Wireless Networks (2021) 27:2991–3006
https://doi.org/10.1007/s11276-021-02632-z

ORIGINAL PAPER

Computation offloading optimization for UAV-assisted mobile edge computing: a deep deterministic policy gradient approach

Yunpeng Wang[1] · Weiwei Fang[1,2] · Yi Ding[3] · Naixue Xiong[4]

Published online: 5 May 2021

Abstract
Unmanned Aerial Vehicle (UAV) can play an important role in wireless systems as it can be deployed flexibly to help improve coverage and quality of communication. In this paper, we consider a UAV-assisted Mobile Edge Computing (MEC) system, in which a UAV equipped with computing resources can provide offloading services to nearby user equipments (UEs). The UE offloads a portion of the computing tasks to the UAV, while the remaining tasks are locally executed at this UE. Subject to constraints on discrete variables and energy consumption, we aim to minimize the maximum processing delay by jointly optimizing user scheduling, task offloading ratio, UAV flight angle and flight speed. Considering the non-convexity of this problem, the high-dimensional state space and the continuous action space, we propose a computation offloading algorithm based on Deep Deterministic Policy Gradient (DDPG) in Reinforcement Learning (RL). With this algorithm, we can obtain the optimal computation offloading policy in an uncontrollable dynamic environment. Extensive experiments have been conducted, and the results show that the proposed DDPG-based algorithm can quickly converge to the optimum. Meanwhile, our algorithm can achieve a significant improvement in processing delay as compared with baseline algorithms, e.g., Deep Q Network (DQN).

Keywords Mobile edge computing · Unmanned aerial vehicle · Deep deterministic policy gradient · Computation offloading

1 Introduction

1.1 Motivation

With the development of 5G technology, computation-intensive applications running on the user equipments (UEs), e.g., online gaming, VR/AR and telemedicine, will become more prosperous and popular. These mobile applications typically require substantial computing resources and incur high energy consumption. Nevertheless, current UEs generally have constrained computing resources and limited battery capacity. Mobile Cloud Computing (MCC) has emerged to augment the computing and storage capabilities of UE [1], and reduce the energy consumption of UE by offloading computation to the cloud via mobile networks. However, cloud servers are often spatially distant from the UE, which may incur high transmission delay [2] and adversely affect user experiences. To reduce the backhaul link delay, mobile computing has recently shifted to a new computation paradigm, i.e., Mobile Edge Computing

✉ Weiwei Fang
wwfang@bjtu.edu.cn

Yunpeng Wang
18120421@bjtu.edu.cn

Yi Ding
dingyi@bwu.edu.cn

Naixue Xiong
xiongnaixue@gmail.com

[1] School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China

[2] Key Laboratory of Industrial Internet of Things & Networked Control, Ministry of Education, Chongqing 400065, China

[3] School of Information, Beijing Wuzi University, Beijing 101149, China

[4] College of Intelligence and Computing, Tianjin University, Tianjin 300350, China

(MEC) [3]. MEC can migrate cloud computing resources and services at a closer proximity to the UE, so as to effectively reduce communication delay and energy consumption [4].

Early studies on MEC mainly focused on enhancing the performance of a MEC system in which computation services are provided by a base station at a fixed position [5–7]. For example, Tran et al. [5] proposed a convex optimization method to optimize resource allocation and a low-complexity heuristic algorithm for task offloading in a multi-user multi-server MEC system. Zhao et al. [6] aimed at maximizing system utility with a Collaborative Computation Offloading and Resource Allocation Optimization (CCORAO) scheme in a cloud-assisted MEC system. To reduce the computation cost in a multi-user MIMO based MEC system, Chen et al.[7] proposed a deep reinforcement learning method to dynamically generate a continuous power allocation strategy. However, the MEC services provided by fixed infrastructures cannot effectively work in the scenarios where communication facilities are sparsely distributed or when sudden natural disasters occur.

Most recently, researchers have paid special attention to Unmanned Aerial Vehicle (UAV) because of its high mobility and flexible deployment [8]. They studied resource allocation or path planning in UAV-assisted MEC systems [9–14]. Aiming at reducing processing delay among UEs in the UAV-assisted MEC system, Hu et al. [9] developed an algorithm based on the penalty dual decomposition optimization framework, by jointly optimizing UAV trajectory, computation offloading and user scheduling. Diao et al. [10] designed a joint UAV trajectory and task data allocation optimization algorithm for energy saving. Cheng et al. [11] proposed a new edge computing architecture called UAV-assisted space-air-ground integrated network (SAGIN), and designed an actor-critic based Reinforcement Learning (RL) algorithm for resource allocation and task scheduling. Considering the dimensional curse of state space, Li et al. [12] adopted a RL algorithm to improve the throughput of UEs' task migrations in a UAV-based MEC system. Xiong et al. [13] presented an optimization algorithm to reduce energy consumption by jointly optimizing offloading decision, bit allocation and UAV trajectory. Selim et al. [14] proposed to use Device-to-Device (D2D) as an additional option for auxiliary communication and computation offloading in UAV-MEC system. Despite extensive studies and applications, there still remain a lot of challenges in UAV-assisted MEC systems, e.g., the system performance is severely affected by the limitation on UE's computation capability and the blockages [15] of environmental obstacles (e.g., trees or buildings), especially in urban areas. Thus, adaptive link selection and task offloading issues are very important in the UAV-assisted MEC system.

1.2 Novelty and contribution

In this paper, we consider a MEC system consisting of a UAV mounted with a nano server and a number of UEs, where the communication conditions are dynamic and time-varying. Unlike the Deep Q Network (DQN) based algorithms proposed for discrete action spaces [12], this paper designs a new Deep Deterministic Policy Gradient (DDPG) based computation offloading algorithm, which can effectively support a continuous action space of task offloading and UAV mobility. The main contributions of this paper can be summarized as follows.

- Considering time-varying channel status in the time-slotted UAV-assisted MEC system, we jointly optimize user scheduling, UAV mobility and resource allocation, and formulate the non-convex computation offloading problem as a Markov Decision Process (MDP) problem to minimize the processing delay.
- The complexity of the system states is very high when we consider the MDP models. Besides, the decisions of computation offloading needs the support to the continuous action space. Thus, we propose a novel DDPG-based computation offloading approach. DDPG is an advanced reinforcement learning algorithm, which uses an actor network to generate unique action and a critic network to approximate Q-value action function [16]. In this paper, DDPG algorithm is adopted to obtain the optimal policy for user scheduling, UAV mobility and resource allocation in our UAV-assisted MEC system.
- The proposed DDPG algorithm is implemented on the TensorFlow platform. Simulation results with different system parameters demonstrate the effectiveness of this algorithm. Under different communication conditions, our algorithm can always achieve the best performance, as compared with other baseline algorithms.

The rest of this paper are organized as follows. Section 2 presents system models for our UAV-assisted MEC system, as well as the optimization problem. Section 3 briefly introduces the preliminaries on deep reinforcement learning. Section 4 proposes our DDPG-based computation offloading algorithm. Section 5 illustrates simulation results of our proposed algorithm, and compares it with other baseline algorithms. Finally, Sect. 6 concludes this work.

2 System model

We consider a UAV-assisted MEC system, which consists of K UEs and a UAV equipped with a nano MEC server. The whole system operates in discrete time with equal-

length time slots [17, 18]. The UAV provides communication and computing services to all UEs, but only to one UE at a time [13]. Due to the constrained computing capability, the UE has to offload a portion of the computing tasks via wireless network to the MEC server at the UAV.

2.1 Communication model

The UAV provides computing services to all UEs in a time-division manner [13], in which the whole communication period T is divided into I time slots. We assume that the UEs move randomly in the area at a low speed. In each time slot, the UAV hovers in a fixed position, and then establishes communication with one of the UEs. After offloading a portion of its computing tasks to the server, the UE executes the remaining tasks locally. In the 3D Cartesian coordinate system [10], the UAV keeps flying at a fixed altitude H, where the UAV has a start coordinate $\mathbf{q}(i) = [x(i), y(i)]^T \in \mathbf{R}^{2\times 1}$ and an end coordinate $\mathbf{q}(i+1) = [x(i+1), y(i+1)]^T \in \mathbf{R}^{2\times 1}$ at time slot $i \in \{1, 2, \ldots, I\}$. The coordinate of UE $k \in \{1, 2, \ldots, K\}$ is $\mathbf{p}_k(i) = [x_k(i), y_k(i)]^T \in \mathbf{R}^{2\times 1}$. The channel gain of the line-of-sight link between the UAV and the UE k can be expressed as [13]:

$$g_k(i) = \alpha_0 d_k^{-2}(i) = \frac{\alpha_0}{||\mathbf{q}(i+1) - \mathbf{p}_k(i)||^2 + H^2}, \tag{1}$$

where α_0 denotes the channel gain at a reference distance $d = 1$ m, and $d_k(i)$ denotes the Euclidean distance between the UAV and k. Due to blockage from obstacles, the wireless transmission rate can be given as:

$$r_k(i) = B\log_2(1 + \frac{P_{up} g_k(i)}{\sigma^2 + f_k(i) P_{NLOS}}), \tag{2}$$

where B denotes the communication bandwidth, P_{up} denotes the transmit power of the UE in the upload link, σ^2 denotes the noise power, P_{NLOS} denotes the transmission loss [19], and $f_k(i)$ denotes the indicator of whether there is any block between UAV and UE k at time slot i (i.e., 0 means no blockage and 1 means blockage) [15].

