Applied Soft Computing 116 (2022) 108361

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Inverse order based optimization method for task offloading and resource allocation in mobile edge computing

Junyao Yang, Yan Wang *, Zijian Li

School of Automation Science and Electrical Engineering, Beihang University, Beijing, China

ARTICLE INFO

Article history:
Received 29 March 2021
Received in revised form 16 October 2021
Accepted 19 December 2021
Available online 25 December 2021

Keywords:
Mobile edge computing
DDPG
GA
Partial offloading
Computation offloading

ABSTRACT

Edge computing, which provides lightweight cloud computing and storage capabilities at the edge of the network, has become a new computing paradigm. A key research challenge for edge computing is to design an efficient offloading strategy for offloading decision-making and resource allocation. Although many researches attempt to address this challenge, the traditional offloading strategies cannot adapt to complex environments, and the offloading strategies based on reinforcement learning require centralized control or the pursuit of the user's best interests, which is impractical. Individual users should rationally pursue benefits in order to create a high-quality offloading environment to obtain long-term benefits. In this paper, we first separate the offloading process into a two-step offloading framework, and reverse the order of solving offloading decision and resource allocation problems to reduce the dimensionality of the action and state space. We formulate the resource allocation as a Markov Decision Process (MDP) and use the Deep Deterministic Policy Gradient Algorithm (DDPG) to adjust load balancing of the edge server and reduce the transmission energy and delay, and then use the genetic algorithm (GA) to search for decisions and use Fully-Connected Network (FCN) to fit the decision-making process, thereby avoiding excessive response time caused by iteration. Simulation results show that compared with baseline methods, the proposed algorithm is more stable, flexible, adaptable and suitable for practical applications.

1. Introduction

Modern smart terminal devices are often restricted in computing power due to typically limitations such as small size and production costs [1]. Therefore, when user devices are faced with compute-intensive and time-sensitive tasks [2,3], the computing power cannot guarantee stable and high quality of experience (QoE). To alleviate the shortcomings of traditional cloud computing [4] technologies in terms of real-time capacity, edge computing that uses servers deployed in network edge nodes to process and analyze data has recently received more and more attention [5]. However, compared with traditional cloud servers, MEC servers tend to have fewer resources and are more variable, which leads to devices that need to compete for the limited computing resources of the server [6,7]. Therefore, resource allocation and scheduling, such as server selection, allocation of offloading ratios, and computational capacity, are quite important for such resource-constrained systems. To efficiently utilize the computing resources and satisfy the computation requirements of users, an intelligent computation offloading strategy is indispensable. Hence, computation offloading has attracted more and more attention of researchers.

Edge computing offloading methods can significantly enhance users' computing capabilities, but due to the limited hardware resources of edge servers, it is still difficult to develop a portable and scalable computing offloading method. When many users try to offload tasks at the same time, too many resources are occupied by the edge server and wireless channel, which will cause a long task response time. Therefore, it is important to design an effective offloading strategy to determine which tasks to offload, which part of tasks to offload to the edge server, and how the edge server allocates computing resources. This problem is considered to be one of the most critical challenges of edge computing, and most of the existing work has unrealistic assumptions. Therefore, although others' proposed method achieves good performance in their simulation experiments, it cannot achieve satisfactory results in the actual environment. Methods based on game theory, such as [8–11], require centralized control of user offloading tasks to achieve global optimal performance. However, it is impractical to force all users to act in accordance with centralized control, because individual users seek to maximize their

* Corresponding author.
E-mail address: e1105_2021@163.com (Y. Wang).

https://doi.org/10.1016/j.asoc.2021.108361
1568-4946/

own interests. The optimization method based on Lyapunov [12–14] requires priori information of the environment and cannot adapt to changes in the environment.

Reinforcement learning is a powerful framework that excels in solving these dynamic, scenario-driven or self-aware decision-making problems. Using reinforcement learning to solve the problem of computational offloading under edge conditions, a simple combination of reinforcement learning and computational offloading has been proposed in the work [15]. With the deepening of research, DRL is becoming an effective method to obtain the best decision-making strategy and maximize long-term returns [16]. These explorations have greatly inspired the research of edge computing, but because Q-learning is only applicable to fields where its actions and states are discrete, inputs and outputs must be low-dimensional that limits the diversity of tasks. Obviously, this is contradictory to actual situations. Although the deep Q network (DQN) can handle high-dimensional sequence data, it is also limited by its simple discrete output and slow convergence speed.

In this article, we propose a MEC task offloading and resource allocation algorithm based on the inverse order. We separate the offloading process into a two-step offloading framework, and reversed the order of solving offloading decision-making and resource allocation problems to reduce the dimensionality of the action and state space. We formulate the resource allocation as a MDP and use DDPG [17]. By comprehensively considering task frequency, load balancing, computational delay and energy consumption, we can achieve optimal resource allocation that not only guarantees user QoE but also minimizes the impact on subsequent user offloads. After resource allocation is set for task offloading on all edge servers, the action space of all offloading decisions is fixed. The optimal offloading decision search is through GA with an appropriate fitness function, and FCN is used to fit the decision process.

The main contributions of this article are summarized as follows.

(1) We separate the continuous–discrete mixed action space in the offloading problem to form a two-step offload framework, and exchange the calculation order to reduce the dimensionality of the action and state space.

(2) We add load balancing to the reward function of the resource distribution network, so that each individual user tends to create a high-quality offloading environment to obtain long-term benefits, and realize multi-user joint optimization without centralized control.

(3) Simulation results show that compared with the baseline method, the proposed algorithm is more stable, flexible, adaptable and suitable for practical applications.

The rest of this article is organized as follows. In Section 2, we discussed some related work. Section 3 introduces the system model and algorithm. Section 4 evaluates system performance through simulation, and Section 5 summarizes this paper.

2. Related work

The edge computing paradigm has attracted considerable attention in both academia and industry over the past several years. Resource management plays an important role in both cloud system and edge system. The effective distribution and deployment of storage, network bandwidth, computing resource can bring significant improvement for energy saving and latency reduction. Offloading aims at improving QoE by properly distributing computational capacity. Real-world scene is complex and dynamic which infers that a good offloading algorithm and computational architecture should have enough scalability. But it is tough for most models to handle problems derived from different scenes.

Researchers have proposed many methods for computation offloading policies. Many of the previous study have adopted classical optimization or game theory-based methods to solve the problem [8–14]. [8–10] use methods based on game theory to describe the multiuser computation offloading problem as a non-cooperative game to optimize user QoE and MEC provider revenue. [11] formulate the problem as a partially observable Markov decision process (POMDP) and formulate it as a multi-agent POMDP., which is solved by a policy gradient DRL based approach. [12–14] propose methods based on Lyapunov optimization theory to optimize user QoE for single-user multitask offloading, multiuser–multitask offloading, and offloading privacy issues.

These studies model users as self-interested game players and propose a decentralized solution to the problem of multi-user computational offloading. However, they are mainly concerned with the problem of computing offloading in their abstracted and simplified simulation environment. In a real-world environment, due to the time change of the wireless network, the utility of each user changes dynamically, so the Nash equilibrium in the static game model may not be solvable. Besides, although classic optimization methods such as Lyapunov are widely used when solving long-term optimization problems, only an approximate optimal solution can be obtained.

