Image Processing, Electronics and Computers
L. Trajkovic et al. (Eds.)

doi:10.3233/ATDE240511

GPER-DDPG-Based Offloading Optimization for UAV-Assisted Mobile Edge Computing

Siyuan YU, Xuxiu ZHANG[1] and Zi GUO
School of Automation and Electrical Engineering, Dalian Jiaotong University, Dalian 116052, PR China

Abstract. In recent years, the technology related to providing computing resources to ground users using Unmanned Aerial Vehicle (UAV) with Mobile Edge Computing (MEC) servers has received widespread attention. In edge computing scenarios with limited communication infrastructure, for the problem of difficulty in deploying conventional terrestrial base stations, this paper designs a UAV-based edge computing network (UAV-MEC) model. The network model adopts a partial offloading strategy, which enables the computational tasks of the ground devices (LD) to be partially executed on the local devices and partially offloaded to the edge computing servers of the auxiliary UAV. When facing computationally intensive and delay-sensitive computational tasks, LD and UAV have limited computational resources and energy, and how to optimize the offloading strategy becomes a key issue. In this paper, we propose an offloading optimization algorithm based on GPER-DDPG. The algorithm introduces the prioritized experience replay technique, adaptive action policy noise, and Actor network delay updating technique to improve the stability and convergence performance of the DDPG algorithm in complex environments. Experimental results show that the performance of the improved algorithm proposed in this paper improves about 10-12% over the performance of the DDPG algorithm.

Keywords. Unmanned aerial vehicle, mobile edge computing, DDPG, prioritized experience replay, adaptive action policy

1. Introduction

In recent years, with the rapid development of virtual reality, Internet of Things (IoT), autonomous driving and other technologies, intelligent devices based on IoT devices have spread all over our lives. However, IoT devices are limited by factors such as size, computational resources, and production costs, making their computational capabilities unable to meet production demands, which greatly reduces the performance of computational tasks to process tasks on local devices [1,2].

Edge computing allows computing tasks to be migrated from traditional cloud computing centers to edge devices closer to data sources and end users [3] to reduce latency and improve system responsiveness, which largely solves the network congestion problem of traditional cloud computing due to centralized processing of tasks [4]. In addition, edge computing has shorter data transmission distance and time and less energy

[1] Corresponding Author: Xuxiu ZHANG, zhangxuxiu@163.com.

consumption compared with cloud computing. In recent years, scholars at home and abroad have shown extensive attention to the computational offloading problem in edge computing scenarios: literature [5] designed a dynamic resource offer mechanism by applying a distributed game mechanism and combining it with the Liapunov optimization theory, successfully realizing the differentiated control of different business types as well as the elastic on-demand allocation of computational resources, and this method effectively reduces the average latency of the Internet of Things (IoT) system. Literature [6] proposed a deep reinforcement learning-based joint optimization approach for SWIPT (Simultaneous Wireless Information and Power Transfer) edge networks, focusing on solving the energy harvesting and computational task offloading problems in IoT. They proposed a joint optimization model that considers factors such as power allocation, CPU frequency, offloading weights, and energy harvesting weights. Literature [7] proposed a distributed task offloading and resource allocation algorithm based on deep reinforcement learning, which successfully solves the task offloading and resource allocation problems in vehicular edge computing. The algorithm has significant advantages in terms of latency, energy consumption, and task completion rate. Literature [8] considered different task attributes, user mobility, and time delay constraints by simulating a mobile edge scenario. Based on the user's mobility trajectory, the objective is modeled as finding a MEC server optimization model that satisfies the time delay constraints and produces minimum energy consumption during the offloading process, and a minimum energy consumption offloading algorithm is proposed, which significantly reduces the energy consumption and delay of the task offloading process. However, the current traditional edge computing servers are generally fixed at the time of deployment, which sometimes fails to meet the user requirements.

