Deep Reinforcement Learning Based Computation Offloading for Mobility-Aware Edge Computing

Minyan Shi[1(✉)], Rui Wang[1,2], Erwu Liu[1], Zhixin Xu[1], and Longwei Wang[3]

[1] School of Electronics and Information Engineering, Tongji University, Shanghai, China
1832915@tongji.edu.cn

[2] Shanghai Institute of Intelligent Science and Technology, Tongji University, Shanghai, China

[3] Department of Electrical Engineering, University of Texas at Arlington, Arlington, TX 76013, USA

Abstract. Mobile Edge Computing (MEC) has become the most likely network architecture to solve the problems of mobile devices in terms of resource storage, computing performance and energy efficiency. In this paper, we first model the MEC system with the exploitation of mobility prediction. Considering the user's mobility, the deadline constraint and the limited resources in MEC servers, we propose a deep reinforcement learning approach named deep deterministic policy gradient (DDPG) to learn the power allocation policies for MEC servers users. Then, the aim of the policy is to minimize the overall cost of the MEC system. Finally, simulation results are illustrated that our proposed algorithm achieves performance gains.

Keywords: Mobile Edge Computing · Computation offloading · Mobility · Deep reinforcement learning

1 Introduction

The rapid development of the Internet has made future networks face the challenges of higher speed and lower latency. Although the processing capacity of the new mobile device's central processing unit (CPU) is becoming more and more powerful, it is unable to handle huge amounts of tasks in a short time [1].

In view of this, Mobile Edge Computing (MEC) has become the most likely network architecture to realize the 5G vision and has attracted wide attention [2, 3]. MEC refers to the deployment of computing and storage resources at the edge of the mobile network to provide Internet services and cloud computing capabilities for mobile networks, providing users with ultra-low latency and high bandwidth networks.

Computation offloading is one of the key technologies in MEC systems [4], which is the terminal device hands over some or all of the computing tasks to the

H. Gao et al. (Eds.): ChinaCom 2019, LNICST 312, pp. 53–65, 2020.
https://doi.org/10.1007/978-3-030-41114-5_5

cloud computing environment to solve the problems of mobile devices in terms of resource storage, computing performance and energy efficiency. For example, in the fields of automatic driving, vehicles must sense road conditions, obstacles, driving information of surrounding vehicles in real time, and these information can be quickly calculated and transmitted through MEC offloading technology.

Computation offloading strategies for MEC servers recently have been widely investigated in the literature in order to achieve higher energy efficiency or better computation experience. In [5], a dimensional search algorithm was studied to determine whether the buffer task is offloaded to the MEC servers with the goal of optimizing the time delay in each time slot. Considering the limited computing resources of the MEC servers for offloading decisions and resource allocation, the authors in [6] proposed a layered MEC deployment architecture and solved the multi-user offloading problem by using Stackelberg game theory. Moreover, with the dynamic voltage and frequency (DVFS) techniques, CPU-cycle frequency was flexibly controlled with other features in [7,8], where the system cost, defined as weighted sum of energy consumption and execution time, has been reduced. Additionally, exploiting users mobility for offloading strategies has also received much attention from researchers. In [9], the resource allocation policy is designed by considering the vehicles mobility and the hard service deadline constraint.

Machine learning has brought unprecedented algorithmic capabilities in providing adept solutions to a great span of complex learning and planning tasks. So there have been some attempts to adopt deep reinforcement learning (DRL) in the design of online resource allocation and scheduling in wireless networks, especially for some recent works targeting computation offloading in MEC servers. The authors in [10] proposed a continuous action space based DRL approach named deep deterministic policy gradient (DDPG) to learn efficient computation offloading policies independently at each mobile user. Most of the existing work is focused on the resource allocation of MEC without considering the insufficient resources of MEC. However, in some cases, it is very likely that there will not be enough MEC resources for computation offloading.