To solve these problems, researchers model the computation offloading problem as a Markov decision process (MDP) and solve it with reinforcement learning (RL) [15] or deep RL(DRL) methods [16]. DRL has recently made great progress and various algorithms [18–21] have been proposed.

DRL is emerging as an effective approach to obtain the optimal decision-making policy and maximize the long-term rewards [22]. [23] propose a DRL algorithm DRGO combined with the adaptive genetic algorithm that achieves system utility optimization and blockchain subtask optimization. [24] formulate the offloading as a MDP and utilize the asynchronous advantage actor-critic algorithm as the offloading decision-making strategy to balance the workload of edge servers and finally increase QoE. However, the performance of the algorithm is greatly affected by the convergence and accuracy of the deep Q-network (DQN). Therefore, improving the performance of DQN for computation offloading has attracted much attention recently. [25] describe the safety-aware task offloading problem as a Markov process with a risk rate constraint, and proposed a DQN-based safety and cost-aware computing offloading algorithm, which improved safety and cost efficiency. [26] propose to improve the DRL algorithm by combining packet loss regularization and double-depth Q-network. The improved algorithm can better approximate the performance of the exhaustive search scheme. [27] develop a double deep Q-network and a linear Q-function decomposition-based SARSA method to deal with the state space explosion. However, the dimension of action space and state space will expand exponentially when considering multiuser–multiserver MEC systems.

The DDPG-based method has certain advantages when dealing with the problem of computing offloading. [28] propose a D^3PG algorithm based on the DDPG to address the trade-off between the choice of target servers and algorithm convergence. [29] redesign the critic network of DDPG and proposed the multi-critic DDPG method to make a balance of the DDPG stability and offloading performance. [30] propose an improved DRL algorithm, called PS-DDPG, with DDPG-based Priority Experience Replay (PER) and Random Weight Averaging (SWA) mechanisms to enhance the availability of the experience replay buffer, thus improving stability and convergence. However, the slow convergence caused by high-dimensional space still remains an urgent and challenging problem.

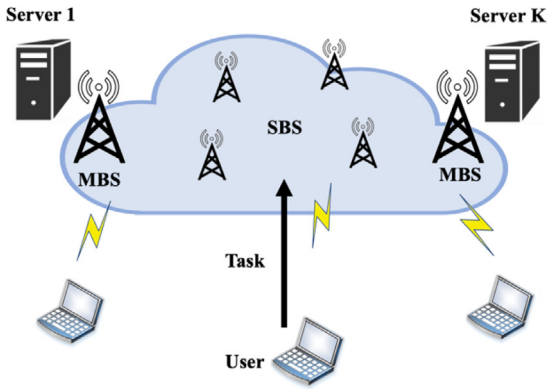

Fig. 1. Computation offloading system model for MEC.

The above RL or DRL-based methods have achieved satisfactory performance without requiring the prior knowledge of environment statistics. However, they are modeled in either a discrete action space or a continuous action space, which restricts the optimization of offloading decisions given the constraints. The action space of offloading problem in practical situations is often continuous–discrete hybrid, where each device needs to jointly decide continuous and discrete actions to accomplish the offloading process. For example, the device should not only decide whether to offload tasks or which server shall them be offloaded to but also select the offloading ratio, or the local computation capacity to balance the time and energy consumption. Therefore, these methods may not perform well as finite discretization of continuous action will be problematic when the action space becomes large, and relaxing discrete action into a continuous set might significantly increase the complexity of the action space. In addition, problems such as commonly local optimization and the incompleteness of the system model are also urgently needed to be solved.

3. Approach

3.1. System model

As shown in Fig. 1, we consider a single-user multi-edge server multi-base station system to model MEC computation offloading. Among them, the computation capacity of user devices is much lower than that of mobile edge computing servers (MECS), and MECS have multi-task parallel computing capacity, while user devices can only handle tasks separately. The communication network adopts frequency division multiple access, and the base station is composed of M macro base stations (MBS) and S small base stations (SBS), and MECS are deployed next to MBS. The base stations deployed in different locations and K nearest edge servers around the user form the communication network. On the communication network between the user and the MECS, each base station has Y channels, and the gain of each channel is different. The transmission speed between the base station and the MECS is inversely proportional to the distance. We add task divisibility, task frequency and communication network congestion parameters to the system model. Next, we will introduce them respectively. For the convenience of reading, we summarize the notations used in this paper in Table 1.

Task divisibility: For each task, it can be completely described by the computation size (the number of CPU cycles required for calculation) and the data size. In this paper, each task is given different divisibility, which can be divided into the following three types:

Table 1
Summary of notations.

Notation	Meaning
M	Number of macro base stations
S	Number of small base stations
K	Number of mobile edge computing servers
Y	Number of channels per base station
N	Number of tasks processed by the user per day
$P(T)$	Task frequency
C_{num}	Number of unavailable channels
W	The maximum congestion parameter of the communication network
d_n	Task data size ($n = 1, 2 \ldots N$)
c_n	Task calculation size
T_n	Task arrival time
$type_n$	Task type
t_n^l	Local delay
f_n^l	Local CPU cycles per second
z_n^l	User device energy efficiency
t_n^{up1}	Transmission delay of uploading data from *UE* to *BS*
r_n	*UE* data upload rate
t_n^{down1}	Transmission delay of downloading data from *BS*
φ_n	Transmission rate
t_n^C	Calculation delay
f_n	Calculation resource allocated by *MECS*
t_n^{up2}	Transmission delay of uploading data from the previous *MECS* to *BS*
t_n^{down2}	Transmission delay of uploading data from *BS* to the target *MECS*
$t_n^{max(2,3)}$	Total delay of subtask 2 and 3
e_n^{up1}	Energy consumption of uploading data form *UE* to *BS*
P_n	Transmission power
e_n^{down1}	Energy consumption of downloading data from *MECS* to *BS*
e_n^C	Calculation energy consumption
z_n^{MECS}	Energy efficiency allocated for task by *MECS*
B_n	Load balancing before offloading
b_n	Load balancing change before and after offloading
F	Computing resources being used by the edge server before offloading
F'	Computing resources being used by the edge server after offloading
F_n^{left}	Remaining computing resources of each edge server
C^{occupy}	Channel occupancy of each base station
F_n^d	Computing resources allocated to each subtask
BS_n^d	Base station and channel used by each subtask

Type-1: Indivisible tasks, which can only be calculated locally or all offloaded.

Type-2: It can be divided into two subtasks and must be calculated sequentially.

Type-3: It can be divided into four subtasks, among which subtask 1 must be completed locally, subtask 2 and subtask 3 can be calculated at the same time (only for offloading calculation), and the input of subtask 4 is the output of subtask 2 and 3.

In Types 2 and 3, the calculation size and data size of each subtask are different, and the proportion in the total task is random.