As a flexible mobile computing platform, UAV can be used to provide computing resources and data transmission services. Currently, UAV have been applied in some resource-constrained scenarios, such as harsh environment monitoring, mine exploration, and seismic evaluation [9]. Deploying edge computing servers to UAV, combined with the advantages of UAV communication technology, can largely make up for the short board of poor flexibility of traditional MEC, and greatly improve the performance of MEC task processing. Literature [10] proposes a deployment optimization and computation offloading model for air-air-heaven integrated mobile edges, using UAV as edge computing servers and near-Earth satellites as cloud servers, and a two-layer algorithm to solve the deployment and offloading problems respectively. Although there have been some works on UAV-assisted edge computing, most of them only consider the scenarios in which the UAV is used as a base station, and do not study the flight state of the UAV, which can not be directly applied to real-world scenarios.

Aiming at the problem of difficulty in deploying ground base stations and seeking the optimal offloading decision of the system, this paper combines two technologies, edge computing and UAV, and proposes a UAV-assisted edge computing network, which is optimized for the maximum processing delay of the system using the GPER-DDPG algorithm. Compared to traditional MEC networks, the performance of MEC networks integrated with UAV communication technology is improved. UAV are characterized by excellent flexibility and ease of deployment, and thus are able to provide communication services to a designated area without being restricted by the terrain on the ground. In addition, UAV, with their unique mobility, are able to flexibly adjust their deployment position to be closer to the area required for providing computing services, depending on the demand situation of the service recipients. The GPER-DDPG algorithm introduces a prioritized experience playback technique, an adaptive action policy noise,

and an Actor network latency updating technique based on the DDPG, and the algorithm learns the user scheduling from the experience, UAV's flight state, and task offloading ratio, which greatly reduces the computational complexity. By comparing the existing algorithms, the experimental results show that the performance of the improved algorithm proposed in this paper is about 10~12% higher than that of the DDPG algorithm.

2. Network Model and Problem Definition

2.1. Network Model Description

For a multi-user UAV-assisted edge computing system, the system model diagram is shown in Figure 1. The system consists of multiple ground-local device LDs and a UAV equipped with a MEC server, and each LD in this system follows a partial offloading policy. The system divides the entire time period equally into k time periods of length T [11-13]. In each time period T, there are N moving LDs randomly distributed in the working area of the system, and the ith local device is denoted as LD_i, i ($i \in \{1, 2, ..., N\}$), whose 3D coordinates are $w_i(k) = [x_{i(k)}, y_{i(k)}, 0], i \in \{1, 2, ..., N\}$,Each LD is equipped with a data transmission device and a local server in order to transmit data or process computational tasks.

The UAV's position remains fixed during the time period T and communicates with one of the ground devices in association with it, and the ground device LD_i with which it communicates can offload part of the task to the MEC server carried by the UAV for processing, and the remaining computational tasks are processed at the local server [14]. The position coordinates of the UAV at the beginning of the time period are $z(k)=[x_{(k)}, y_{(k)}, H]$, and the end coordinates are $z(k+1)=[x_{(k+1)}, y_{(k+1)}, H]$, where H is the fixed flight altitude of the UAV, which is kept constant in the paper. In this paper, it is assumed that the task computational power of the edge computing server carried on the UAV is much larger than the computational power of each LD and the results processed by the edge server are small, and its return delay is negligible.

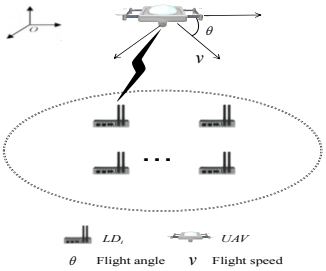


Figure 1. System model diagram.

2.2. Computational Model

In this paper, the communication channel between UAV and ground equipment LD_i is modeled as a line-of-sight channel link model, and the wireless channel gain can be quantified as

$$h_i(k)=d_{i(k)}^2 h_o=\frac{h_o}{\left(x_{(k+1)}-x_{i(k)}\right)^2+\left(y_{(k+1)}-y_{i(k)}\right)^2+H^2} \tag{1}$$

where h_o is the size of the channel gain per unit distance (d=1m).$d_{i(k)}$ is the distance to the LD_i in the kth time slot, the magnitude of which is derived from the Euclidean distance formula in three-dimensional space.