2.2 Computation model

In our MEC system, a partial offloading strategy is used for the UE's tasks in each time slot [20]. Let $R_k(i) \in [0, 1]$ denotes the ratio of tasks offloaded to the server, and $(1 - R_k(i))$ denotes the remaining tasks that are locally executed at the UE. Then, the local execution delay of UE k at time slot i can be given as:

$$t_{local,k}(i) = \frac{(1 - R_k(i)) D_k(i) s}{f_{UE}} \tag{3}$$

where $D_k(i)$ denotes the computing task sizes of UE k, s indicates the CPU cycles required to process each unit byte, and f_{UE} denotes the UE's computing capability. At i-th time slot, UAV flies from position $\mathbf{q}(i)$ to the new hovering position

$$\mathbf{q}(i+1) = [x(i) + v(i) t_{fly} \cos\beta(i),\ y(i) + v(i) t_{fly} \sin\beta(i)]^T$$

at a speed $v(i) \in [0, v_{\max}]$ and an angle $\beta(i) \in [0, 2\pi]$. The energy consumed by this flight can be expressed as:

$$E_{fly}(i) = \phi ||v(i)||^2 \tag{4}$$

where $\phi = 0.5 M_{UAV} t_{fly}$[9], M is relevant to the UAV's payload, t_{fly} is fixed flight time. In the MEC system, the size of computation results provided by the server is usually very small and negligible [9]. Thus, the transmitting delay for the downlink is not taken into consideration here. The processing delay of MEC server can be divided into two parts. One part is the transmission delay, which can be given as:

$$t_{tr,k}(i) = \frac{R_k(i) D_k(i)}{r_k(i)}. \tag{5}$$

The other part is the delay resulting from the computation at the MEC server, which can be given as:

$$t_{UAV,k}(i) = \frac{R_k(i) D_k(i) s}{f_{UAV}} \tag{6}$$

where f_{UAV} denotes the computation frequency of server CPU. Correspondingly, the energy consumed to offload the computing tasks to the server in time slot i can also be divided into two parts, including the one for transmission and the other one for computation. When the computation is performed at the MEC server, its power consumption can be expressed as:

$$P_{UAV,k}(i) = \kappa f_{UAV}^3. \tag{7}$$

Then, the energy consumption of the MEC server at time slot i is given as:

$$E_{UAV,k}(i) = P_{UAV,k}(i) t_{UAV,k}(i) = \kappa f_{UAV}^2 R_k(i) D_k(i) s. \tag{8}$$

2.3 Problem formulation

Based on the models introduced above, we formulate the optimization problem in our UAV-assisted MEC system. To ensure efficient utilization of the limited computation resources, we aim at minimizing the maximum processing delay for all UEs, by jointly optimizing user scheduling, UAV mobility and resource allocation in the system. Specifically, the optimization problem can be denoted as:

$$\min_{\{\alpha_k(i),\mathbf{q}(i+1),R_k(i)\}} \sum_{i=1}^{I}\sum_{k=1}^{K}\alpha_k(i)\max\{t_{local,k}(i),t_{UAV,k}(i)+t_{tr,k}(i)\} \tag{9a}$$

$$\text{s.t.}\alpha_k(i)\in\{0,\,1\},\ \forall i\in\{1,\,2,\,\ldots,\,I\},\ k\in\{1,\,2,\,...,\,K\}, \tag{9b}$$

$$\sum_{k=1}^{K}\alpha_k(i)=1,\ \forall i, \tag{9c}$$

$$0\le R_k(i)\le 1,\ \forall i,\ k, \tag{9d}$$

$$\mathbf{q}(i)\in\{(x(i),\,y(i))|x(i)\in[0,\,L],\,y\in[0,\,W]\},\ \forall i, \tag{9e}$$

$$\mathbf{p}(i)\in\{(x_k(i),\,y_k(i))|x_k(i)\in[0,\,L],\,y_k\in[0,\,W]\},\ \forall i,\ k, \tag{9f}$$

$$f_k(i)\in\{0,\,1\},\ \forall i,\ k, \tag{9g}$$

$$\sum_{i=1}^{I}(E_{fly,k}(i)+E_{UAV,k}(i))\le E_b,\ \forall k, \tag{9h}$$

$$\sum_{i=1}^{I}\sum_{k=1}^{K}\alpha_k(i)D_k(i)=D, \tag{9i}$$

where Constraint (9b) and constraint (9c) guarantee that only one user is scheduled for computing in time slot i. Constraint (9d) denotes the range of values for the offloading ratio of the computing task. Constraints (9e) and (9f) show that UE and UAV can only move in the given area. Constraint (9g) represents the blockage in the wireless channel between the UAV and the UE at time slot i. Constraint (9h) ensures that the energy consumption of UAV flight and calculation in all time slots does not exceed the maximum battery capacity. Constraint (9i) specifies all of computing tasks to be completed in the whole time period.

3 DDPG based computation offloading optimization

In this section, we first introduce fundamental knowledge of MDP, Q-Learning, DQN, and DDPG which are important emerging RL technologies. Then, we discuss how to utilize DDPG to train an efficient computation offloading policy in our UAV-assisted MEC system. In detail, we define the state space, the action space and the reward function, describe the state normalization for data preprocessing, and illustrate the procedures of training algorithm and testing algorithm.

3.1 A primer on RL

3.1.1 MDP

MDP [21] is a mathematical framework to describe the discrete time stochastic control process in which outcomes are partly random and under control of an agent or a decision maker. It formally describes an environment which is fully observable for reinforcement learning. Typically, a MDP can be defined as a tuple $(\mathcal{S},\,\mathcal{A},\,p(.,\,.),\,r)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space,$p(s_{i+1}|s_i,\,a_i)$ is the transition probability from the current state $s_i\in\mathcal{S}$ to the next $s_{i+1}\in\mathcal{S}$ after action $a_i\in\mathcal{A}$ is executed, and $r:\mathcal{S}\times\mathcal{A}\to\mathcal{R}$ is instantaneous/immediate reward function. We denote $\pi{:}\mathcal{S}\to\mathcal{P}(\mathcal{A})$ as a "policy" which is a mapping from a state to an action. MDP aims to find an optimal policy which can maximizes the expected accumulated rewards:

$$R_i=\sum_{l=i}^{\infty}\gamma^{l-i}r_l, \tag{10}$$

where $\gamma\in[0,\,1]$ is the discount factor, $r_l=r(s_l,\,a_l)$ is the immediate reward at l-th time slot. Under policy π, the expected discounted return from state s_i is defined as a state value function, i.e.,

$$V_\pi(s_i)=\mathbb{E}_\pi[R_i|s_i]. \tag{11}$$

Similarly, the expected discounted return after taking action a_i in state s_i under a policy π is defined as an action value function, i.e.,

$$Q_\pi(s_i)=\mathbb{E}_\pi[R_i|s_i,\,a_i]. \tag{12}$$

Based on the Bellman equation, the recursive relationship of the state value function and the action value function can be expressed as, respectively,

$$V_\pi(s_i)=\mathbb{E}_\pi[r(s_i,\,a_i)+\gamma V_\pi(s_{i+1})], \tag{13}$$

$$Q_\pi(s_i,\,a_i)=\mathbb{E}_\pi[r(s_i,\,a_i)+\gamma Q_\pi(s_{i+1},\,a_{i+1})]. \tag{14}$$

Since we aim to find the optimal policy $\pi*$, the optimal action in each state can be found by the optimal value function. The optimal state value function can be expressed as:

$$V_*(s_i)=\max_{a_i\in\mathcal{A}}\mathbb{E}_\pi[r(s_i,\,a_i)+\gamma V_\pi(s_{i+1})]. \tag{15}$$