Task frequency: It is assumed that the user can only generate one task at a time, and the local CPU and uplink data communication can only manage one task. The tasks that user need to process every day are constant N, and the distribution conforms to the mean of two normal distributions, where A a.m. and B p.m. are peaks. As shown in Fig. 2, we use $P(T)$ to denote the task frequency:

$$P(T) = \frac{1}{2}(P(T \sim N(A, 4)) + P(T \sim N(B, 4))) \quad (1)$$

Communication network congestion: The degree of communication network congestion is expressed by adjusting the

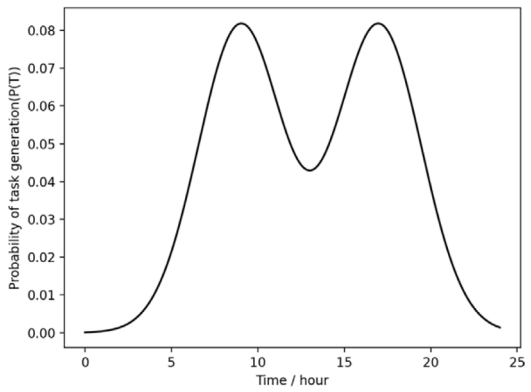

Fig. 2. Task frequency.

number of unavailable channels C_{num}:

$$C_{num} = W\left(\frac{P(T)}{P(T)\,max}\right) * (M+S)\,Y \tag{2}$$

Where $W \in [0, 1]$ is the maximum congestion parameter of the communication network, when W is 1, there is no frequency channel available in communication network at A or B moment.

In this section, we introduce the improved system model and add parameters to create a more realistic offloading environment, which makes us more convincing in evaluating the usability and effectiveness of the proposed method. Then we will introduce the model of computation offloading in detail.

3.2. Computation model

In addition to delay and energy consumption mainly included in QoE, we consider load balancing as another index of performance evaluation. Load balancing is an important method to effectively realize the use of multiple resources in a cluster. It is mainly used to optimize resource usage, maximize throughput, minimize response time, avoid overload, enhance network data processing capabilities, and improve network flexibility and availability.

Next, we will introduce the calculation methods of these three evaluation criteria for the two modes of local and offloading computing respectively.

3.2.1. Local computing

Tasks are described by data size, calculation size, arrival time and task type. The user executes the calculation $Task_n = (d_n, c_n, T_n, type_n)$ locally. The local delay t_n^l only includes local calculation delay and is given by:

$$t_n^l = c_n / f_n^l \tag{3}$$

Where f_n^l represents the local CPU cycles per second number.

Meanwhile, we use z_n^l to denote the energy efficiency of the user, the energy consumption for local computing is given by:

$$e_n^l = c_n z_n^l \tag{4}$$

Since the local CPU cannot implement the parallel calculation of subtasks 2 and 3 of the Type-3 task, no matter which subtask is calculated locally, we only need to set d_n and c_n as the sum of each subtask before the subtask of the offload calculation.

According to the above formulas, local execution delay and energy consumption can be calculated, and local calculation does not change the load balancing of the surrounding environment.

3.2.2. Offloading model

Delay: When the task is offloaded, the computation offloading process is divided into: the *UE* uploads the data to the *MECS*, the *MECS* calculates the task, and the *MECS* sends the result back to the *UE*. The delay of different task types can be calculated by a combination of the following steps.

The process of uploading data to *MECS* can be divided into two steps: from *UE* to *BS* and from *BS* to *MECS*.

The transmission delay for user to upload *task*(*orsubtask*) data from *UE* to *BS* is given by:

$$t_n^{up1} = \frac{d_n}{r_n} \tag{5}$$

And the delay of *MECS* downloading $Task_n$(*orsubtask*) data from *BS* is given by:

$$t_n^{down1} = d_n \varphi_n \tag{6}$$

Where r_n is the data upload rate from *UE* to *BS*, and φ_n is the transmission rate from the *BS* to the *MECS*, which is proportional to the distance from *BS* to *MECS*. If *MBS* transmits data to *MECS* deployed with it, $\varphi_n = 0$.

Meanwhile, the delay of *MECS* calculation task is given by:

$$t_n^C = \frac{c_n}{f_n} \tag{7}$$

Where f_n is the computing resource allocated by *MECS*. The allocated computing resources cannot exceed the remaining resources.

After the calculation of the previous subtask is completed, the result data needs to be transmitted to the target *MECS* of the next subtask. It can be divided into two steps: from the previous *MECS* to the *BS* and from the *BS* to the target *MECS*.

The delay of the result data from the previous *MECS* to *BS* is given by:

$$t_n^{up2} = d_n' \varphi_n' \tag{8}$$

The delay of transmitting task data from *BS* to the target *MECS* is:

$$t_n^{down2} = d_n' \varphi_n'' \tag{9}$$

At the end of the task calculation, the calculation result needs to be returned to *UE*. Since the returned data size is much smaller than uploaded when the task is offloaded, the delay in the data return process is ignored.

The subtasks of the Type-3 task can be calculated in parallel, so it is necessary to compare the entire period of time between subtask 2 and 3 from the subtask 1 device to upload data to the calculation result send to the subtask 4 device, and take the larger one as the delay of subtask 2 and 3:

$$t_n^{max(2,3)} = \max(t_n^{sub(2)}, t_n^{sub(3)}) \tag{10}$$

When two consecutive subtasks are calculated on the same device, there is no delay in the upload and download process.

Energy consumption The energy consumption includes all devices involved in computation offloading, including *UE* and all used *BS* and *MECS*.

The offloading process can be divided into *UE* to *BS*, *BS* to *MECS*, *MECS* calculation, *MECS* to *BS*, and *BS* to the next device. The next device refers to the device for calculating the next subtask. If it is the last subtask, it will be returned to the user. The energy consumption of different task types can be calculated by a combination of the following steps.

The energy consumption of uploading data form *UE* to *BS* is:

$$e_n^{up1} = P_n^{up-UE} t_n^{up1} \tag{11}$$

Where P is the transmission power when UE sends data. The energy consumption of $MECS$ downloading data from BS is given by:

$$e_n^{down1} = P_n^{down-MECS} t_n^{down1} \tag{12}$$

The energy consumption calculated by $MECS$ is related to the energy efficiency allocated for task z_n^{MECS}. The calculation energy consumption e_n^C is:

$$e_n^C = c_n z_n^{MECS} \tag{13}$$

The energy consumption of the $MECS$ uploading data back to the base station is related to the transmission power of the $MECS$. The transmission energy consumption e_n^{up2} is:

$$e_n^{up2} = P_n^{up-MECS} t_n^{up2} \tag{14}$$

The energy used by the UE to download the result can be ignored. When the subtask is the last one, t_n^{up2} can be ignored, then $e_n^{up2} = 0$. Similarly, when two consecutive subtasks are calculated on the same device, the upload and download process do not consume energy.

Load balancing: Load balancing is the entropy value of the remaining resources of K edge servers around the user, which can be expressed by variance:

$$B_n = \frac{1}{K}\sum_{i=1}^{K}\left(F_i - \frac{1}{K}\sum_{j=1}^{K}F_j\right)^2 \tag{15}$$

When the $Task_n$ is offloaded, the remaining computing resources of $MECS$ change, resulting in a change in load balancing. The load balancing variation is given by:

$$b_n = \Delta B_n = \frac{1}{K}\sum_{i=1}^{K}\left(F_i - \frac{1}{K}\sum_{j=1}^{K}F_j\right)^2 - \frac{1}{K}\sum_{i=1}^{K}\left(F_i' - \frac{1}{K}\sum_{j=1}^{K}F_j'\right)^2 \tag{16}$$

Where K represents the $K-th$ edge server, F represents the computing resources being used by the edge server before the task is offloaded, and F' represents the computing resources being used by the edge server after offloading.