From Shannon's theorem, the data transfer rate of the task offloading process can be expressed as

$$r_i(k)=B\log_2\left(1+\frac{p_i h_i(k)}{\sigma^2+h_{si}(k)}\right) \tag{2}$$

where B is the channel width, σ^2 is the channel noise interference, p_i is the data transmission power of LD_i , and $h_{si}(k)$ denotes the transmission loss, which acts as an indicator of the loss when the communication between the UAV and the ground equipment is obstructed.

The computational delay of the system is mainly composed of two parts, one is the local computational delay and the other is the offloading computational delay, which is further divided into the task transmission delay and the task processing delay, the specific model is as follows

(1) Local computing: In local computing mode, the total delay incurred by local computing is:

$$t_{loc,i(k)}=\frac{\left(1-a_i(k)\right)I_i(k)S}{f_{LD}} \tag{3}$$

Where, $I_i(k)$is the total number of computation tasks of the LD_i, $\alpha_i(k)$is the task offloading ratio,f_{LD} is the CPU computation frequency of each ground device LD, the total number of local tasks to be computed can be computed as $(1-\alpha_i(k))I_i(k)$, and S is the computational complexity i.e., the CPU cycles required to process each unit byte.

(2) Offloading computation: In the case of edge computing, the delay generated by the processing task consists of two parts, i.e., the delay of transmitting the task to be processed to the UAV and the delay of processing the task on the UAV, and the return delay is ignored. According to the above analysis, combined with equations (1) and (2), the task transmission delay and processing delay can be expressed as respectively:

$$t_{tra,i(k)}=\frac{\alpha_i(k)I_i(k)}{r_{\mathrm{i}}(\mathrm{k})} \tag{4}$$

$$t_{MEC,i(k)} = \frac{\alpha_i(k) I_i(k) S}{f_{MEC}} \tag{5}$$

f_{MEC} in Eq. (5) is the edge server CPU computation frequency on the UAV. the computational energy consumption of the MEC server can be expressed as:

$$E_{MEC,i(k)} = P_{MEC,i(k)} \mathrm{t}_{MEC,i(\mathrm{k})} \tag{6}$$

$$P_{MEC,i(k)} = \chi f_{MEC}^3 \tag{7}$$

where χ is the CPU energy consumption factor of the MEC server.
Considering the position change of the UAV in the kth time slot T, the flight energy consumption of the UAV is expressed as:

$$E_{UAV,(k)} = \beta \left\| v_k \right\|^2 \tag{8}$$

where β denotes the energy consumption coefficient, which is a fixed value [15].

2.3. Definition of the Problem

Considering that the system proposed in this paper contains multiple ground devices, the scheduling of users and UAV as well as the offloading rate of tasks need to be jointly optimized to minimize the maximum processing delay of the system, and the optimization problem of joint scheduling and computational offloading can be formulated as follows.

$$\mathrm{P1}: \min_{i(k),z(k+1),\alpha_i(k)} \sum_{k=1}^{k} \sum_{i=1}^{N} i(k) \max\{t_{loc,i(k)}, t_{tra,i(k)} + t_{MEC,i(k)}\} \tag{9}$$

$$\text{s.t.} \quad \mathrm{C1}: i(k) \in \{0,1\}, \forall k \in \{1,2,\ldots,k\}, i \in \{1,2,\ldots,N\},$$

$$\mathrm{C2}: \sum_{i=1}^{N} i(k) = 1, \forall k,$$

$$\mathrm{C3}: 0 \le \alpha_i(k) \le 1, \forall i, k,$$

$$\mathrm{C4}: \sum_{k=1}^{k} \left(E_{MEC,i(k)} + E_{UAV,(k)} \right) \le E, \forall i,$$

$$\mathrm{C5}: \sum_{k=1}^{k} \sum_{i=1}^{N} i(k) I_i(k) = I,$$

$$\mathrm{C6}: kT = T_{\text{total}},$$

Where C1 and C2 denote that there is one and only one ground device selected to be associated with the UAV in the kth time slot, 1 if selected, and 0 otherwise; C3 is the task offloading ratio limitation interval; C4 denotes the total energy consumption

limitation; and C5 and C6 denote that the total computational tasks need to be processed within the total time.