In this paper, we propose a continuous action space based algorithm named deep deterministic policy gradient (DDPG) to derive better power control of local execution and task offloading by considering the mobility of users and hard deadline delay. Specifically, the contributions of this paper can be summarized as follows.

(1) We model the MEC system with the exploitation of mobility prediction and design dynamic computation offloading policies, where both users and MEC servers have computing capabilities. The agent decides the dynamic power allocation of both local execution and computation offloading during the movement. In case the user offload the tasks to a MEC server with insufficient computing resources, the MEC server passes the unfinished tasks to the core network.
(2) We design a DRL framework based on DDPG to learn efficient policies for power allocation of both local execution and computation offloading with the goal of minimizing the cost of communication and computation.

(3) We present numerical simulations to illustrate the performance of the DDPG algorithm and analyze the effects of different parameters on the results.

The rest of this paper is organized as follows. Section 2 describes the MEC system model. We briefly introduce the deep reinforcement learning and formulate the power allocation optimization problem by using deep reinforcement learning in Sect. 3. Numerical results will be illustrated in Sect. 4. Last, Sect. 5 concludes the paper and outlines future work plans.

2 System Model

2.1 Network Model

We consider a hybrid network which includes one mobile user k, $\mathcal{Q}$ MEC servers and the core network as shown in Fig. 1. Let $\mathcal{Q} = \{1, \ldots, q\}$ be the sets of the MEC servers. Note that every MEC server is equipped with N antennas and have the capability of computing and transmitting. For example, we set the scene to a vehicular network and regard mobile users as vehicles. The red vehicle can offload tasks to different MEC servers during its movement. In case several vehicles simultaneously offload tasks to the same MEC server and the requested MEC server lacks enough resources to process tasks from the user, it will transmit unprocessed tasks to the core network.

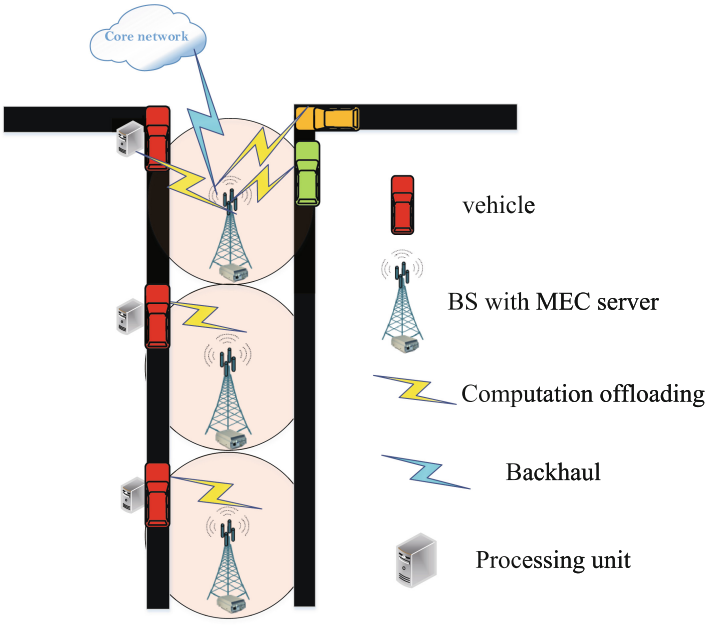

Fig. 1. Mobility-aware computation offloading system. (Color figure online)

2.2 Communication Model

Assume that the channel vector of vehicle-to-MEC links and MEC-to-core network are both time-varying and modeled as the finite-state Markov chain (FSMC). For each time slot $t \in \mathcal{T}$, the received signal of the MEC server q from user k can be written as

$$\mathbf{y}_q(t) = \mathbf{g}_{k,q}(t)x_k(t) + \mathbf{n}_q(t) \tag{1}$$

where $\mathbf{g}_{k,q}(t)$ is the $\mathrm{N} \times 1$ channel matrix between user k and MEC server q, $x_k(t)$ is the data symbol with unit variance, and $\mathbf{n}_q(t)$ is a vector of additive white Gaussian noise with variance σ^2. Then the received signal for user k is

$$\mathbf{v}_{k,q}^T\mathbf{y}_q(t) = \mathbf{v}_{k,q}^T\mathbf{g}_{k,q}(t)x_k(t) + \mathbf{v}_{k,q}^T\mathbf{n}_q(t) \tag{2}$$

where $\mathbf{v}_{k,q}^T$ is the normalized Beamforming vector and $||\mathbf{v}_{k,q}|| = 1$.