3.3. Algorithm

3.3.1. Problem formulation

Current research tends to optimize for multiple users at the same time. At this time, even all offload will generate a very large state space and action space. When optimizing I users, their status contains information about I tasks, and the number of decisions is up to 2^I. When the number of users is large, the state and action space dimensions are too high. The system model in this paper is partial offloading, which is more complicated than all offloading, and the problem of excessive spatial dimension is more serious.

Since it is difficult to solve the resource allocation problem only by the two optimization goals of delay and energy consumption before the offloading decision is realized, the common method to solve the computational offloading problem is to implement the offloading decision first, and then determine the resource allocation.

Therefore, in order to reduce the state and action space, we choose to train only one user. Train a dedicated model for the constant user based on the location of the user and surrounding edge servers.

At the same time, our method provides a new idea in the order of solving the computational offloading problem. We introduced optimization indicators for load balancing and added task frequency parameters.

When allocating resources, the time to generate tasks affects the task frequency, and the resource occupation after task allocation affects the load of the communication network around the user.

Through the above two factors that affecting the offloading decision of other users, the optimal resource allocation amount for the entire computing offloading environment is obtained, and the impact of the offloading of a single user on the offloading calculation of others is reduced. It is foreseeable that because the edge servers around each user are different, when the load balancing of the edge servers around each user is low, the load on all edge servers tends to be balanced, which is another meaning of multiuser joint optimization.

However, the QoE is not considered, so it is not the amount of resource allocation that can meet user demand and is not the most favorable to the environment. Therefore, it is also necessary to consider the time delay and energy consumption caused by the calculation after the task is offloaded.

In general, the calculation delay and energy consumption and load balancing are optimized in solving the resource allocation problem, and the task data transmission delay and energy consumption are optimized in solving the offloading decision problem.

To clarify the goal of the entire computational offloading problem, describe the input and output of the entire problem.

The K edge servers closest to the user and base stations form the communication network around the user, collect information about the user's surrounding environment, user information and task information, and obtain offloading decisions and computing resources allocated to it. Then for $Task_n$, the input and output are:

$$\begin{cases} x_n = (environment_n, user_n, task_n) \\ y_n = (decision_n, F_n, BS_n) \end{cases} \tag{17}$$

Where $environment_n$ includes the remaining computing resources of each edge server F_n^{left}, the channel occupancy of each base station C^{occupy}; $user_n$ includes the remaining user computing resources f_n^l, and the user energy transmission efficiency z_n^l and user data transmission power P_n^{send}; $task_n$ includes task data volume d_n, calculation volume c_n, arrival time T_n and task type and subtask parameters $type_n$; $decision_n$ represents the offloading decision of each subtask; F_n^d represents the computing resources allocated to each subtask, BS_n^d represents the base station and channel used by each subtask.

By decomposing the NP hard problem (P1), the computational offloading task is divided into resource allocation (P2) and offloading decision (P3). Our proposed reverse two-stage solution method is shown in Fig. 3.

3.3.2. Resource allocation

The purpose of solving the resource allocation problem is to obtain the optimal resource allocation amount that not only satisfies the user QoE but also benefits the subsequent offloading environment by comprehensively considering task frequency, load balancing, calculation delay and energy consumption. As shown in Fig. 4, the resource allocation problem can be seen as a Markov decision process.

MDP state: To interpret the resource allocation problem as a Markov decision process, we define the MDP state for $Task_n$ as follows:

$$s_n = \left(F_n^{left}, user_n, task_n\right) \tag{18}$$

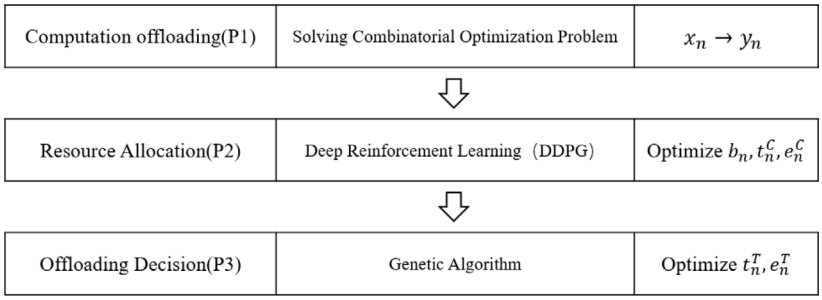

Fig. 3. The two-step optimization model for solving (P1).

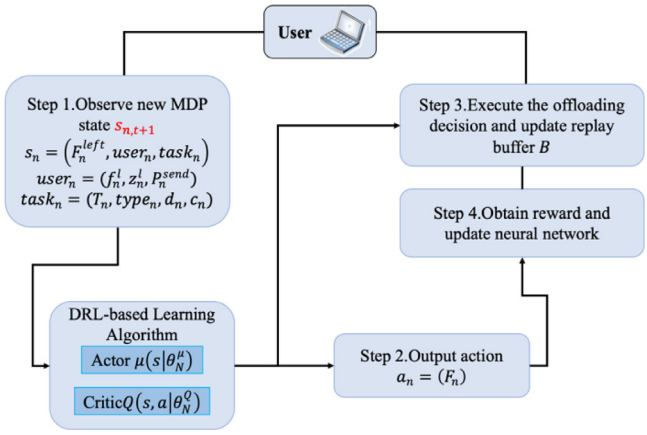

Fig. 4. Resource allocation(P2).

In particular, the MDP state s_n is a 1×(K+3+10) vector, which includes user information, task information and the remaining computing resources of each edge server. Specifically, it is the same as Section 3.3.1:

$$user_n = (f_n^l, z_n^l, P_n^{send}) \tag{19}$$

$$task_n = (T_n, type_n, d_n^{sub1}, d_n^{sub2}, d_n^{sub3}, d_n^{sub4}, c_n^{sub1}, c_n^{sub2}, c_n^{sub3}, c_n^{sub4}) \tag{20}$$

MDP action: We assume that the task is offloaded to all surrounding edge servers, and the optimal resource allocation is obtained in turn, as the MDP action a_n

$$a_n = (F_n) \tag{21}$$

Compared with the method of considering multiuser and solving the offloading decision-making problem and resource allocation problem together, our method can greatly reduce the dimensions of action space and state space, accelerate the convergence speed of deep reinforcement learning algorithm, and improve the training efficiency.

Reward function: In order to obtain the optimal resource allocation quantity which not only guarantees the QoE but also benefits the subsequent task offloading. The reward function includes calculation delay, calculation energy consumption, load balancing and task frequency:

Among them, load balancing and task frequency are used to reduce the offloading cost of subsequent tasks, such as reducing the allocated resources when the task frequency is high. It is foreseeable that if we only consider load balancing, we will always tend to equalize the remaining resources of the surrounding edge servers when allocating resources. This will definitely greatly affect the QoE, especially in terms of latency.

Therefore, optimization indicators that guarantee user experience must be added to the reward function.

The amount of resource allocation is only related to calculation delay and energy consumption, and does not affect transmission delay and energy consumption, so these parameters are added to our reward function. The final reward function is as follows

$$Cost_{n1} = \omega_C^t t_n^C + \omega_C^e e_n^C + \omega^b b_n + \omega^\rho \rho_n + I^{finish} + I^{fail} \tag{22}$$

Where ω^t, ω^e, ω^b, ω^ρ are artificially selected weight coefficients, I^{finish} and I^{fail} are rewards or penalties given when the task succeeds or fails, through DDPG [17] obtains the optimal resource allocation.