The optimal action value function also follows the optimal policy $\pi*$, we can write Q_* in terms of V_* as follows:

$$Q_*(s_i,\,a_i)=\mathbb{E}_\pi[r(s_i,\,a_i)+\gamma V_*(s_{i+1})]. \tag{16}$$

3.1.2 Q-learning

RL is an important branch of machine learning, in which an agent tries to get the maximum return by interacting with a control environment to its optimal state. Although RL is often used to solve the optimization problems of MDPs, the underlying transmission probability $p(s_{i+1}|s_i, a_i)$ is unknown or even non-stationary. In RL, an agent tries to get the maximum return by interacting with a control environment and adjusting its action through previous experience. Q-learning [21] is a popular and effective method in RL, and it is an off-policy Temporal Difference (TD) control algorithm. The Bellman optimality equation for the state-action function, i.e., the optimal Q function, can be expressed as:

$$Q_*(s_i, a_i) = \mathbb{E}_\pi[r(s_i, a_i) + \gamma \max_{a_{i+1}} Q_*(s_{i+1}, a_{i+1})]. \tag{17}$$

The optimal value of the Q function can be found by an iterative process. The agent learns from experience tuple (s_i, a_i, r_i, s_{i+1}) and the Q function can be updated at time step i as follows:

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha[r(s_i, a_i) + \gamma \max_{a_{i+1}} Q(s_{i+1}, a_{i+1}) - Q(s_i, a_i)], \tag{18}$$

where α is the learning rate, and $r(s_i, a_i) + \gamma \max_{a_{i+1}} Q(s_{i+1}, a_{i+1})$ is the predicted Q value and $Q(s_i, a_i)$ is current Q value. The difference between the predicted Q value and the current Q value is a TD error. When the appropriate learning rate is selected, the Q learning algorithm converges [22].

3.1.3 DQN

The Q-learning algorithm updates the Q value of each item in the state action space by maintaining a look-up table, which is suitable for small state-action space. Considering the complexity of system models in practice, these spaces are usually very large. The reason is that a large number of states are rarely accessed and the corresponding Q values are rarely updated, resulting in a longer convergence time of the Q function. By combining deep neural networks (DNNs) with Q-learning, DQN [23] addresses the shortcomings of Q-learning. The core idea behind DQN is to use a DNN parameterized by θ to get the approximate Q value $Q(s, a)$ instead of Q table, i.e., $Q(s, a|\theta) \approx Q_*(s, a)$.

However, the stability of RL algorithm using DNN cannot be guaranteed, which is stated in [23]. In order to solve this problem, two mechanisms are employed. The first one is experience replay. At each time step i, the agent's interactive experience tuples (s_i, a_i, r_i, s_{i+1}) are stored in the experience replay buffer, i.e., experience pool B_m. Then, a small number of samples, i.e., mini-batches, are randomly selected from the experience pool to train the parameters of the deep neural network, instead of directly using continuous samples for training. The second stabilization method is to use a target network that initially contains the weights of the network that sets the policy, but remains frozen for a fixed number of time steps. The target Q network is updated slowly, but the value of the main Q network is updated frequently. In this way, the correlation between the target and the estimated Q value is significantly reduced, which makes the algorithm stable.

In each iteration, the deep Q function is trained toward the target value by minimizing the loss function $L(\theta)$. The loss function can be written as:

$$L(\theta) = \mathbb{E}[(y - Q(s, a|\theta))^2], \tag{19}$$

where the target value y is expressed as $y = r + \gamma \max_{a'} Q(s', a'|\theta_i^-)$. In the Q-learning, the weights $\theta_i^- = \theta_{i-1}$, whereas in deep Q-learning $\theta_i^- = \theta_{i-X}$, i.e., the target network weights update every X time steps.

3.1.4 DDPG

Although the DQN algorithm can solve the problem of high-dimensional state spaces, it still can't deal with the continuous action spaces. As a model-free off-policy actor-critic algorithm using DNN, DDPG [16] algorithm can learn polices in continuous action spaces. The actor-critic algorithm is composed of a policy function and a Q-value function. The policy function acts an actor to generate actions. The Q-value function acts as a critic to evaluate the actor's performance and direct the follow-up action of the actor.

As shown in Fig. 1, DDPG uses two different DNNs to approximate the actor network $\mu(s|\theta^\mu)$, i.e., the policy function and the critic network $Q(s, a|\theta^Q)$, i.e., the Q-value function, respectively. In addition, both the actor network and the critic network contain a target network with the same structure as them: actor target network μ' with parameter $\theta^{\mu'}$, critic target network Q' with parameter $\theta^{Q'}$. Similar to DQN, critic network $Q(s, a|\theta^Q)$ can be updated as follows:

$$L(\theta^Q) = \mathbb{E}_{\mu'}[(y_i - Q(s_i, a_i|\theta^Q))^2], \tag{20}$$

where

$$y_i = r_i + \gamma Q(s_{i+1}, \mu(s_{i+1})|\theta^Q). \tag{21}$$

As has been proved by Silver et al. [24], the policy gradient can be updated by chain rule,

$$\begin{aligned} \nabla_{\theta^\mu} J &\approx \mathbb{E}_{\mu'}[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i|\theta^\mu)}] \\ &= \mathbb{E}_{\mu'}[\nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i|\theta^\mu)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_i}]. \end{aligned} \tag{22}$$

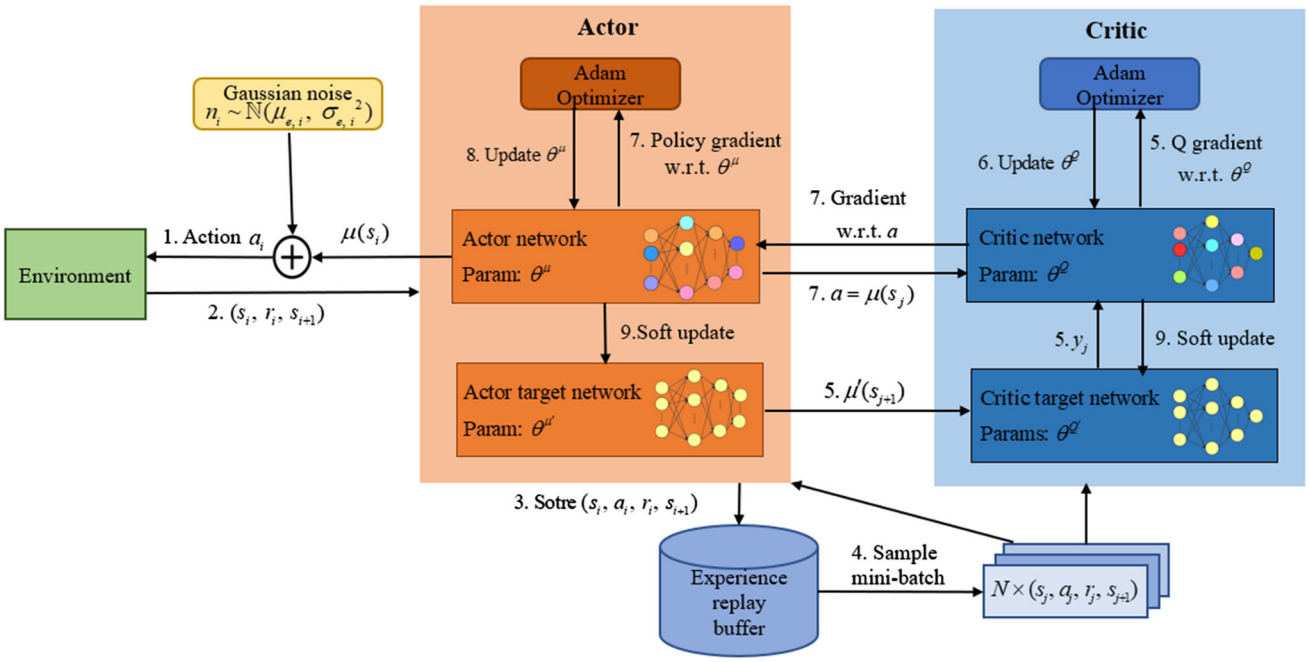

Fig. 1 Diagram of DDPG

The entire training process for the DDPG algorithm can be summarized as follows. Firstly, the actor network μ outputs $\mu(s_i)$ after the previous training step. In order to provide adequate exploration of the state space, we need to balance the tradeoff between exploration and exploitation. Actually, we can treat the exploration of DDPG independently from the learning process, as DDPG is an off-policy algorithm. Therefore, we construct the action space by adding behavior noise n_i to obtain action $a_i = \mu(s_i) + n_i$, where n_i obeys the Gaussian distribution $n_i \sim \mathbb{N}(\mu_e, \sigma_{e,i}^2)$, μ_e is the mean and $\sigma_{e,i}$ is the standard deviation. After performing a_t in the environment, the agent can observe the next state s_{i+1} and the immediate reward r_t. Then the transition (s_i, a_i, r_i, s_{i+1}) is stored in the experience replay buffer. After that, the algorithm randomly selects N transitions (s_j, a_j, r_j, s_{j+1}) in the buffer to make up a mini-batch and inputs it into actor network and critic network. With the mini-batch, the actor target network μ' outputs the action $\mu'(s_{j+1})$ to the critic target network Q'. With mini-batch and $\mu'(s_{j+1})$, the critic network can calculate the target value y_j based on (21).

