5}$$

$$t_{i,j}^{comp} = c_{i,j}/f_{i,j} \tag{6}$$

which $t_{i,j}^{trans}$ represents the transmission time that will calculate the data sent from the terminal device with the MEC server. $t_{i,j}^{comp}$ represents the processing time of computing tasks performed by the MEC server, $b_{i,j}$ represents the calculated data size transferred. $R_{i,j}$ represents the transmission speed of the i-th terminal device between the j-th slot and the server. $f_{i,j}$ represents the computing power allocated by the MEC server to the terminal device. Since the computing resources of the server are limited, the computing resources allocated by the MEC server to each terminal device should be smaller than the computing resources possessed by the MEC server, that is, the following constraints are satisfied:

$$\sum\nolimits_{i=1}^{N} f_{i,j} \leq F, \forall j \tag{7}$$

Also taking into account the time limit of task processing, it is proposed to represent the penalty for not completing the task within the deadline. The calculation formula is as follows:

$$\beta_{i,j} = 1_{\{t_{i,j}^{off} > \tau\}} \tag{8}$$

3.4 Problem Modeling

According to formula (2) (4) (5) (6), the computational task delay can be expressed as

$$t_{i,j} = a_{i,j} t_{i,j}^{off} + \left(1 - a_{i,j}\right) t_{i,j}^{l} \tag{9}$$

where $a_{i,j}$ represents the unloading decision vector of the terminal device. Considering the task processing delay and the task timeout failure rate, the objective function is obtained, as follows:

$$\begin{aligned} U = & \frac{1}{T}\frac{1}{N}\sum\nolimits_{j=1}^{T}\sum\nolimits_{i=1}^{N}\left(\omega_1 t_{i,j}/\tau + \omega_2 \beta_{i,j}\right) \\ s.t. \;\; & C1: a_{i,j} \in \{0, 1\}, \; \forall i, j \\ & C2: \; \sum_{i=1}^{N} f_{i,j} \leq F, \forall j \end{aligned} \tag{10}$$

Which $C1$ means that the type of computational unloading considered in this paper is binary unloading, $C2$ indicates computing resource constraints. The objective function consists of two parts, which $\frac{1}{T}\frac{1}{N}\sum_{j=1}^{T}\sum_{i=1}^{N}\left(\omega_1 t_{i,j}/\tau\right)$ represents the computing task delay, $\frac{1}{T}\frac{1}{N}\sum_{j=1}^{T}\sum_{i=1}^{N}\omega_2\beta_{i,j}$ represents the timeout task penalty item. ω_1, ω_2 represents the weighted value of these two terms.

4 Computational Offloading Algorithm Based on Reinforcement Learning

This section defines the three elements of reinforcement learning, state, action, and reward in detail. And introduced the reinforcement learning algorithm Q-Learning and DQN algorithm, combined with the computational offloading scenario designed above, the multi-objective optimization problem with computational offloading constraints is transformed into a problem of maximizing the Q value. Among them, Q-Learning stores the Q value in the Q table. As the number of user devices increases, the problem of excessive dimension may occur. The DQN algorithm is further proposed, which replaces the Q table with a deep neural network and uses it to approximate the Q function.

4.1 Q-Learning Algorithm

Reinforcement learning algorithms differ from MDPs in that RL algorithms attempt to derive optimal policies without an explicit model of the environment's dynamics. Reinforcement learning agents learn in real-world interactions with the environment and refine their behavioral policies through the experience gained from interacting with the environment. In the scenario of edge computing offloading, the observation information of the agent should include the transmission rate between the terminal device and the edge server $R_{i,j}$, the amount of data sent by the terminal device to the server $b_{i,j}$, and computing resources required for computing tasks $c_{i,j}$, That is, the state vector is $s(j) = [R_1, R_2, ...R_N, b_1, b_2...b_N, c_1, c_2, ...c_N]$.

The action vector of the agent includes the unloading decision of each terminal device and the computing resources allocated to the terminal device by the MEC server. That is, the action vector is $a(j) = \left[a_1, a_2, ...a_N, f_1, f_2...f_N\right]$. The reward function of the agent is set according to the optimization goal in the edge computing offloading scenario. Since the optimization goal is to obtain the minimum value of the function, and reinforcement learning is to obtain the maximum reward, the reward function needs to take a negative value on the joint optimization goal. That is, the reward $r(i, j)$ is $-\sum_{j=1}^{T}\sum_{i=1}^{N}\left(\omega_1 t_{i,j}/\tau + \omega_2\beta_{i,j}\right)$. Q-Learning is a classic off-policy algorithm based on time difference. It directly approximates the Q-value, and the agent state transitions are independent of the policy it is learning. The RL agent will learn from actual interactions with the environment and adjust its behavior after experiencing the consequences of its behavior to maximize the expected discounted return. The formula for estimating the value of Q is as follow:

$$Q(\mathrm{s}, \mathrm{a}) \leftarrow \mathrm{Q}(\mathrm{s}, \mathrm{a}) + \alpha\left(\mathrm{r}(\mathrm{s}, \mathrm{a}) + \gamma \max \mathrm{Q}\left(\mathrm{s}', \mathrm{a}'\right) - \mathrm{Q}(\mathrm{s}, \mathrm{a})\right) \tag{11}$$

where α represents the learning rate, in the iterative process, it is responsible for controlling the learning progress of the agent. $r(s, a) + \gamma \max Q(s', a') - Q(s, a)$ represent TD-error. The current Q value can be updated by this increment. Indicates the value obtained by taking action in the current state, s′ represents the state reached after taking an action a in the state s.a′ represents the action of obtaining the maximum Q value in state s. The parameter γ represents the discount factor, which represents the importance of future rewards. When γ approaches 0, the agent will only consider the rewards obtained in the current state, and when it approaches 1, the agent will also pay attention to the rewards obtained in the future.

The process of the Q-Learning algorithm is as follows: first, initialize the Q table randomly, and in each training episode, read the state information between the terminal device and the MEC server, select the largest Q value according to the greedy strategy, and get its corresponding action, then send the computing offload and resource allocation information to the terminal device, then enter the next state. Update the Q table according to the formula (12), and iterate continuously.

In the Q-Learning algorithm, the Q table is used to access the Q value corresponding to each state and behavior. However, due to the large number of terminal devices connected to an MEC server, it is difficult for computer memory to store all of them. Therefore, the neural network in machine learning is introduced, and the set of states and actions is used as the input of the neural network, after the convolution and activation functions in the neural network, Output state - Q value of action set. That is, the network f with parameter θ is used to estimate the Q value. The formula is as follows:

4.2 DQN Algorithm

Deep Q network (DQN) adopts the convolutional neural network in deep learning as the generalization model of the state-action value function; at the same time, it uses the classical algorithm of reinforcement learning-Q-Learning to update the model parameters of the state-action value function, so that the model eventually a better strategy can be learned.

$$f(s, a, \theta) = Q^*(s, a) \tag{12}$$

Deep Q networks have two very important mechanisms. First, DQN has a memory pool for storing previous experiences. Each time DQN is updated, some previous experience can be randomly selected for learning. Second, there are two neural networks with the same structure but different parameters. The neural network for predicting the Q value has the latest parameters, while the neural network for the target Q value uses parameters from a period of time ago and remains unchanged for a period of time. So that the difference between the two network outputs can be used for continuous training, and the stability during the training process can be enhanced at the same time.

For the DQN algorithm, its execution process is as follows: First initialize the experience pool space to N, use random weights θ,θ^-. Initialize the evaluation network and the target network. At this time, the parameters of the target network and the evaluation network are the same. Perform operations for each episode: obtain status information between the terminal device and the MEC server, get the initial state. For each time slot,

the action a_t is selected according to the ε-greedy algorithm. After that, the terminal device calculates reward r_t according to the unloading decision and resource allocation. And get the observation state s_{t+1} in the next slot.Store the results (s_t, a_t, r_t, s_{t+1}) obtained in the above process into the experience pool. Randomly select min-batch data from the experience pool to update the evaluation network parameters θ, the target network parameters θ^- are updated at intervals of C steps. The DQN algorithm is deployed in the MEC server, and a centralized method is used to obtain the state between the terminal device and MEC server. At the same time, due to the obvious correlation between each continuous state in reinforcement learning, it cannot meet the requirement of independent and identical distribution of data in neural network training, so the experience replay mechanism is introduced. The experience is stored in the experience pool, and during training, data is randomly selected for neural network training. The training process is shown in Fig. 2.