With the above optimization methods, two neural networks are used to learn the offloading policy. To stabilize the learning procedure, an actor network, a critic network, and a replay memory are used. Besides, an exploration policy $a_N = \mu\left(s_N|\theta_N^\mu\right) + \Delta\mu$ for actor and an ε-greedy policy for critic are used in the training procedure to balance the tradeoff between exploration and exploitation. $\Delta\mu$ is a noise process [31]. Algorithm 1 illustrates the training procedure of IO-DDPG for resource allocation.

Algorithm 1 Training Stage for IO-DDPG for Resource Allocation in Mobile Edge Computing

Resource allocation (P1):

Randomly initialize the actor network $\mu(s|\theta^\mu)$ and the critic network $Q(s,a|\theta^Q)$;

Initialize the associated target networks with weights $\theta^{\mu\prime} \leftarrow \theta^\mu$ and $\theta^{Q\prime} \leftarrow \theta^Q$;

Initialize the experience replay buffer B;

for each episode k=1, …, K_{max} **do**

for task number N=1, …, N_{max} **do**

Reset system model parameters for the MEC environment and task information as the initial status s_N;

for each time slot t=1, …, T_{max} **do**

Determine the power for computation offloading by selecting an action $a_{N,t} = \mu\left(s_{N,t}|\theta_N^\mu\right) + \Delta\mu$ using running the current policy network θ_N^μ and generating exploration noise $\Delta\mu$.

Execute action $a_{N,t}$ independently at the user agent, and then receive reward $r_{N,t}$ and observe the next state $s_{N,t+1}$ from the environment;

Collect and save the tuple $(s_{N,t}, a_{N,t}, r_{N,t}, s_{N,t+1})$ into the replay buffer B_m;

Randomly sample a mini-batch of I tuples $\{(s_i, a_i, r_i, s_i')\}_{i=1}^I$ from B_m;

Update the critic network $Q\left(s,a|\theta_N^Q\right)$ by minimizing the loss L with the samples:

$$L = \frac{1}{I}\sum_{i=1}^{I}\left(r_i + \max_{a\in\mathcal{A}} Q\left(s_i', a \mid \theta_N^{Q'}\right) - Q\left(s_i, a_i \mid \theta_N^Q\right)\right)^2;$$

Update the actor network $\mu\left(s|\theta_N^\mu\right)$ by using the sampled policy gradient:

$$\nabla_{\theta_N^\mu} J \approx \frac{1}{I}\sum_{i=1}^{I} \nabla_a Q\left(s_i, a \mid \theta_N^Q\right)\Big|_{a=a_i} \nabla_{\theta_N^\mu}\mu\left(s_i \mid \theta_N^\mu\right);$$

Update the target networks by $\theta_N^{\mu'} \leftarrow \tau\theta_N^\mu + (1-\tau)\theta_N^{\mu'}$ and $\theta_N^{Q'} \leftarrow \tau\theta_N^Q + (1-\tau)\theta_N^{Q'}$

end for

end for

end for

3.3.3. Offloading decision

After obtaining the optimal resource allocation for task offloading on all edge servers, the action space for all offloading decisions is fixed, including only the offloading position of each

Algorithm 2 Training Stage for the IO-DDPG for Offloading Decision in Mobile Edge Computing

Offloading decision（P2）:

Initialize the cross probability P_c, mutation probability P_m, population size M;

for task number N=1, …, N_{max} **do**

Reset system model parameters for the BS environment and user information as part of the initial status s_{N_1};

Obtain the optimal resource allocation F_N for offloading tasks to each edge server from the resource allocation (P1) as part of the initial status s_{N_2};

Combine s_{N_1} and s_{N_2} to get the initial state s_N

Randomly generate the first-generation population P_1;

Calculate the fitness of each individual in the population P_1 by the fitness function F:

$$F = \omega_T^t t_N^T + \omega_T^e e_N^T + \omega_C^t t_N^C + \omega_C^e e_N^C$$

for generation number $G = 2, \ldots, G_{max}$ **do**

Generate new-generation population P_G by proportionally select individuals from the population P_{G-1} for mutation and crossover through comparing fitness $F(G-1)$ until the fitness reaches the maximum;

end for

end for

subtask and the base station and channel used by each subtask. Therefore, only an optimal search is required to obtain the optimal offloading decision. Genetic Algorithm always has good results in solving such search problems. The optimal offloading decision is obtained by setting the fitness function of the genetic algorithm, and the optimization goals are transmission delay and energy consumption. However, the transmission delay and calculation delay of some types of tasks cannot be separated, so the calculation delay and energy consumption must also be optimized together. Therefore, by weighting, the fitness function is determined as:

$$Cost_{n2} = \omega_T^t t_n^T + \omega_T^e e_n^T + \omega_C^t t_n^C + \omega_C^e e_n^C \tag{23}$$

Algorithm 2 illustrates the training procedure of IO-DDPG for offloading decision.

3.3.4. Decision time

The genetic algorithm finds the optimal solution through iteration, and its decision time is too long to meet the real-time requirements of computation offloading. Therefore, we use a three-layer fully connected neural network to fit the output decision y_n and input information x_n obtained by the genetic algorithm.

Since the actual decision output is a discrete value, such as which edge server to choose. In order to improve the training speed and network accuracy, a layer of discretization is added behind the fully connected neural network. Select the discretization method so that the continuous value of the output is equal to the nearest integer within a limited range. The loss function is cross entropy error function between the neural network output after the discrete and the decision searched by the genetic algorithm:

$$Loss = -\frac{1}{\alpha}\sum_{d}\sum_{i}\sum_{c=1}^{\beta} y_{ic} log(p_{ic}) \tag{24}$$

Where α is the number of samples, β is the number of categories, y_{ic} (0 or 1) represents whether the sample i is the same as the data for category c, pic represents the predicted probability that sample i belongs to category c, and d is the number of decision categories.

In this section, we propose an inverse order based optimization method for task offloading and resource allocation in mobile edge computing. By separating the continuous space (resource) and discrete space (selected server and channel) in the decision space and optimizing single user through load balancing, the huge state space and action space due to multiuser and multitask are avoided. Finally, the problem of long processing time caused by iterative optimization of search algorithm is solved by training FCN.

4. Experiment

In this section, we conduct extensive simulations to evaluate the performance of the proposed algorithm. In order to accurately show the effectiveness and superiority of our algorithm in complex MEC scenarios, we compare our method with DDPG, PS-DDPG [30], and D-DRL [11] methods, referring the total reward, energy cost, time cost and load balancing.

We have studied in detail the impact of task density, edge server computing resources, task data volume and computing volume on time consumption, energy consumption and load balancing, confirming the advantages of our proposed method in terms of long-term benefits.

We consider an edge offloading system composed of K=5 edge servers and 5 UEs. The number of edge servers and UEs is variable, however, the experimental workload will increase exponentially without gaining additional research value. Considering that tasks in edge computing are mostly compute-intensive and time-sensitive tasks, user equipment parameters, task data

Table 2
Parameter values used in the simulations.