3. Algorithm Design

3.1. Definition of the Problem

DDPG (Deep Deterministic Policy Gradient) is a deep deterministic policy gradient algorithm that combines the policy gradient method and Q-learning method for solving reinforcement learning problems in continuous action space, which is mainly used to solve reinforcement learning problems in continuous action space, such as robot control, continuous motion control etc. Unlike discrete action space, there are infinite number of action choices in continuous action space, DDPG algorithm overcomes the problem under continuous action space by introducing deterministic policy and value function based approach.

In this section, the problem of 2.3 is described as a Markov Decision Process (MDP) and solved using the GPRE-DDPG algorithm, which assigns mission offloading ratios to the UAV's flight state in continuous space and dynamically adjusts the ground equipment scheduling.

According to the properties of Markov decision making, the intelligent body selects an action to interact with the environment based on the current input state and obtains a reward as feedback used to adjust the next action. Markov decision making defines $\{S_i, A_i, R_i, S_{i+1}\}$, where $S_i$is the set of current possible states, $A_i$is the action selected by the intelligent body, $R_i$is the reward received after interacting with the environment, and $S_{i+1}$is the next moment state [16].

To solve the optimization problem presented in Section 2.3, the state space, action space, and reward function of the system are represented as follows:

(1) State space: Contains the set of all possible states of the system. In this paper, the model state space is defined as

$$S_k = \left\{ \mathrm{z(k)}, \mathrm{w}_i\mathrm{(k)}, I_i(k), h_{si}(k), E_{\mathrm{res}}(k), I_{\mathrm{res}}(k) \right\} \tag{10}$$

where $w_i(k), I_i(k), h_{si}(k)$denotes the 3D coordinates of the *i*th ground device in time slot k, the total number of tasks generated, and the channel masking index, respectively, and $E_{res}(k)$is the amount of power remaining, and $I_{res}(k)$ is the total number of tasks remaining.

In order to ensure that the influence of each state feature on the model training is relatively balanced, so that different features have similar value ranges, and to avoid slowing down the training speed due to the large differences in the feature values during the gradient update, a linear feature scaling mechanism is introduced, i.e., the original state value is divided by the difference between the maximum value and the minimum value of the corresponding feature, in order to scale the state value to within the range of [0, 1]. Where the difference between the maximum and minimum values of a feature is defined as the scaling factor φ.

(2) Action space: Contains the set of all actions that can be taken by an intelligent body. The central controller decides the actions that the intelligent body should take based on S_k. The action space is defined as

$$A_k = \{\alpha_i(k), i(k), v_k, \theta_k\} \tag{11}$$

Where v_k, θ_k together determine the flight state of the UAV, and the equation for the movement of the UAV is as follows

$$z(k+1) = z(k) + \left[v_k t_{fly} cos\theta_k, v_k t_{fly} sin\theta_k, 0 \right] \tag{12}$$

Where t_{fly} is the flight time.

Since the DDPG algorithm is a continuous action space algorithm, its output is usually a continuous value domain, but denotes the serial number of the ground device that is selected to communicate with the UAV, it is necessary to quantize the action output by the intelligent body. In this paper, a threshold quantization method is designed, and the quantization thresholds are respectively $\frac{1}{N}$、$\frac{2}{N}$、…、$\frac{N-1}{N}$.For example, when N is 4, the thresholds are 0.25, 0.5, and 0.75, which are quantized as 0 when $0 \le i(k) \le 0.25$, i.e., the 1st ground device is selected, and 1 when $0.25 \le i(k) \le 0.5$, and so on.