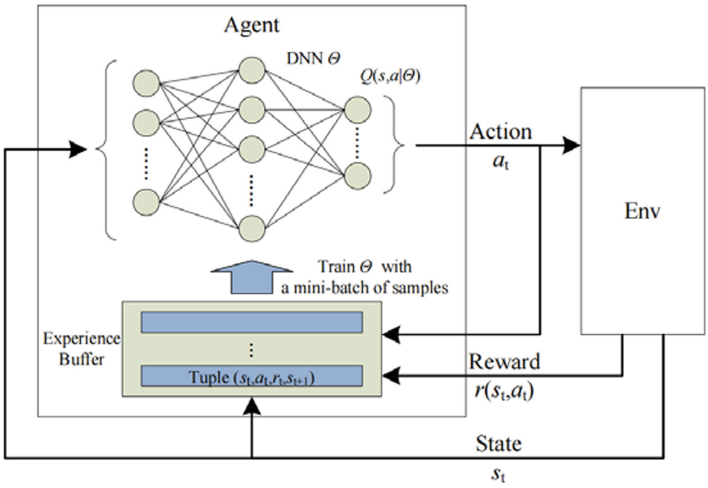

Fig. 2. DQN training process diagram

Select $l(\theta)$ as the loss function for training, in which the network parameters of the target network are updated after C steps to ensure the stability of the training process. The formula for calculating $l(\theta)$ is as follows:

$$l(\theta) = \left[r_t + \gamma \max Q^-\left(s_{t+1}, a'; \theta^-\right) - Q\left(s_{t+1}, a'; \theta\right)\right]^2 \tag{13}$$

5 Experimental Simulation

5.1 Simulation Parameter Settings

The simulation parameters are set as shown in Table 1.

In order to describe the performance of the algorithm proposed in this paper, three algorithms are selected as benchmark algorithms for comparison:

Table 1. Simulation parameter table.

parameter	value
Number of terminal devices N	4
MEC server performance F	40 GHZ
Terminal equipment performance f	3 GHz
task data size l	[20,40] kb
The size of the task calculation c	[0.5,3] GHz
length of a slot τ	1 s
target parameter ω_1	5
target parameter ω_2	5

(1) **All offload to the server for execution:** All tasks on the terminal device are offloaded to the MEC server for calculation, and at the same time, the computing resources are selected and distributed evenly.
(2) **All local execution:** The task on the terminal device is selected to be executed locally without offloading.
(3) **Greedy algorithm:** When the channel condition is optimal, task is selected to be offloaded to the MEC server for execution, and under other conditions, the computing task is executed on the terminal device. And the computing tasks offloaded to the MEC server evenly distribute the computing resources of the MEC server.

5.2 Objective Function Weight Setting

To determine the weight of the objective function ω_1, ω_2, it is necessary to study the influence of the weight value on the delay and task failure rate under the use of the DQN algorithm. As for ω_1, ω_2, the effect of each weight on latency and task failure rate was studied by fixing one of the values to 1.

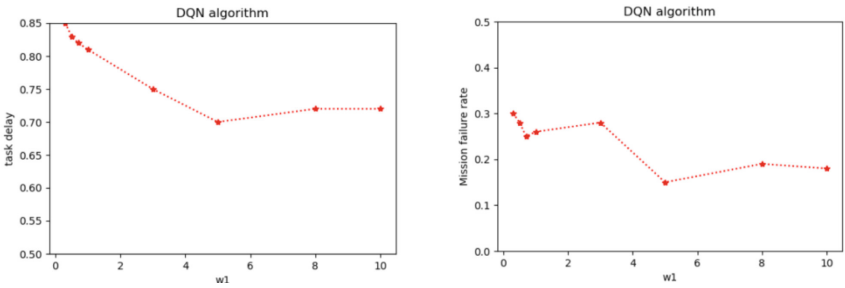

Fig. 3. The effect of parameter ω_1 on task delay and Mission failure rate

The above Fig. 3 represents the impact of different values on the delay and mission failure rate. It can be seen from the figure that with the increase of ω_1, the overall delay

first decreases and then increases, and when equal to 5 time to get the minimum value. With the increase of ω_1, the task failure rate decreases first and then increases as a whole, and it is also obtained near 5. Therefore, the selected value is 5. Using the same experimental method, it is determined that the value of ω_2 is also 5.

5.3 The Effect of the Number of End Devices

This part studies the effect of changes in the number of IoT devices in the entire MEC system on the overall objective function when other conditions remain unchanged. The values of ω_1 and ω_2 are both 5.