Parameter	Value
Episode	8000
Replay buffer size	1000000
Mini-batch size	128
Actor network learning rate	1×10^{-8}
Critic network learning rate	1×10^{-7}
Return discount factor	0.99
Critic network update frequency	1×10^{-5}
Computational capacity of the user device	0.5∼2 GHz
Computational capacity of the edge server	5∼20 GHz
Channel bandwidth	2 MHz
Channel gain	[−4, −6, −8, −10] dB
Task type generation probability	[1/3, 1/3, 1/3]
Task data size	0.2∼10MB
Task calculation size	1∼50 GHz
Task frequency	10∼30/100s
User transmission power	3∼8 W
User energy transmission efficiency	1∼ 4W/GHz^{-2}

volume, and task calculation volume are set accordingly. The edge server device parameters and base station parameters are set based on actual information. Table 2 summarizes the detailed parameter settings.

4.1. Comparison of total reward on strategies

In this subsection, we evaluate the comprehensive performance of four different strategies including DDPG, PS-DDPG, D-DRL and IO-DDPG. We first chose total reward as the metrics of strategies performance since the progress on energy consumption, time cost and load balancing will directly lead to the increase of reward. Larger reward means better performance.

The task frequency we set for each UE is 20 tasks/100 s on average, the task data size and calculation size are 2MB and 10 GHz on average, and the total resource of each edge server is 20 GHz. We collect the whole tasks created in 1000 s and train them using Pytorch for total 8000 episodes. The convergence curve of five strategies has been plotted in Fig. 5.

The results show that our strategy can make better decisions for the same sequence. Specifically, the reward of IO-DDPG can be close to the highest value at 4000 episodes, but DDPG and D-DRL require about 5000 episodes. Although PS-DDPG only needs about 3000 episodes to reach the maximum reward, our strategy achieved the highest reward among all methods, followed by D-DRL, PS-DDPG and DDPG in order.

Most of the research does not add load balancing to the reward function, and it is not complete to use the total reward alone to discuss and compare. At this time, we cannot be sure that our method is better. Therefore, we will decompose and analyze the rewards in the following subsections, and analyze the effects of task frequency, computation capacity and task density on delay, energy consumption, and load balancing in turn.

4.2. Influence of task frequency on energy and time cost

In this section, we analyze the impact of different task frequency on the energy cost and time cost of the strategy. At this time, the MEC's computational capacity and average task data size and calculation size need to be frozen. Set the MEC's computational capacity to 20 GHz, and the average task data size and calculation size to 2MB and 10 GHz. We run our experiment on five task frequencies, 10, 15, 20, 25, and 30 tasks every 100 s per UE. The frequency change from 10 to 30 involves low workload conditions and high workload conditions. The performance comparison between frequency of different tasks helps to verify the stability of the proposed method.

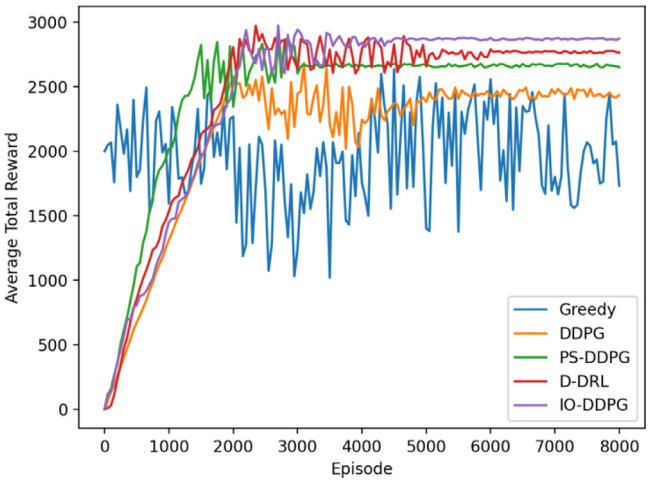

Fig. 5. Performance evaluation in terms of total reward.

In Fig. 6, we plot the energy cost and time cost given the five tasks frequency conditions. At frequencies 10, 15 and 20, all methods can make decisions normally. At this time, our method IO-DDPG is at the maximum in terms of delay and energy consumption, but it is not far behind other methods. Because even when the task density is not high, IO-DDPG is more inclined to maintain the stability of the offloading environment to face the task frequency that may change at any time in the actual environment. In the decision-making process, separating more divisible tasks produces more calculation delay, transmission delay, transmission energy consumption, and more calculation delay for indivisible tasks and less allocation of resources.

When the task frequency gradually increases to 25 and 30, due to the three methods of DDPG, PS-DDPG and D-DRL allocating too many resources for each task, which makes the next task unable to offload normally, and more and more tasks are backlogged. Since the task will be solved sooner or later, the energy consumption will basically increase in proportion, but the time delay will increase significantly. D-DRL is slightly better among the three methods due to the consideration of distributed offloading. IO-DDPG is based on partial offloading, which allows us to offload divisible tasks to different edge servers.

The increase in task frequency promotes more separating tasks in the decision-making of IO-DDPG, resulting in more transmission delay and transmission energy consumption, but the high-quality offloading environment enables more computation resources for each subtask, which greatly reduces the calculation delay. This also releases resources faster for the next tasks, forming a virtuous circle. Therefore, IO-DDPG has more energy consumption than the other three methods, but obtains a shorter time delay. After normalization, our method produces the best QoE.

4.3. Influence of task data and calculation size on energy and time cost

In this section, we analyze the impact of different task data and calculation sizes on the energy cost and time cost of the strategy. Set the computing power of the MEC to 20 GHz and the task frequency to 20/100s per UE. We ran our experiments under four conditions, 1MB/5 GHz, 2MB/10 GHz, 5MB/25 GHz, and 10MB/50 GHz for each task. The change in task data size from 1MB to 10MB affects the transmission delay and energy consumption, while the change in task calculation size from

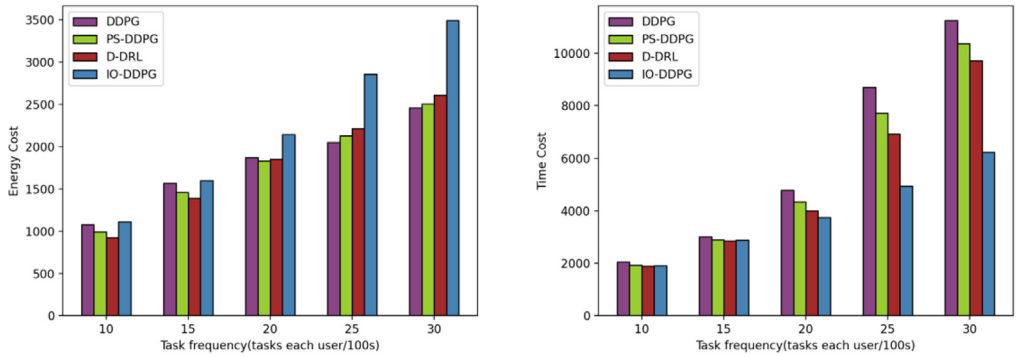

Fig. 6. Energy/time cost comparison on different task frequency.