(3) Reward function: The reward or punishment obtained by an intelligent body after taking a certain action in a specific state. In this paper, considering that the optimization objective is to minimize the maximum time delay of the system, it is designed that when the time delay is larger, the reward is smaller; the smaller the time delay is, the larger the reward is. Therefore, the reward function is defined as

$$R_k = -t_d(k) = -\sum_{i=1}^{N} i(k) max\left\{ t_{loc,i(k)}, t_{tra,i(k)} + t_{MEC,i(k)} \right\} \tag{13}$$

The GPER-DDPG algorithm adds the priority experience replay (PER) to the DDPG algorithm, the traditional experience replay mechanism usually samples from the experience pool using a uniform distribution, but in practice, this way is not efficient, for neural networks, the priority of the experience samples has some differences, so the higher priority experience samples are given larger sampling weights to increase the probability of being sampled. The priority g of the empirical samples is denoted as:

$$g = |TD| + \varpi \tag{14}$$

where $|TD|$ is the absolute error value of the most recent sample and ϖ is a non-zero bias, if ϖ is 0, it is uniform sampling. The sampling probability and learning rate are defined as respectively:

$$p_i = \frac{p_i^g}{\sum_u p_u^g} \tag{15}$$

$$w_i = w \cdot (o \cdot p_i)^{-\beta} \tag{16}$$

where the hyperparameter β is the importance increment, increasing in the interval [0,1], and O is the total number of samples in the empirical pool.

In addition, the GPER-DDPG algorithm introduces adaptive action strategy noise. In this paper, an adaptive mechanism is introduced to dynamically adjust the strength of the action strategy noise so that it is more exploratory in the early stage of training and more focused on utilizing the learned strategies in the later stage of training. The adaptive process uses a linear descent method for adjusting the variance of the Gaussian noise, and the adaptive variance adjustment process is as follows

$$\rho = \rho_{initial} - decay\_rate \times Episode \tag{17}$$

where $Episode$ is the training step, the attenuation factor $decay\_rate$ is set to 0.0002, and $\rho_{initial}$ is the initial value of the Gaussian noise variance, which is set to 0.12. In the training process of the target network, the delayed updating method is used to update the target network by delaying the target network by a certain number of training steps in the middle and late stages of the training process, so as to reduce the frequency of the update of the target network, to reduce the oscillations during the training process, and to improve the algorithm's Stability. The interval training step is denoted by C.

3.2. Implementation of GPER-DDPG

The GPER-DDPG algorithm implementation process is mainly divided into three processes: sampling, training, and parameter update:

(1) Sampling: Randomly select the state as S_1, feature scale the state of the selection and input it into the Actor online network, Actor online network selects the action according to the state and adaptive Gaussian noise, input the action into the environment to get the reward and the next state, feature scale the next state, generate the quintet to be stored into the prioritized experience playback pool and assign the priority, and the number of data in the experience playback pool reaches the threshold value Start training, otherwise continue sampling.

(2) Training: L data from the empirical playback pool are taken out and fed to Critic's online network and target network to get the action values and calculate the loss separately.

(3) Parameter update: A delayed update is used, where the soft update of the target network is performed only after half of the training process, by delaying the training step C.

Algorithms GPER-DDPG

1:Initialize Actor and Critic online network parameters θ^{μ} and θ^{Q}, copy online network parameters to target network $(\theta^{\mu'} \leftarrow \theta^{\mu}, \theta^{Q'} \leftarrow \theta^{Q})$
2:Initialize replay buffer D and priority buffer P with a capacity of n
3:**for** episode = 1 to M **do**
4:Get the initial state of the observed state S_1
5: **for** k = 1 to k **do**
6: State feature scaling for state S_k
7: Use actor network θ^{μ} to select actions based on the current state with some Gaussian exploration noise $A_k = clip(\mu_\theta(s) + \varepsilon,\ A_{low},\ A_{high})$,Noise variance adaptive adjustment $\rho = \rho_{initial} - decay\_rate \times Episode$
8: Execute an action A_k, Get rewarded R_k and next state S_{k+1}
9: State feature scaling for state S_{k+1}
10: Store the normalized $(S_k,\ A_k, R_k, S_{k+1})$ in the replay buffer D and assign a priority g
11: Sample a conversion mini-batch from the priority buffer P as L
12: Calculation of $y_i = R_i + \gamma Q'(S_{i+1}, \mu'(S_{i+1}|\theta^{\mu'})|\theta^{Q'})$
13: Calculate the *TD* error and update the priority in the priority buffer P
14: Minimize the loss function of the cubic network update the parameters of the cubic network $L = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i \mid \theta^{Q})\right)^2$
15: **if** k mod C and episode $\geq \frac{episode_{MAX}}{2}$ **then**
16: Updating the parameters of the actor network using the sampled policy gradient $\nabla_{\theta^{\mu}} J \approx \frac{1}{I}\sum_i \nabla_a Q(s_i, a_i \mid \theta^{Q}) \nabla_{\theta^{\mu}} \mu(s_i \mid \theta^{\mu})$
17: Update the parameters of the target network using the soft update method $\theta^{Q'} \leftarrow \omega\theta^{Q} + (1-\omega)\theta^{Q'}, \theta^{\mu'} \leftarrow \omega\theta^{\mu} + (1-\omega)\theta^{\mu'}$