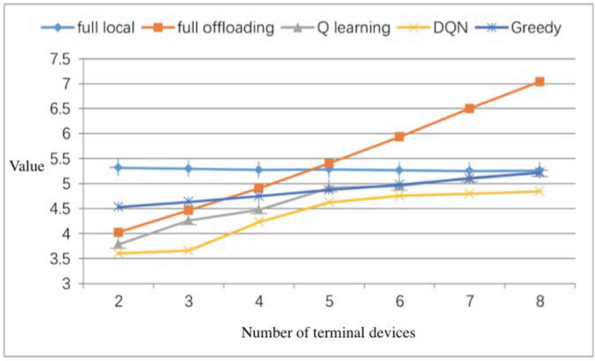

Fig. 4. The influence of the number of terminal devices on the objective function value

Figure 4 represents the effect of the number of end devices on the overall objective function. As can be seen from the figure, as the number of terminal devices increases, it can be seen from the figure that for policies that are all executed locally, the increase in the number of IoT devices will not affect their objective function. With the increase of the number of IoT devices, the total utility function of the strategy, greedy algorithm, Q learning and DQN strategy that are all offloaded to the edge server for execution gradually increases, and the growth rate of the all offloading strategy is the largest. When the number of IoT devices is less than or equal to 4, the full offloading strategy is better than the all locally executed strategy. When the number of terminal devices is greater than 5, the local offloading strategy is better than the all offloading strategy. At the same time, the advantages of the DQN algorithm are more obvious.

6 Future Work

This paper mainly models for multi-user edge computing scenarios, and designs a function that comprehensively considers task delay and task failure rate as the goal of further optimization. Combining the Q-Learning algorithm and the DQN algorithm, and conducting simulation experiments with different numbers of terminal devices, the simulation results show that it has certain advantages in time delay and task success rate. Some

algorithms proposed in this paper based on deep reinforcement learning are applied to the problem of computational offloading, but only the task delay and task failure rate are considered. In real edge computing scenarios, the power of terminal devices is usually limited. How to reduce energy consumption as much as possible in the process of computing offloading while ensuring that the delay is met is a problem that needs to be considered in the future. In real scenarios, the number of terminal devices changes dynamically, and how the algorithm performs adaptive adjustment is also a problem worthy of consideration.

Acknowledgment. This work was supported by the National Natural Science Foundation of China (No. 62173026).

References

1. Saeik, F., et al.: Task offloading in edge and cloud computing: a survey on mathematical, artificial intelligence and control theory solutions. Comput. Netw. **195**, 108177 (2021)
2. Mach, P., Becvar, Z.: Mobile edge computing: a survey on architecture and computation offloading. IEEE Commun. Surv. Tutor. **19**(3), 1628–1656 (2017)
3. Zhang, Z., Li, S.: A survey of computational offloading in mobile cloud computing. In: 2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), pp. 81–82. IEEE (2016). https://doi.org/10.1109/MobileCloud.2016.15
4. Zhang, G., Zhang, S., Zhang, W., Shen, Z., Wang, L.: Joint service caching, computation offloading and resource allocation in mobile edge computing systems. IEEE Trans. Wirel. Commun. **20**(8), 5288–5300 (2021)
5. Zhan, Y., Guo, S., Li, P., Zhang, J.: A deep reinforcement learning based offloading game in edge computing. IEEE Trans. Comput. **69**(6), 883–893 (2020)
6. Dinh, T.Q., Tang, J., La, Q.D., Quek, T.Q.: Offloading in mobile edge computing: task allocation and computational frequency scaling. IEEE Trans. Commun. **65**(8), 3571–3584 (2017)
7. Hong, S.T., Kim, H.: QoE-aware computation offloading to capture energy-latency-pricing tradeoff in mobile clouds. IEEE Trans. Mob. Comput. **18**(9), 2174–2189 (2018)
8. Wang, C., Liang, C., Yu, F.R., Chen, Q., Tang, L.: Computation offloading and resource allocation in wireless cellular networks with mobile edge computing. IEEE Trans. Wirel. Commun. **16**(8), 4924–4938 (2017)
9. Lu, J., et al.: A multi-task oriented framework for mobile computation offloading. IEEE Trans. Cloud Comput. **10**(1), 187–201 (2019). https://doi.org/10.1109/TCC.2019.2952346
10. Huang, C.M., Wu, Z.Y., Lin, S.Y.: The mobile edge computing (MEC)-based VANET data offloading using the staying-time-oriented k-hop away offloading agent. In: 2019 International Conference on Information Networking (ICOIN), pp. 357–362. IEEE (2019). https://doi.org/10.1109/ICOIN.2019.8718188
11. Aiwen, Z., Leyuan, L.: Energy-optimal task offloading algorithm of resources cooperation in mobile edge computing. In: 2021 4th International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE), pp. 707–710. IEEE (2021). https://doi.org/10.1109/AEMCSE51986.2021.00146
12. Ko, H., Pack, S., Leung, V.C.: Performance optimization of serverless computing for latency-guaranteed and energy-efficient task offloading in energy harvesting industrial IoT. IEEE Internet Things J. (2021). https://doi.org/10.1109/JIOT.2021.3137291