Denoting $p_{k,q}^o$ as the transmission power of user k to MEC server q. Thus, the corresponding signal-to-noise (SNR) can be derived by

$$\gamma_{q,k}(t) = \frac{\mathbf{E}\left\{|\mathbf{v}_{k,q}^T\mathbf{g}_{k,q}x_k(t)|^2\right\}}{\mathbf{E}\left\{|\mathbf{v}_{k,q}^T\mathbf{n}_q|^2\right\}} = \frac{|\mathbf{v}_{k,q}^T\mathbf{g}_{k,q}|^2 p_{k,q}^o}{\sigma_0^2||\mathbf{v}_{k,q}||^2} \tag{3}$$

$$\max\ SNR = \frac{|\mathbf{v}_{k,q}^T\mathbf{g}_{k,q}|^2}{\sigma_0^2||\mathbf{v}_{k,q}||^2} \tag{4}$$

$$s.t. \quad ||\mathbf{v}_{k,q}|| = 1 \tag{5}$$

Then we can obtain that

$$\mathbf{v}_{k,q} = \frac{\mathbf{g}_{k,q}^*}{||\mathbf{g}_{k,q}||} \tag{6}$$

So, the communication rate of user k and MEC server q can be expressed as

$$r_{q,k}(t) = W_q log(1 + \gamma_{q,k}(t)) \tag{7}$$

Due to the limitations in computing resources at the MEC servers, tasks can be transferred from MEC server q to core network. For each time slot $t \in \mathcal{T}$, the received signal of the core network can be written as

$$\mathbf{y}_c(t) = \mathbf{g}_{q,c}^T(t)\mathbf{u}x_c(t) + \mathbf{n}_c(t) \tag{8}$$

where $\mathbf{g}_{q,c}(t)$ is the $\mathrm{N} \times 1$ channel matrix between MEC server q and core network, $p_{q,c}^o$ is the transmission power between MEC server q and core network. Thus, the corresponding signal-to-noise (SNR) can be derived by

$$\gamma_{q,c}(t) = \frac{\mathbf{E}\left\{||\mathbf{g}_{q,c}^T\mathbf{u}(t)x_c(t)||^2\right\}}{\mathbf{E}\left\{n_c(t)\right\}} = \frac{|\mathbf{g}_{q,c}^T\mathbf{u}|^2 p_{q,c}^o}{\sigma_1^2} \tag{9}$$

$$\max \ SNR = \frac{|\mathbf{u}\mathbf{g}_{q,c}^{T}|^2}{\sigma_1^2} \tag{10}$$

$$s.t. \quad ||\mathbf{u}||_2^2 \leq \ p_{q,c}^{o} \tag{11}$$

Thus, we can obtain that

$$\mathbf{u} = \sqrt{\alpha}\mathbf{g}_{q,c}^{*} \tag{12}$$

$$\alpha = \frac{p_{q,c}^{o}}{||\mathbf{g}_{q,c}||^2} \tag{13}$$

So, the communication rate of MEC q and core network can be expressed as

$$r_{q,c}(t) = W_c log(1 + \gamma_{q,c}(t)) \tag{14}$$

2.3 Computation Model

Let C_k and C_q be the number of CPU cycles required for user k and MEC server q to accomplish one task bit, respectively. Since the tasks can be computed locally or offloaded to MEC servers, we can obtain that