18: end if
19: end for
20:end for

4. Simulation and Result Analysis

In order to verify the effectiveness and feasibility of the algorithm proposed in this paper, a simulation environment is constructed through Python language based on the setup parameters proposed in this chapter. The feasibility of the proposed algorithm is illustrated based on the effect of different parameters on the proposed algorithm, and the effectiveness of the algorithm in this paper is illustrated by comparing it with other existing algorithms.

4.1. Parameter Settings

In our experiments, we consider a 100 × 100 rectangular area where the server CPU carried by the ground devices distributed in the area have a computing frequency of 0.2 GHz, the complexity of the computational task is set to 1000 cycles/bit, with a total time T_{total} of 400 and k of 40[17-19]. The CPU computing frequency of the MEC server of the UAV is 1.12 GHz , the maximum flight speed of the UAV is 50 m/s , h_o is -50 dB,B is 1 MHz, the channel noise interference σ^2 is -95 dBm, the transmission power $p_i$is set to 0.1 W, and the total amount of batteries at the initial time is 500000 J, the specific parameters are shown in Table 1.

Table 1. Parameter setting table

System parameters	Value
N	4
f_{LD}	0.2GHz
S	1000 cycles/bit
T_{total}	320
f_{MEC}	1.12GHz
h_o	-50dB
B	1MHz
σ^2	-95dBm
p_i	0.1W
E	500000J
k	40
H	100m
m_{UAV}	9400g
$decay\_rate$	0.0002
$\rho_{initial}$	0.12
C	2
β	4700
t_{fly}	1
χ	10^{-27}

4.2. Analysis of Results

Figure 2 depicts the convergence performance of the reward function of the GPER-DDPG algorithm, from which it can be found that the value of the reward function rises sharply in Episode for 300 times, and then basically reaches a stable value within 600 times. This is mainly because the actions performed by the intelligent agent in the early stage of training have a significant impact on the reward value, so the reward function will rise sharply. Finally, as the number of training times increases, the neural network parameters learned by the intelligent body gradually converge to the optimal value, and the reward value then converges to the optimal value.

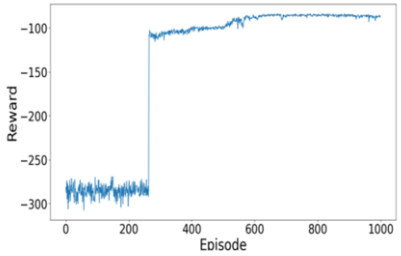
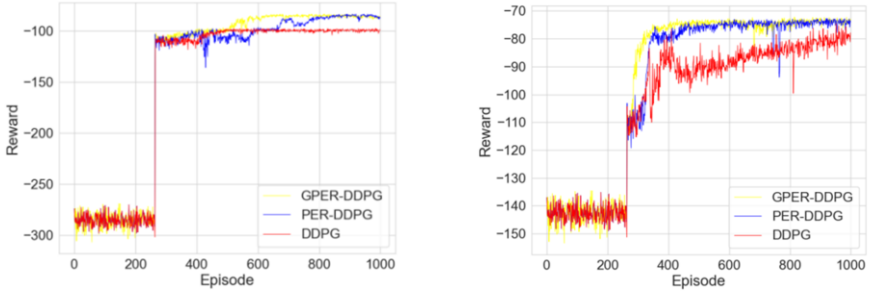


Figure 2. GPER-DDPG reward value curve.