In order to minimize the loss function, the critic network Q will be updated by a given optimizer, e.g., Adam Optimizer. Afterwards, the actor network μ gives the mini-batch action $a = \mu(s_j)$ to the critic network to achieve the action a's gradient $\nabla_a Q(s, a|\theta^Q)|_{s=s_j, a=\mu(s_j)}$. The parameter $\nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_j}$ can be derived by its own optimizer. Using these two gradients, the actor network can be updated with the following approximation:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_j [\nabla_a Q(s, a|\theta^Q)|_{s=s_j, a=\mu(s_j|\theta^\mu)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_j}]. \tag{23}$$

Finally, the DDPG agent uses a small constant τ to update the critic target network and actor target network softly:

$$\theta^{Q'} \leftarrow \theta^Q + (1-\tau)\theta^{Q'}, \tag{24}$$

$$\theta^{\mu'} \leftarrow \theta^\mu + (1-\tau)\theta^{\mu'}. \tag{25}$$

4 DDPG-based algorithm

4.1 State space

In the UAV-assisted MEC system, the state space is jointly determined by K UEs, a UAV and their environment. The system state at time slot i can be defined as:

$$\begin{aligned} s_i = &(E_{battery}(i), \mathbf{q}(i), \mathbf{p}_1(i), \ldots, \mathbf{p}_K(i), D_{remain}(i), \\ &D_1(i), \ldots, D_K(i), f_1(i), \ldots, f_K(i)), \end{aligned} \tag{26}$$

where $E_{battery}(i)$ denotes the remaining energy of UAV battery at i-th time slot, $\mathbf{q}(i)$ denotes the UAV location information, $\mathbf{p}_k(i)$ denotes the location information of UE k served by the UAV, $D_{remain}(i)$ denotes the size of remaining tasks that the system needs to complete in the whole time period, $D_k(i)$ denotes the task size randomly generated by UE k in the i-th slot, and $f_k(i)$ denotes an indication of

whether the signal of UE k is blocked by obstacles. Especially when $i = 1$, $E_{battery}(i) = E_b$ and $D_{remain}(i) = D$.

4.2 Action space

Based on the current state of the system and the observed environment, the agent selects actions including UE k' to be served, flight angle of UAV, flight speed of UAV and task offloading ratio in time slot i. The action a_i can be denoted as:

$$a_i = (k(i),\ \beta(i),\ v(i), R_k(i)). \tag{27}$$

It is worth noting that the actor network in DDPG outputs continuous actions. The action variable UE $k(i) \in [0, K]$ selected by the agent needs to be discretized, i.e., if $k(i) = 0$, then $k' = 1$; if $k(i) \neq 0$, then $k' = \lceil k(i) \rceil$, where $\lceil \cdot \rceil$ is the ceiling operation. The flight angle of UAV, the flight speed of UAV and the task offloading ratio can be accurately optimized in a continuous action space, i.e., $\beta(i) \in [0, 2\pi]$, $v(i) \in [0, v_{\max}]$, and $R_k(i) \in [0, 1]$. The above four variables are jointly optimized to minimize system cost.

4.3 Reward function

The agent's behavior is reward-based, and the choice of an appropriate reward function plays a vital role in the performance of the DDPG framework. We aim to maximize reward by minimizing the processing delay defined in the problem (9), as follows:

$$r_i = r(s_i,\ a_i) = -\tau_{delay}(i), \tag{28}$$

where the processing delay at time slot i is

$$\tau_{delay}(i) = \sum\nolimits_{k=1}^{K} \alpha_k(i) \max\{t_{local,k}(i),\ t_{UAV,k}(i) + t_{tr,k}(i)\}$$

, and if $k = k'$, then $\alpha_k(i) = 1$; otherwise $\alpha_k(i) = 0$. With the DDPG algorithm, the action to maximize the Q value can be found. The long-term average reward of system can be expressed using the Bellman equation as follows:

$$Q_\mu(s_i,\ a_i) = \mathbb{E}_\mu[r(s_i,\ a_i) + \gamma Q_\mu(s_{i+1},\ \mu(s_{i+1}))]. \tag{29}$$

4.4 State normalization

In the process of DNN training, the distribution of input in each layer changes with the change of the previous layer parameters, which slows down the training by requiring lower learning rates and careful parameter initialization. Ioffe and Szegedy proposed a batch normalization mechanism that allows training to use a higher learning rate and be less careful with initialization. We propose a state normalization algorithm to preprocess the observed states, so that DNN can be trained more effectively. It is worth noting that, different from Qiu's state normalization algorithm [25], the proposed algorithm takes the difference between maximum and minimum of each variable as the scaling factor. The proposed state normalization algorithm can solve the problem of magnitude difference of input variables.

In our work, the variables

$$E_{battery}(i),\ \mathbf{q}(i),\ \mathbf{p}_1(i),\ ...,\ \mathbf{p}_K(i),\ D_{remain}(i),\ D_1(i),\ ...,\ D_{K-1}(i)$$

and $D_K(i)$ in the state set lie in different ranges, which may result in problems during training. As shown Algorithm 1,those variables are normalized by state normalization to prevent such a problem. We use five scaling factors in the state normalization algorithm. Each factor can be explained as follows. The scaling factor γ_b is used to scale down the UAV battery capacity. Since both UAV and UE have the same x and y coordinate ranges, we use γ_x and γ_y to scale down the x and y coordinates of the UAV and UE respectively. We use $\gamma_{D_{rm}}$ to scale down the remaining tasks in the whole time period, and $\gamma_{D_{UE}}$ to scale down the task size of each UE in time slot i.

4.5 Training and testing

To learn and evaluate the DDPG-based computation offloading algorithm, there are two phases including training and testing. The training algorithm based on DDPG for computation offloading is shown in Algorithm 2. In the training process, the critic network parameters and the actor network parameters of the training behavior policy are iteratively updated. Algorithm 3 describes computation offloading testing process, which uses the trained actor network θ^μ in Algorithm 2. It should be noted that since the actor network is trained with normalized state, we also need to preprocess the input state in the testing process.

Algorithm 1 State Normalization

Input:

All state variables that require to be normalized: $E_{battery}(i),\ \mathbf{q}(i),\ \mathbf{p}_1(i),\ \ldots,\ \mathbf{p}_K(i),\ D_{remain}(i),\ D_1(i),\ \ldots,\ D_K(i)$;

Scale factors: $\gamma_b,\ \gamma_x,\ \gamma_y,\ \gamma_{D_{rm}},\ \gamma_{D_{UE}}$;

Output:

Normalized variables $E_{battery}'(i),\ \mathbf{q}'(i),\ \mathbf{p}_1'(i),\ \ldots,\ \mathbf{p}_K'(i),\ D_{remain}'(i),\ D_1'(i),\ \ldots,\ D_K'(i)$

1. $E_{battery}'(i) = E_{battery}(i) / \lambda_b$

   $x'(i) = x(i) / \lambda_x$

   $y'(i) = y(i) / \lambda_y$

   $x_k'(i) = x_k(i) / \lambda_x,\ \forall k \in \{1,\ 2,\ \ldots,\ K\}$`

   $y_k'(i) = y_k(i) / \lambda_y,\ \forall k$

   $D_{remain}'(i) = D_{remain}(i) / \lambda_{D_{rm}}$

   $D_k'(i) = D_k(i) / \lambda_{D_{UE}},\ \forall k$

2. return $E_{battery}'(i),\ \mathbf{q}'(i),\ \mathbf{p}_1'(i),\ \ldots,\ \mathbf{p}_K'(i),\ D_{remain}'(i),\ D_1'(i),\ \ldots,\ D_K'(i)$

Algorithm 2: DDPG-based computation offloading training algorithm

Input:

Training episode length E, training sample length I;

Critic learning rate α_{Critic}, actor learning rate α_{Actor};

Discount factor γ, soft update factor τ;

Experience replay buffer B_m, mini-batch size N;

Gaussian distributed behavior noise n with the mean value $\mu_e = n_0$ and standard deviation $\sigma_{e,i} = \sigma_e$;

1. Randomly initialize the weights of actor network θ^{μ} and critic network θ^{Q}, respectively
2. Initialize the target network with weights $\theta^{\mu} \leftarrow \theta^{\mu'}$ and $\theta^{Q} \leftarrow \theta^{Q'}$, respectively
3. Empty the experience replay buffer B_m
4. **for** each episode $e = 1, 2, \ldots, E$ **do**
5. Reset simulation parameters of the UAV-assisted MEC system and obtain initial observation state s_1
6. **for** $i = 1, 2, \ldots, I$ **do**
7. Normalize state s_i to $\hat{s}_i$
8. Get the action with actor network θ^{μ} and the behavior noise n_i:

$$a_i = \min(\max(\mu(\hat{s}_i \mid \theta^{\mu}) + n_{i,5}, -1), 1)$$

9. Perform action a_i, and get the reward r_i from equation (28) and observer next state s_{i+1}
10. Normalize next state s_{i+1} to $\hat{s}_{i+1}$
11. **if** the replay buffer is not full **then**

Store transition $(\hat{s}_i, a_i, r_i, \hat{s}_{i+1})$ in replay buffer B_m

else

Randomly replace a transition in replay buffer B_m with $(\hat{s}_i, a_i, r_i, \hat{s}_{i+1})$

12. Randomly sample a mini-batch of I transitions $(\hat{s}_j, a_j, r_j, \hat{s}_{j+1}), \forall j = 1, 2, \ldots, I$ from B_m
13. Set $y_j = r_j + \gamma Q'(\hat{s}_{j+1}, \mu'(\hat{s}_{j+1} \mid \theta^{\mu'}), \theta^{Q'})$
14. Update the θ^{Q} in critic network by minimizing the loss:

$$L(\theta^{Q}) = \frac{1}{N}\sum_{j=1}^{N}\left((y_j - Q(\hat{s}_j, a_j \mid \theta^{Q}))^2\right)$$

15. Update actor network θ^{μ} by the sampled policy gradient (23)
16. Soft update the critic target network according to (24) and update actor target network according to (25)
17. **end if**
18. **end for**
19. **end for**
20. **return** The actor network $\mu(\hat{s} \mid \theta^{\mu})$

Algorithm 3: DDPG-based computation offloading testing algorithm

Input:

Testing episode length E', testing steps length N;

Trained actor network $\mu(\hat{s} \mid \theta^{\mu})$

Current states: $E_{battery}(i)$, $\mathbf{q}(i)$, $\mathbf{p}_1(i)$, …, $\mathbf{p}_M(i)$, $D_{remain}(i)$, $D_1(i)$, …, $D_K(i)$, $f_1(i)$, …, $f_K(i)$

Parameters in state normalization: γ_b, γ_x, γ_y, $\gamma_{D_{rm}}$, $\gamma_{D_{UE}}$

Output:

Reward r_i

1. **for** each episode $e = 1, 2, \ldots, E'$ **do**
2. Reset simulation parameters of the UAV-assisted MEC system and obtain initial observation state s_1
3. **for** $i = 1, 2, \ldots, I$ **do**
4. Normalize state s_i to $\hat{s}_i$
5. Get the action with the trained actor network: $a_i = \mu(\hat{s}_i \mid \theta^{\mu})$
6. Perform action a_i, and obtain the reward r_i from equation (28)
7. **end for**
8. **end for**

5 Results and analysis

In this section, we illustrate the proposed DDPG-based framework for computation offloading in the UAV-assisted MEC system by numerical simulations. Firstly, the setting of simulation parameters is introduced. Then, the performance of the DDPG-based framework is validated under different scenarios, and compared with other baseline schemes.

5.1 Simulation setting

In the UAV-assisted MEC system, we consider a 2D square area with $K = 4$ UEs randomly distributed in $L \times W = 100 \times 100$ m^2 [10], and assume that the UAV flies at a fixed height $H = 100$ m [11]. As defined in [26], the gross mass of UAV is $M_{UAV} = 9.65$ kg. The whole time period $T = 400$ s is divided into $I = 40$ time slots. Referring to [9], the maximum UAV flight speed is $v_{\max} = 50$ m/s and the flight time of UAV is $t_{fly} = 1$ s in each time slot. The channel power gain is set to be $\alpha_0 = -50$ dB [9] at a reference distance of 1 m. The transmission bandwidth is set to $B = 1$ MHz. The noise power of the receiver is assumed to be $\sigma^2 = -100$ dBm [9] without signal blockage. If the signal is blocked during transmission between UAV and UE k, i.e., $f_k(i) = 1$, the signal.

penetration loss is $P_{NLOS} = 20$ dB [19]. We assume that the transmission power of UEs $P_{up} = 0.1$ W [22], UAV battery capacity $E_b = 500$ kJ [26] and the required CPU cycles per bit $s = 1000$ cycles/bit [9]. The computing capability of the UE and the MEC server is set to $f_{UE} = 0.6$ GHz and $f_{UAV} = 1.2$ GHz [9], respectively. The scaling factors in the proposed state normalization algorithm are set to $\gamma_b = 5 \times 10^5$, $\gamma_x = 100$, $\gamma_y = 100$, $\gamma_{D_{rm}} = 1.05 \times 10^8$ and $\gamma_{D_{UE}} = 2.62 \times 10^6$ respectively. The detailed simulation parameters are listed in Table 1, unless otherwise specified. In our experiments, the average reward taken from multiple runs of Algorithm 3 with same settings are used for performance comparison.

For comparison, four baseline approaches are described as follows:

Table 1 Simulation parameters

Parameter	Default value	Parameter	Default value
K	4	L, W, H	100 m
M_{UAV}	9.65 kg	T	400 s
I	40	$v_{\max}$	50 m/s
t_{fly}	1 s	α_0	− 50 dB
B	1 MHz	σ^2	− 100 dBm
P_{NLOS}	20 dB	P_{up}	0.1 W
E_b	500 kJ	S	1000 cycles/bit
f_{UE}	0.6 GHz	f_{UAV}	1.2 GHz
γ_b	5×10^5	γ_x, γ_y	100
$\gamma_{D_{rm}}$	1.05×10^8	$\gamma_{D_{UE}}$	2.62×10^6

– Offloading all tasks to UAV (Offload-only): During each time slot, the UAV provides computing services to the UE at a fixed location in the area center. The UE offloads all its computing tasks to the MEC server on UAV.
– Executing all tasks locally (Local-only): Without the assistance of UAV, all computing tasks of UE are executed locally.
– Actor Critic-based computation offloading algorithm (AC): To evaluate the performance of the proposed DDPG-based computation offloading algorithm, the continuous action space based RL algorithm, i.e., AC [11], is also implemented for the computation offloading problem. In order to compare it with DDPG fairly, AC also applies state normalization.
– DQN-based computation offloading algorithm (DQN): The traditional discrete action space based DQN [12] algorithm is compared with the proposed DDPG-based algorithm. During the UAV flight, the angle levels are defined as $\mathcal{B} = \{0, \pi/5, \ldots, 2\pi\}$, and the speed levels are expressed as $\mathcal{V} = \{0, v_{\max}/10, \ldots, v_{\max}\}$ and the offloading ratio levels can be set as $\mathcal{O} = \{0, 0.1, \ldots, 1.0\}$. In order to compare it with DDPG and AC fairly, DQN also applies state normalization.

5.2 Simulation results and discussions

5.2.1 Parametric analysis

We first conduct a series of experiments to determine the optimal values for important hyper-parameters used in algorithm comparisons. The convergence performance of the proposed algorithm with different learning rates is shown in Fig. 2. We assume that the learning rates of the critic network and actor network are different. Firstly, we can clearly observe that when $\alpha_{Actor} = 0.1$, $\alpha_{Critic} = 0.2$ or $\alpha_{Actor} = 0.001$, $\alpha_{Critic} = 0.002$, the proposed algorithm can converge. However, the proposed algorithm converges to a local optimal solution when $\alpha_{Actor} = 0.1$ and $\alpha_{Critic} = 0.2$. The reason is that the large learning rates will make both critic network and actor network take a large update step. Secondly, we can find that when the learning rate is very small, i.e., $\alpha_{Actor} = 0.00001$ and $\alpha_{Critic} = 0.00002$, the algorithm cannot converge. That is because lower learning rates will result in a slower update rate of DNNs, which requires more iterative episodes to converge. Thus, the best learning rates of actor network and critic network are $\alpha_{Actor} = 0.001$ and $\alpha_{Critic} = 0.002$, respectively.