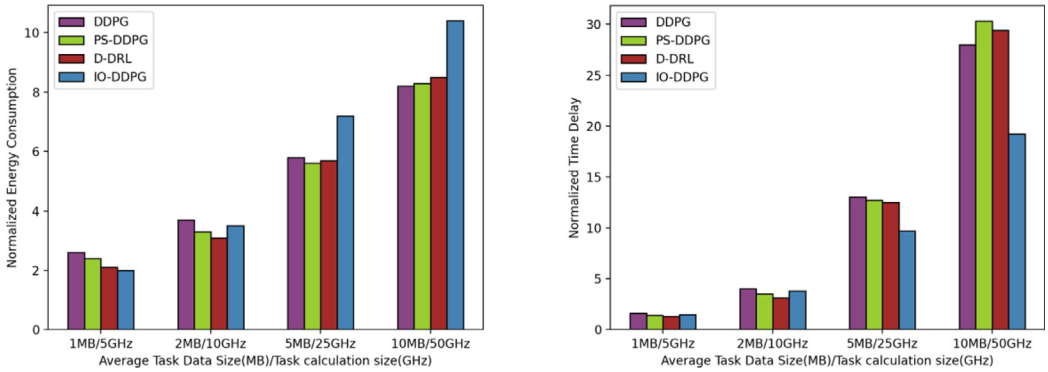

Fig. 7. Energy/time cost comparison on different task data and calculation size.

5 GHz to 50 GHz affects the calculation delay and energy consumption. The performance comparison between different task data and calculation sizes helps to verify the self-adjustment and adaptability of the proposed method in the face of lightweight tasks and heavyweight tasks.

As shown in Fig. 7, we plot the comparison of energy cost and time cost in the four states. When the average task data and calculation sizes are less than 2MB/10 GHz, all methods normally select the optimal offloading strategy. Our method IO-DDPG does not pursue the shortest delay when solving lightweight tasks, but is the weight that may arrive Level tasks retain more computing space, and more partial offloading of tasks increases transmission delay and energy consumption. Therefore, in terms of delay and optimization of energy consumption did not reach the best level, but it is not far behind other methods.

When the average task data and calculation sizes reach 5MB/25 GHz, the advantages of the above-mentioned reflected. The average task data and calculation sizes reaching 5MB/25 GHz can be regarded as a set of test sets of 2MB/10 GHz and 10MB/50 GHz, that is, a mixture of lightweight tasks and heavyweight tasks. The other three methods allocate too many computation resources to lightweight tasks, resulting in insufficient computation resources to complete the heavyweight tasks that follow quickly. Therefore, at this time, the IO-DDPG reaches the optimum in terms of delay optimization. Although the energy consumption of transmission is increased, the optimal QoE is obtained overall.

And when the average task data and calculation sizes reach 10MB/50 GHz, the advantages of our method in the face of a large number of heavyweight tasks are shown. The three methods of DDPG, PS-DDPG and D-DRL face continuous heavyweight tasks, which will cause allocable resources to become idle just now, and a large amount of resources are immediately allocated to the next task. In good times and bad offloading environment, there is no guarantee that the next task can be completed quickly. IO-DDPG divides the heavyweight tasks into several lightweight tasks and distributes them to multiple edge servers for offloading, which greatly reduces the calculation delay and enables users to obtain the best experience.

4.4. Influence of computational capacity on energy and time cost

In this section, we analyze the impact of different MEC computational capacity on the energy cost and time cost of the strategy. Set the task frequency to 20/100s per UE, and the average task data size and calculation size to 2MB and 10 GHz. We conduct our experiments based on four computational capacities, 5 GHz, 10 GHz, 15 GHz and 20 GHz. The performance comparison of computational capacity from 5 GHz to 20 GHz verifies the flexibility of the proposed method in terms of computing resource allocation.

In Fig. 8, we plot the energy cost and time cost of the four situations of computational capacity. At MEC computational capacity 20 GHz, all methods can make decisions normally. In this state, our method IO-DDPG is only slightly inferior to D-DRL in terms of delay and energy consumption.

But when the MEC computational capacity gradually decreases, we can draw the conclusion:

(1) The energy consumption of the DDPG method changes very little, but the time delay increases significantly. This shows that the method cannot more appropriately allocate the task offloading position and change the resource allocation method when the available computational capacity changes.

(2) PS-DDPG and D-DRL have a certain increase in energy consumption, indicating that these two methods are trying to better schedule task offloading positions, but the rising time delay indicates that they have not achieved very good results. D-DRL achieved a slightly higher effect than PS-DDPG through partial offloading.

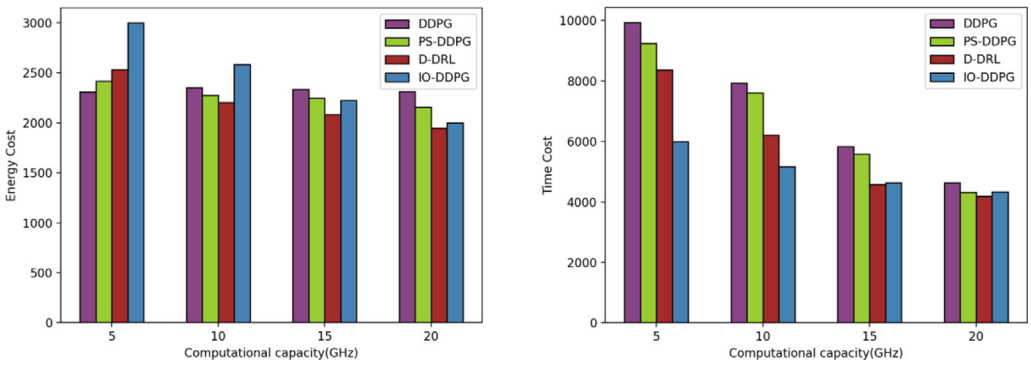

Fig. 8. Energy/time cost comparison on different computational capacity.

Table 3
Performance evaluation in terms of load balancing.

Method	Load balancing variance						Resource occupancy rate
	2MB/10 GHz	10MB/50 GHz	20/100s	30/100s	20 GHz	5 GHz	
DDPG	6.73	20.87	11.27	23.13	7.43	4.33	20%–30%
PS-DDPG D-DRL	6.17	21.23	12.85	22.32	6.58	3.87	20%–30%
	4.46	9.23	8.75	14.37	5.11	3.18	15%–20%
IO-DDPG	**1.38**	**3.37**	**3.66**	**2.41**	**1.77**	**2.34**	**8%–12%**

(3) IO-DDPG uses partial offloading to reallocate subtasks to edge servers that are more suitable for offloading. It saves more time cost for the transmission process by using more energy cost, even if the available computational capacity is small, it always maintains a high-quality offloading environment.

4.5. Influence on load balancing

In the previous section, we analyze the impact of task frequency, task data, calculation scale, and computational capacity on energy and time cost. We have mentioned many times that our method creates a better offloading environment, realizes long-term benefits, can better face special conditions, and has better robustness. Next, we will verify the advantages of our method through changes in load balancing when task frequency, task data, calculation scale, and computational capacity change.

As shown in Table 3, our method can obtain the lowest load balancing variance regardless of whether it is in a low workload state or a high workload state, which shows that our method balances the load of surrounding edge servers at all times. Resource occupancy rate is another indicator that demonstrates our method. It represents the amount of resources allocated to each task (subtask). Through partial offloading, we provide only 8%–12% of the total resources for each subtask (or indivisible task). But it exceeds the QoE obtained by the DDPG and PS-DDPG methods using 20%–30% of the total resources.