Figures 3(a) and (b) show the comparison of the convergence performance of the three algorithms.3(a) and (b) show the comparison of the convergence performance of each algorithm when f_{LD} is 0.2 GHz and 0.4 GHz, respectively.

From Figure 3(a), it can be found that at the beginning of training, the reward values of the three algorithms are basically the same; however, at the late stage of training, the optimal reward value of the ordinary DDPG algorithm gradually converges to a suboptimal solution; in the comparison between the PER-DDPG and GPER-DDPG algorithms, it can be found that, although the two algorithms converge to a similar result, the convergence speed of the GPER-DDPG algorithm is significantly better than that of the former. This is mainly due to the fact that the GPER-DDPG algorithm introduces adaptive action strategy noise, which gradually reduces the action noise during the training process and smoothes the convergence process. The convergence performance of the improved GPER-DDPG algorithm is 12% better than the normal DDPG algorithm. In Figure 3(b), it can be found that the performance of GPER-DDPG algorithm and PER-DDPG does not have any obvious change, and it can still converge to the optimal solution, but the convergence speed of DDPG algorithm decreases obviously, and the fluctuation of the convergence process is large. The convergence performance of the improved GPER-DDPG algorithm is about 10% higher than that of the ordinary DDPG algorithm, and the convergence performance is significantly improved. By comparing Figure 3 (a) and (b), it can be found that the maximum rewards of the three algorithms are higher than the maximum rewards when f_{LD} is 0.4 than when f_{LD} is 0.2.



(a). f_{LD}=0.2GHz **(b). f_{LD}**=0.4GHz

Figure 3. Algorithm convergence performance comparison.

Figure 4 depicts the effect of the number of ground users on the total system delay under the five algorithms, from which it can be found that the delay obtained from the processing of each algorithm does not change significantly as the number of users increases. The improved GPER-DDPG algorithm still obtains the lowest delay, proving that it can obtain the optimal control decision in the case of user changes.

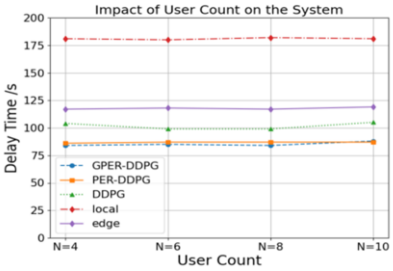


Figure 4. Impact of the number of users on the total delay.

Combining the above analysis, the GPER-DDPG algorithm obtains lower maximum system delay than both PER-DDPG and DDPG algorithms for different f_{LD} and number of users, and the convergence speed is significantly improved.

5. Conclusion

In this paper, a UAV-assisted edge computing network model is proposed by combining the two technologies of edge computing and UAV, and the GPER-DDPG algorithm is used to optimize for the maximum processing delay of the system. The algorithm can learn user scheduling, UAV's flight state, and task offloading ratio from experience to optimize the maximum processing delay of the system as the optimization goal. By comparing several existing baseline algorithms, the experimental results show that the performance of the improved algorithm proposed in this paper improves the performance of the DDPG algorithm by about 10~12% compared to the DDPG algorithm. Future research will consider the new variables of the system with the energy model improvement, synthesize the computational cost, and apply it to new smart connected scenarios.

References

[1] Zhang H S ,Wang S J ,Zhang W S , et al. Snake optimizer with oscillating factors to solve edge computing task unloading and scheduling optimization problem. Alexandria Engineering Journal, 2024, 91273-304.

[2] V. De Nitto Personè and V. Grassi, "Architectural Issues for Self-Adaptive Service Migration Management in Mobile Edge Computing Scenarios," 2019 IEEE International Conference on Edge Computing (EDGE), Milan, Italy, 2019, pp. 27-29, doi: 10.1109/EDGE.2019.00020.