In Fig. 3, we compare the impact of different discount factors γ on the convergence performance of the proposed algorithm. It is observed that when the discount factor $\gamma = 0.001$, the trained computation offloading policy has the best performance. The reason is that the environment in different periods varies greatly, so the data of the whole time period cannot completely represent the long-term behavior. A larger γ means that the Q table will consider the data collected in the whole time period as long-term data, which leads to the poor generalization ability of different time periods. Therefore, an appropriate value of γ will improve the ultimate performance of our trained policies, and we set the discount factor γ to 0.001 in the following experiments.

Figure 4 shows the performance comparisons of the proposed algorithm in terms of the processing delay under different exploration parameter σ_e. The convergence performance of the proposed algorithm is greatly affected by this exploration parameter. When the algorithm converges at $\sigma_e = 0.1$, the optimal delay fluctuates at 63 s. A larger value of σ_e produces a larger random noise distribution space, which makes it possible to explore a larger spatial

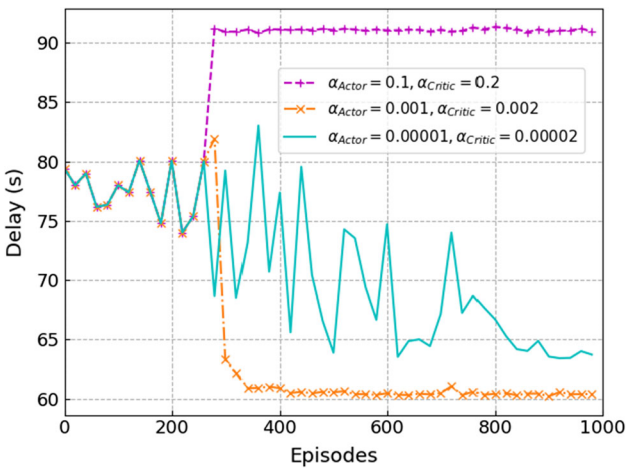

Fig. 2 Convergence performance of the DDPG-based algorithm under different learning rates

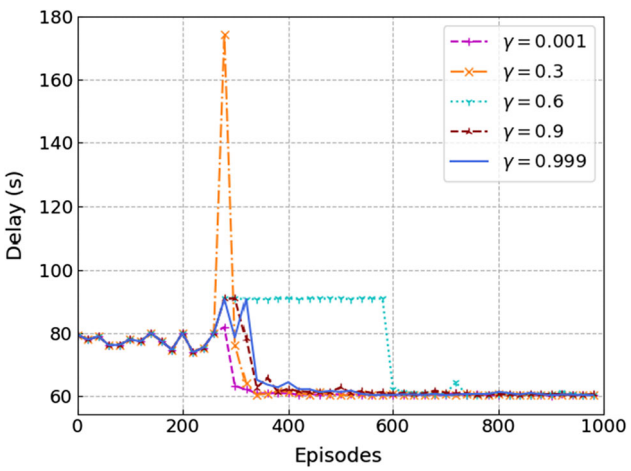

Fig. 3 Convergence performance with different discount factors

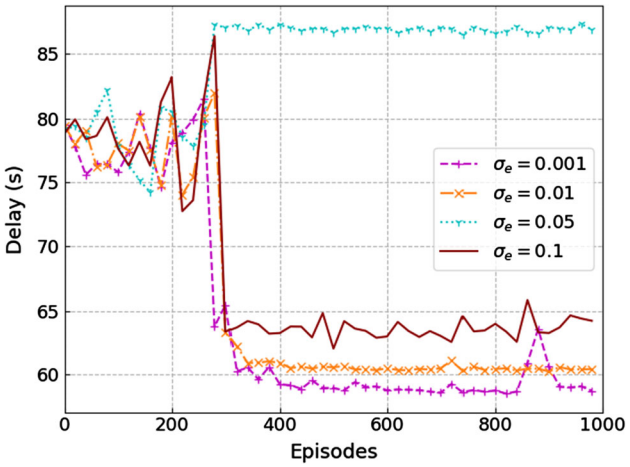

Fig. 4 Performance of the delay under different exploration parameter settings

range for the agent. When $\sigma_e = 0.001$, the performance of the algorithm decreases at 850 episodes, as the algorithm falls into a local optimal solution due to a small σ_e. Therefore, a large number of experiments are needed to obtain the appropriate exploration setting in the UAV-assisted scenario. Hence, we choose $\sigma_e = 0.01$ for better performance in the following experiments.

Figure 5 shows the influence of training strategies that do not use state normalization or behavior noise in DDPG training algorithms. On the one hand, if the DDPG algorithm is trained without behavior noise, the convergence speed of the algorithm will be slower. On the other hand, if the policy is trained without state normalization, i.e., without introducing scale factor into the state normalization, the training algorithm will not work. The reason is that without the state normalization strategy, the values of $E_{battery}(i)$, $D_{remian}(i)$ and $D_k(i)$ are too large, which results in the random initialization of DNNs to output a larger value. Thus, if the state normalization strategy proposed by us is not adopted in DDPG algorithm, the algorithm will eventually become a greedy algorithm.

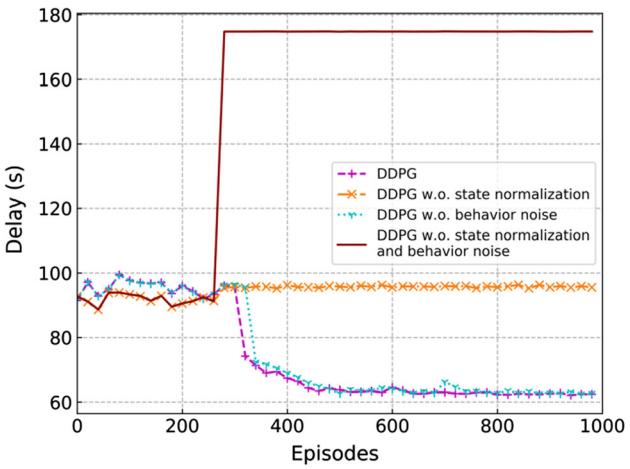

Fig. 5 Performance without state normalization or behavior noise

5.2.2 Performance comparison

Figure 6 shows the performance comparison between different algorithms. In Fig. 6a, we trained the DNNs of RL algorithms for a total of 1,000 episodes. From this figure, it can be observed that the AC algorithm cannot converge as the number of iteration increases, while both DQN and DDPG algorithms can achieve convergence. That's because the AC algorithm suffers from the problem of simultaneous updating of the actor network and the critic network. The action selection of the actor network depends on the value function of the critic network, but the critic network itself is difficult to converge. Therefore, the AC algorithm may not converge in some cases. In contrast, both DQN and DDPG benefit from the dual network structure of an evaluation network and a target network,

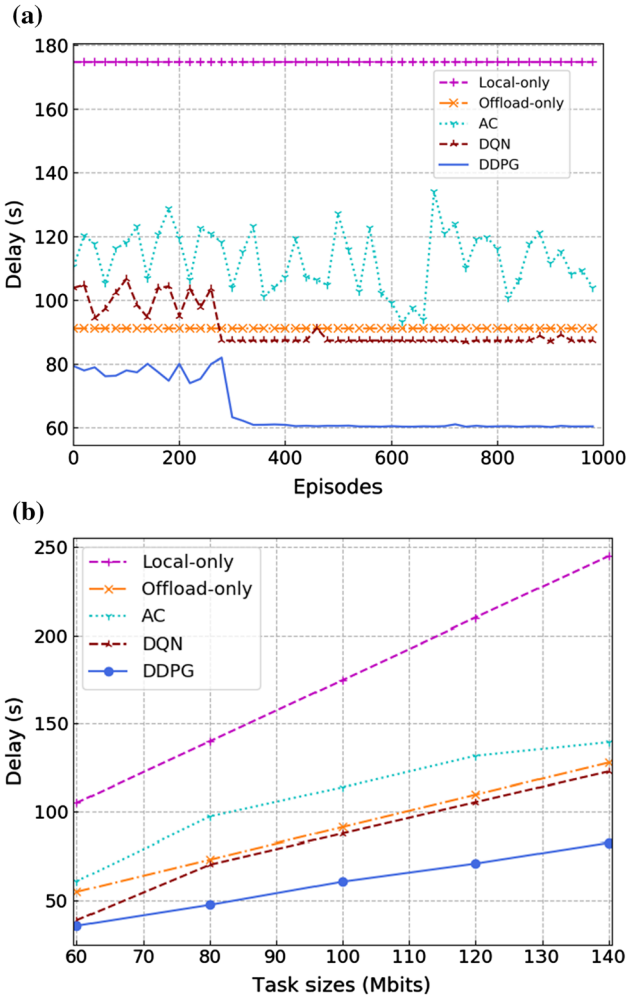

Fig. 6 (**a**) Performance of different algorithms under task sizes D = 100 Mb. (**b**) Delay with different task sizes