5. Conclusion

In this paper, we propose a task offloading and resource allocation algorithm based on inverse order method named IO-DDPG for computation offloading in MEC, which address the challenges of continuous–discrete hybrid action spaces and coordination among users. We separate the continuous–discrete mixed action space in the offloading problem to form a two-step offloading framework, exchange the calculation order to reduce the dimensionality of the action and state space, accelerating the convergence speed of the algorithm. By introducing load balancing, each user is allowed to maintain a high-quality offloading environment to obtain long-term interests. This enables our method to achieve better results in the face of sudden changes in task frequency, computational capacity, task data and calculation size. Simulation results show that compared with the baseline method, the proposed algorithm is more stable, flexible, adaptable and is more suitable for practical applications.

CRediT authorship contribution statement

Junyao Yang: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Yan Wang:** Conceptualization, Resources, Writing – review & editing, Supervision, Project administration. **Zijian Li:** Software, Investigation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the National Key Research and Development Project of China [Grant numbers 2018YFB2003501] and National Nature Science Foundation of China [Grant numbers 61320106010, 61573019, 61627810].

References

[1] X. Peng, J. Ren, L. She, et al., BOAT: A block-streaming app execution scheme for lightweight IoT devices, IEEE Internet Things J. (2018) 1.

[2] Y. Mao, C. You, J. Zhang, et al., A survey on mobile edge computing: The communication perspective, IEEE Commun. Surv. Tutor. (99) (2020) 1–10.

[3] M. Satyanarayanan, The emergence of edge computing, Computer 50 (1) (2017) 30–39.

[4] J.W. Rittinghouse, J.F. Ransome, Cloud Computing: Implementation, Management, and Security, CRC Press, Boca Raton, FL, USA, 2017.

[5] S. Verma, Y. Kawamoto, Z.M. Fadlullah, et al., A survey on network methodologies for real-time analytics of massive IoT data and open research issues, IEEE Commun. Surv. Tutor. 19 (3) (2017) 1457–1477.
[6] X. Chen, L. Jiao, W. Li, et al., Efficient multi-user computation offloading for mobile-edge cloud computing, IEEE/ACM Trans. Netw. 24 (5) (2016) 2795–2808.
[7] P. Mach, Z. Becvar, Mobile edge computing: A survey on architecture and computation offloading, IEEE Commun. Surv. Tutor. (99) (2020) 1–10.
[8] G. Tefera, K. She, M. Chen, et al., Congestion-aware adaptive decentralised computation offloading and caching for multi-access edge computing networks, IET Commun. 14 (19) (2020) 3410–3419.
[9] Z. Ning, P. Dong, X. Wang, et al., Partial computation offloading and adaptive task scheduling for 5G-enabled vehicular networks, IEEE Trans. Mob. Comput. (99) (2020).
[10] H. Cao, J. Cai, Distributed multiuser computation offloading for cloudlet-based mobile cloud computing: A game-theoretic machine learning approach, IEEE Trans. Veh. Technol. 67 (1) (2018) 752–764.
[11] Y. Zhan, S. Guo, P. Li, et al., A deep reinforcement learning based offloading game in edge computing, IEEE Trans. Comput. 69 (6) (2020) 883–893.
[12] Y. Zhang, J. Fu, Energy-efficient computation offloading strategy with tasks scheduling in edge computing, Wirel. Netw. (2020) 1–12.
[13] R. Lin, Z. Zhou, S. Luo, et al., Distributed optimization for computation offloading in edge computing, IEEE Trans. Wireless Commun. (99) (2020) 1–10.
[14] X. Zhao, J.H. Peng, W. You, et al., A privacy aware computation offloading method mased on lyapunov optimization, J. Electr. Inform. Technol. 42 (3) (2020) 704–711.
[15] L. Ji, G. Hui, T. Lv, et al., Deep reinforcement learning based computation offloading and resource allocation for MEC[C]//, in: 2018 IEEE Wirel. Commun. Netw. Conf., WCNC, IEEE, 2018.
[16] V. Mnih, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.
[17] T.P. Lillicrap, et al., Continuous control with deep reinforcement learning, in: Proc. Int. Conf. Learn. Represent., San Juan, Puerto Rico, 2016, pp. 1–14.
[18] R.S. Sutton, A.G. Barto, Reinforcement learning, in: A Bradford Book, Vol. 15, No. 7, 1998, pp. 665–685.
[19] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double Q-learning, in: Proc. AAAI Conf. Artif. Intell., Phoenix, AZ, USA, Feb. 2016, pp. 2094–2100.
[20] J. Xiong, et al., Parametrized deep Q-networks learning: Reinforcement learning with discrete-continuous hybrid action space, 2018, [Online]. Available: arXiv:1810.06394.
[21] A. Dp, B. Ws, C. Nw, Automatic ship classification for a riverside monitoring system using a cascade of artificial intelligence techniques including penalties and rewards, ISA Trans. (2021).
[22] R. Lowe, Y. Wu, A. Tamar, J. Harb, O.P. Abbeel, I. Mordatch, Multi-agent actor–critic for mixed cooperative-competitive environments, in: Proc. Adv. Neural Inf. Process. Syst., Long Beach, CA, USA, Dec. 2017, pp. 6379–6390.
[23] X. Qiu, L. Liu, W. Chen, et al., Online deep reinforcement learning for computation offloading in blockchain-empowered mobile edge computing, IEEE Trans. Veh. Technol. (99) (2020) 1–10.
[24] J. Zou, T. Hao, C. Yu, et al., A3C-DO: A regional resource scheduling framework based on deep reinforcement learning in edge scenario, IEEE Trans. Comput. (99) (2020) 1–10.
[25] B. Huang, Y. Li, Z. Li, et al., Security and cost-aware computation offloading via deep reinforcement learning in mobile edge computing, Wirel. Commun. Mob. Comput. 2019 (2019) 1–20.
[26] P. Dong, X.X. Wang, J.J.P.C. Rodrigues, et al., Deep reinforcement learning for vehicular edge computing: An intelligent offloading system, ACM Trans. Intell. Syst. Technol. (TIST) 10 (6) (2019).
[27] H. Lu, X. He, M. Du, et al., Edge QoE: Computation offloading with deep reinforcement learning for internet of things, IEEE Internet Things J. (99) 1.
[28] Q. Zhou, K. Wang, S. Guo, et al., Falcon: Towards computation-parallel deep learning in heterogeneous parameter server[C]//, in: 2019 IEEE 39th International Conference on Distributed Computing Systems, ICDCS, IEEE, 2019.
[29] Z. Wei, B. Zhao, J. Su, X. Lu, Dynamic edge computation offloading for internet of things with energy harvesting: A learning method, IEEE Internet Things J. 6 (3) (2019) 4436–4447.
[30] X. He, H. Lu, M. Du, et al., QoE-based task offloading with deep reinforcement learning in edge-enabled internet of vehicles, IEEE Trans. Intell. Transp. Syst. (99) (2020) 1–10.
[31] S. Li, S. Bing, S. Yang, Distributional advantage actor-critic, 2018, CoRR, [Online]. Available: https://arxiv.org/abs/1806.06914.