[3] Sun, Z.; Chen, G. Enhancing Data Freshness in Air-Ground Collaborative Heterogeneous Networks through Contract Theory and Generative Diffusion-Based Mobile Edge Computing. Sensors 2024, 24, 74. https://doi.org/10.3390/s24010074

[4] Chai, S., Guo, L. Edge Computing with Fog-cloud for Heart Data Processing using Particle Swarm Optimized Deep Learning Technique. J Grid Computing 22, 3 (2024). https://doi.org/10.1007/s10723-023-09706-6

[5] Xia S C, Yao Z X, Xian Y J, et al. Distributed heterogeneous task offloading algorithm in mobile edge computing. Journal of Electronics and Information, 2020, 42 (12): 2891-2898.

[6] Wang Z, Wang Q M, Li T et al. A joint optimization method for SWIPT edge networks based on deep reinforcement learning. Computer Applications, 2023, 43(11):3540-3550.

[7] Guo X D, Hao S D, Wang L F. Deep reinforcement learning-based task offloading method for vehicular edge computing. Computer Application Research, 2023, 40(09):2803-2807+2814.DOI:10.19734/j.issn.1001-3695.2023.02.0027.

[8] Peng S, Zhao J B, Wei Agile et al. Task offloading optimization based on mobile edge computing. Computer System Applications, 2023, 32(04):262-267.DOI:10.15888/j.cnki.csa.009013.

[9] D. Van Huynh, Y. Li, A. Masaracchia, T. Hoang and T. Q. Duong, "Optimal Resource Allocation for 6G UAV-enabled Mobile Edge Computing with Mission-Critical Applications," 2023 IEEE International Conference on Metaverse Computing, Networking and Applications (MetaCom), Kyoto, Japan, 2023, pp. 720-723, doi: 10.1109/MetaCom57706.2023.00135.

[10] Zheng H Q, Zhang J S, Chen X et al. Deployment optimization and computational offloading of air-sky-ground integrated mobile edge computing system. Computer Science, 2023, 50(02):69-79.

[11] Bultitude, R. Measurement, characterization and modeling of indoor 800/900 MHz radio channels for digital communications. Communications Magazine, IEEE, 1987, 25(6):5-12.DOI:10.1109/MCOM.1987.1093629.

[12] Howard S J, Pahlavan K. Doppler spread measurements of indoor radio channel. Electronics Letters, 1990, 26(2):107-109.DOI:10.1049/el:19900074

[13] Herbert S, Wassell I, Loh T H, et al. Characterizing the Spectral Properties and Time Variation of the In-Vehicle Wireless Communication Channel. IEEE Transactions on Communications, 2014, 62(7):2390-2399.DOI:10.1109/TCOMM.2014.2328635.

[14] Xiong J, Guo H, Liu J .Task Offloading in UAV-Aided Edge Computing: Bit Allocation and Trajectory Optimization. IEEE Communications Letters,2019:538-541.DOI:10.1109/LCOMM.2019.2891662.

[15] Hu, Q., Cai, Y., Yu, G., Qin, Z., Zhao, M., & Li, G. Y. (2019). Joint offloading and trajectory design for UAV-enabled mobile edge computing systems. IEEE Internet of Things Journal, 6(2), 879–1892

[16] Xingjian L, Jun F, Wen C, et al. Intelligent Power Control for Spectrum Sharing in Cognitive Radios: A Deep Reinforcement Learning Approach. IEEE Access, 2018, 6:25463-25473.DOI:10.1109/ACCESS.2018.2831240.

[17] Wang Y, Fang W, Ding Y, et al. Computation offloading optimization for UAV-assisted mobile edge computing: a deep deterministic policy gradient approach. Wireless Networks, 2021:1-16.DOI:10.1007/s11276-021-02632-z.

[18] Cheng N, Lyu F, Quan W, et al. Space/Aerial-Assisted Computing Offloading for IoT Applications: A Learning-Based Approach. IEEE Journal on Selected Areas in Communications, 2019, PP(5):1-1.DOI:10.1109/JSAC.2019.2906789.

[19] Coldrey M, Berg J E, Manholm L, et al. Non-line-of-sight small cell backhauling using microwave technology. IEEE Communications Magazine, 51[2024-04-08].DOI:10.1109/MCOM.2013.6588654.