which can be used to cut off the correlation between training data to find the optimal action strategy. Using the delay results obtained after the algorithm convergence, we compare these algorithms under different settings of task size and show the results in Fig. 6b. In Fig. 6b, the delay of the DDPG algorithm is always the lowest among the five algorithms for the same task size. Due to the exploration of discrete action space and non-negligible spaces between the available actions, the DQN cannot accurately find the optimal offloading strategy. However, the DDPG algorithm explores a continuous action space and takes a precise action, which finally obtains the optimal strategy and significantly reduces the delay. Besides, the DQN algorithm converges with a processing delay much higher than DDPG. The Offload-only and Local-only algorithms cannot fully utilize the computing resources in the whole system. Thus, for the same task size, the processing delay of the DDPG algorithm is significantly lower than those of Offload-only and Local-only. Moreover, as the task size increases, the increasing speed of processing delay optimized by the DDPG algorithm is much slower than those of Offload-only and Local-only, which indicates the advantages of our proposed algorithm.

Figure 7a and b show the performance of the same group of experiments in terms of delay and offloading ratio. Figure 7a shows the convergence performance of DQN scheme and DDPG scheme with different computing capabilities of UE. The reason why the proposed scheme is not compared with the AC scheme is that the AC scheme is still not convergent. We can find that when the computing capability of UE is small, i.e., $f_{UE} = 0.4$ GHz, the processing delay optimized by the two optimization schemes is higher than that when the computing capability of UE $f_{UE} = 0.6$ GHz. On the other hand, when the computing capability of UE is large, the average offloading ratio in the system is small according to Fig. 7b, and thus the UE prefers to execute the task locally. The smaller the computing capability of UE, the slower the data processing speed of the system in the same time, resulting in the larger maximum delay between local execution and offloading. Figure 7c shows the optimized delay comparison between the proposed scheme and DQN scheme under different CPU frequency conditions. It can be observed from Fig. 7c that the proposed DDPG scheme achieves lower delay compared with DQN scheme under different UE computing capabilities. This reason is that, DDPG scheme can output a number of continuous actions, instead of a limited discrete set of actions in DQN. Thus, DDPG can find an accurate factor that affects greatly the delay in the continuous action control system, i.e., the offloading ratio.

In Fig. 8, we compare the average processing delay between the proposed DDPG scheme, DQN, Offload-only and Local-only, as the number of UEs varies from 1 to 10.

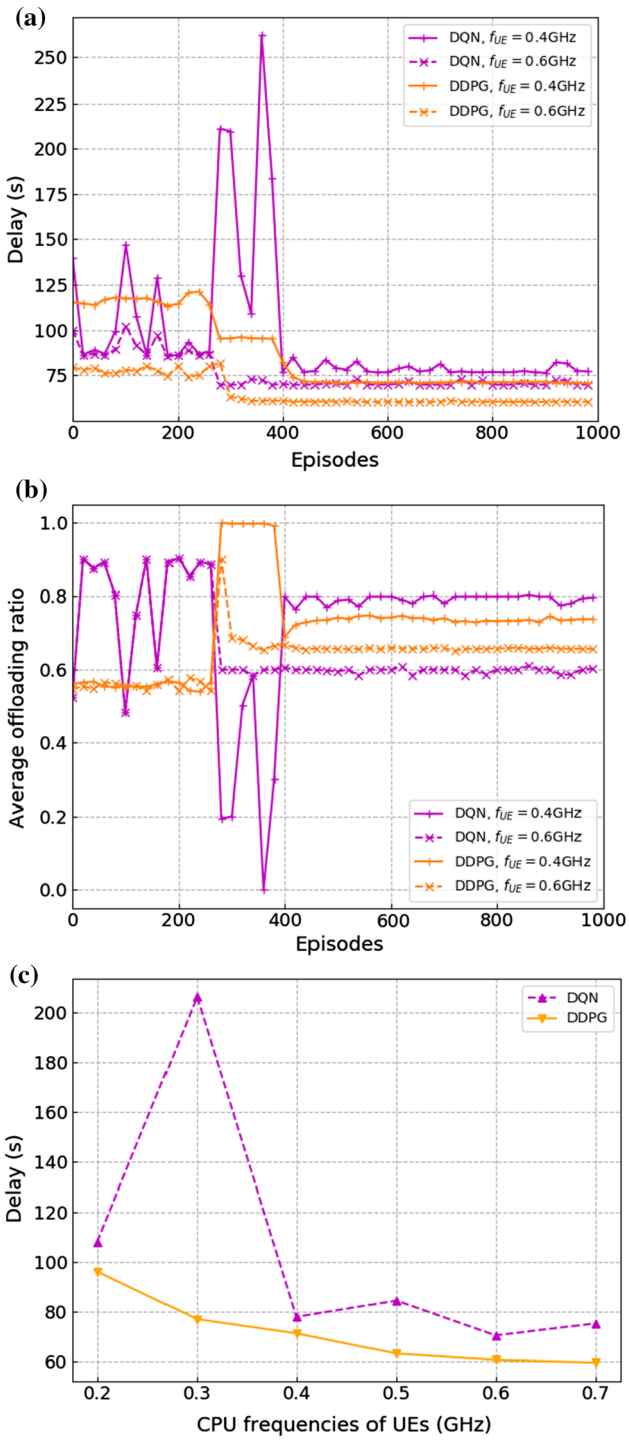

Fig. 7 (**a**) Performance of DQN and DDPG under different computing capabilities of UE. (**b**) Offloading ratio of DQN and DDPG under different computing capabilities of UE. (**c**) Comparison between DQN and DDPG

We assume that the total task size to be completed in a time period is the same under different number of UEs. As shown in Fig. 8, the average processing delay of all schemes except DQN is almost constant as the number of

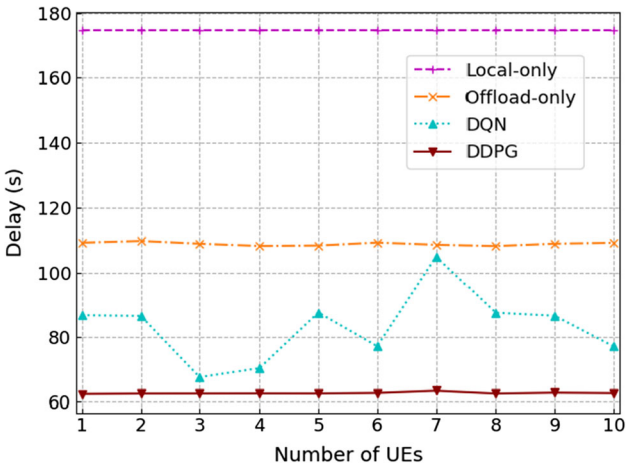

Fig. 8 Comparison of processing delay as the number of UEs varies from 1 to 10

UEs increases. With the increase of the number of UEs, the processing delay of DQN scheme fluctuates at about 86 s. The reasons can be explained as follows. The value ranges of the DQN output actions under different UE numbers vary greatly. Thus, when the sample is used as the input of DNN training, DNNs may prefer to output a larger value. The actor network of DDPG outputs multi-dimensional actions to ensure that the input data of DNNs are in the same range, i.e., [0, 1], which ensures the convergence and stability of DDPG algorithm. Besides, the proposed DDPG scheme achieves the lowest delay. This is due to the fact that DDPG scheme can find the optimal value in the continuous action and obtain the optimal control policy.

6 Conclusion

In this paper, we study computation offloading in a UAV-assisted MEC system, where a flying UAV provides communication and computing services for UEs. We aim to minimize the sum of the maximum processing delays in the whole time period by jointly optimizing user scheduling, task offloading ratio, UAV flight angle and flight speed. In order to solve the nonconvex optimization problem with discrete variables, we introduce the DDPG training algorithm to obtain the optimal offloading strategy. We describe the RL related background knowledge and the DDPG training process. Then, the state normalization algorithm is proposed to make the DDPG algorithm easier to converge. We analyze the parameters of DDPG algorithm through simulation, and compare the impacts of different parameters, including learning rate, discount factor and training strategies. The simulation results show that, as compared with the baseline algorithms, our proposed algorithm can achieve better performance in terms of processing delay. For future work, we will investigate the performance of our algorithm with other popular RL algorithm, e.g., ACER, ACKTR, PRO2 and TRPO [21, 27].

Acknowledgements This work is supported by the Beijing Municipal Natural Science Foundation (Joint Fund for Frontier Research of Fengtai Rail-Transit) under Grant L191019, the Open Project of Key Laboratory of Industrial Internet of Things & Networked Control, Ministry of Education under Grant 2019FF03, the Open Project of Beijing Intelligent Logistics System Collaborative Innovation Center under Grant BILSCIC-2019KF-10, the Traffic Control Technology Innovation Fund under Grant 9907006515, the Research Base Project of Beijing Municipal Social Science Foundation under Grant 18JDGLB026 and the Science and Technique General Program of Beijing Municipal Commission of Education under Grant KM201910037003.

References

1. Dinh, H. T., Lee, C., Niyato, D., & Wang, P. (2013). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing, 13*(18), 1587–1611
2. Mach, P., & Becvar, Z. (2017). Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials, 19*(3), 1628–1656
3. Abbas, N., Zhang, Y., & Taherkordi, A. (2018). Mobile edge computing: A survey. *IEEE Internet of Things Journal, 5*(1), 450–465
4. Mao, Y., You, C., Zhang, J., Huang, K., & Letaief, K. B. (2017). A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials, 19*(4), 2322–2358
5. Tran, T. X., & Pompili, D. (2018). Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Transactions on Vehicular Technology, 18*(1), 856–868
6. Zhao, J., Li, Q., Gong, Y., & Zhang, K. (2019). Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks. *IEEE Transactions on Vehicular Technology, 18*(8), 7944–7956
7. Chen, Z., & Wang, X. (2020). Decentralized computation offloading for multi-user mobile edge computing: A deep reinforcement learning approach. *EURASIP Journal on Wireless Communications and Networking, 2020*(1), 1–21
8. Feng, G., Wang, C., & Li, B. (2019). UAV-assisted wireless relay networks for mobile offloading and trajectory optimization. *Peer-to-Peer Networking and Applications, 12*(6), 1820–1834
9. 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
10. Diao, X., Zheng, J., Cai, Y., Wu, Y., & Anpalagan, A. (2019). Fair data allocation and trajectory optimization for UAV-Assisted mobile edge computing. *IEEE Communications Letters, 23*(12), 2357–2361
11. Cheng, N., Lyu, F., & Quan, W. (2019). Space/aerial-assisted computing offloading for IoT applications: A learning-based approach. *IEEE Journal on Selected Areas in Communications, 37*(5), 1117–1129
12. Li, J., Liu, Q., Wu, P., Shu, F., & Jin, S. (2018). Task offloading for uav-based mobile edge computing via deep reinforcement

learning. *IEEE/CIC International Conference on Communications in China (ICCC), 2018*, 798–802
13. Xiong, J., Guo, H., & Liu, J. (2019). Task offloading in UAV-aided edge computing: Bit allocation and trajectory optimization. *IEEE Communications Letters, 23*(3), 538–541
14. Selim, M.M., Rihan, M., & Yang, Y. (2020). Optimal task partitioning, Bit allocation and trajectory for D2D-assisted UAV-MEC systems. *Peer-to-Peer Networking and Applications*, 1–10.
15. Ge, L., Dong, P., & Zhang, H. (2020). Joint beamforming and trajectory optimization for intelligent reflecting surfaces-assisted UAV communications. *IEEE Access, 8*, 78702–78712
16. Lillicrap, T.P., Hunt, J.J., & Pritzel, A. (2015). Continuous control with deep reinforcement learning. arXiv:1509.02971.
17. Dai, Y., Xu, D., & Maharjan, S. (2019). Artificial intelligence empowered edge computing and caching for internet of vehicles. *IEEE Wireless Communications, 26*(3), 12–18
18. Fang, W., Ding, S., Li, Y., Zhou, W., & Xiong, N. (2019). OKRA: optimal task and resource allocation for energy minimization in mobile edge computing systems. *Wireless Networks, 25*(4), 2851–2867
19. Coldrey, M. (2013). Non-Line-of-Sight small cell backhauling using microwave technology. *IEEE Communications Magazine, 51*(9), 78–84
20. Saleem, U., Liu, Y., Jangsher, S., & Li, Y. (2018). Performance guaranteed partial offloading for mobile edge computing. In *2018 IEEE Global Communications Conference (GLOBECOM)* (pp. 1–6). IEEE, New York.
21. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. Cambridge: MIT press.
22. Nie, J., & Haykin, S. (1999). A Q-learning-based dynamic channel assignment technique for mobile communication systems. *IEEE Transactions on Vehicular Technology, 48*(5), 1676–1687
23. Mnih, V., Kavukcuoglu, K., & Silver, D. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533
24. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014, January). Deterministic policy gradient algorithms. In *International conference on machine learning* (pp. 387–395). PMLR.
25. Qiu, C., Hu, Y., & Chen, Y. (2019). Deep deterministic policy gradient (DDPG)-based energy harvesting wireless communications. *IEEE Internet of Things Journal, 6*(5), 8577–8588
26. Jeong, S., Simeone, O., & Kang, J. (2017). Mobile edge computing via a UAV-mounted cloudlet: Optimization of bit allocation and path planning. *IEEE Transactions on Vehicular Technology, 67*(3), 2049–2063
27. Galatolo, F. A., Cimino, M. G., & Vaglini, G. (2021). Solving the scalarization issues of Advantage-based Reinforcement Learning algorithms. *Computers & Electrical Engineering*, *92*, 107117.

Yunpeng Wang received his B.S. degree from the School of Information Engineering from Ningxia University in 2018. Currently, he is a graduate student in the School of Computer and Information Technology at Beijing Jiaotong University. His main current research interests include Deep Reinforcement Learning and Mobile Edge Computing.

Weiwei Fang received the B.S. degree from Hefei University of Technology, Hefei, China, and the Ph.D. degree from Beihang University, Beijing, China, in 2003 and 2010, respectively. He is currently an Associate Professor with the School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China. His research interests include edge computing, cloud computing, and Internet of Things. He has published over 60 papers in journals, international conferences/workshops.

Yi Ding received the Ph.D. degree from Beihang University, Beijing, China, in 2014. Currently, he is an Assistant Professor in the School of Information at Beijing Wuzi University. His current research interests include edge computing and blockchain systems.

Naixue Xiong received the Ph.D. degree in sensor system engineering from Wuhan University, and the Ph.D. degree in dependable sensor networks from the Japan Advanced Institute of Science and Technology. Before he attended Tianjin University, China, he was with Northeastern State University, Georgia State University, Wentworth Technology Institution, and Colorado Technical University (Full Professor about five years) about ten years. He is currently a Professor with the College of Intelligence and Computing, Tianjin University. His research interests include cloud computing, security and dependability, parallel and distributed computing, networks, and optimization theory. He has published over 300 international journal articles and over 100 international conference papers. Some of his works were published in IEEE JSAC, IEEE or ACM TRANSACTIONS, ACM Sigcomm Workshop, IEEE INFOCOM, ICDCS, and IPDPS. He is a Senior Member of the IEEE Computer Society. He received the Best Paper Award in the 10th IEEE International Conference on High Performance Computing and Communications (HPCC-08) and the Best Student Paper Award in the 28th North American Fuzzy Information Processing Society Annual Conference (NAFIPS2009). He has been the General Chair, the Program Chair, the Publicity Chair, a PC Member, and a OC Member of over 100 international conferences, and a reviewer of about 100 international journals, including IEEE JSAC, IEEE SMC (Park: A/B/C), the IEEE TRANSACTIONS ON COMMUNICATIONS, the IEEE TRANSACTIONS ON MOBILE COMPUTING, and the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. He is serving as the Editor-in-Chief and an Associate Editor or an Editor Member for over ten international journals, including an Associate Editor for the IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS: SYSTEMS, Information Science, the Editor-in-Chief for the Journal of Internet Technology (JIT) and the Journal of Parallel and Cloud Computing (PCC), and a Guest Editor for over ten international journals, including Sensors Journal, WINET, and MONET. He is also the Chair of Trusted Cloud Computing Task Force, the IEEE Computational Intelligence Society (CIS), and the Industry System Applications Technical Committee.