(1) *Local computing:* The CPU frequency at t-th slot can be written by

$$f_k^l(t) = \sqrt[3]{p_k^l(t)/k} \tag{15}$$

where $p_k^l(t)$ is the allocated power for computing locally, k is the effective switched capacitance depending on the chip architecture (CPUs per second). So the computing rate (bits computed per second) for user k locally is expressed as

$$R_k^l(t) = \frac{f_k^l(t)}{C_k} = \frac{\sqrt[3]{p_k^l(t)/k}}{C_k} \tag{16}$$

(2) *Edge computing:* The computing rate (bits computed per second) for MEC server q is expressed as

$$R_{q,k}^{mec}(t) = \frac{\sqrt[3]{p_{q,k}^{mec}(t)/k}}{C_q} \tag{17}$$

where $p_{q,k}^{mec}(t)$ is the allocated power for computation offloading.

2.4 Mobility Model

We model the mobility of users by contact time T_{con}^{mec} and user k keeps contact with the same MEC within T_{con}^{mec}. Besides, the number of contacts between user k and MEC server q follow the Poisson distribution with parameters of $\lambda_{q,k}$. Hence the connect frequency $\lambda_{q,k}$ accounts for mobility intensities [11].

The duration T_{con}^{mec} includes transmitting time and computing time, so we obtain

$$T_{con}^{mec} = \frac{H_{q,k}}{r_{q,k}} + \frac{\alpha_q H_{q,k}}{R_{q,k}^{mec}} + \frac{(1-\alpha_q)H_{q,k}}{r_{q,c}} \tag{18}$$

$$H_{q,k} = \frac{T_{con}^{mec}}{\frac{1}{r_{q,k}} + \frac{\alpha_q}{R_{q,k}^{mec}} + \frac{(1-\alpha_q)}{r_{q,c}}} \tag{19}$$

where $\alpha_q \in [0,1]$ is the weight of the bits of tasks computed in MEC server q, which can be learned from DDPG algorithm. $H_{q,k}^n$ is the maximum bits of offloading tasks from user k to MEC server q during one contact and $\lambda_{q,k}t$ is the number of contacts for user k with MEC server q. Then we can calculate the total offloading bits of user k within time slot t as

$$V_{k,q}(Z) = \sum_{n=1}^{\lambda_{q,k}t} H_{q,k}^n \tag{20}$$

Thus the probability distribution function (PDF) of variable $V_{k,q}(Z)$ can be expressed as

$$f_{V_{k,q}(Z)}(x) = \frac{x^{\lambda_{q,k}t-1}e^{-xH_{q,k}}}{(H_{q,k})^{-\lambda_{q,k}t}\Gamma(\lambda_{q,k}t)} \tag{21}$$

Denoting Z_q as the computation capacity of MEC server q, so the probability that user k offloads m bits tasks to MEC server q is

$$P_{k,q}(m) = \begin{cases} \int_m^{m+1} f_{V_{k,q}(Z)}(x)dx, & 0 \le m < Z_q \\ \int_m^{\infty} f_{V_{k,q}(Z)}(x)dx, & m = Z_q \\ 0, & otherwise \end{cases} \tag{22}$$

3 Deep Reinforcement Learning

In this section, we will briefly introduce the deep reinforcement learning and DDPG.

3.1 Deep Deterministic Policy Gradient

Reinforcement learning is learning how to map situations to actions by maximizing reward function. In reinforcement learning, the environment is typically formulated as a Markov decision process (MDP). So far, describe MDP with a tuple (S, A, P, R, γ) including an agent, a set of possible states S, a set of available actions A, transition probability matrix P, a reward function R and discount factor of returns γ. We can define the value of the current state s_t with the long-term expected discounted reward, called the value function

$$V_\pi(s_t) = \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^k R_{t+k+1} | s_t\Big] \tag{23}$$

Further consider the value of each action, we can define state-action value function after taking action a_t in state s_t:

$$Q_\pi(s_t, a_t) = \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^k R_{t+k+1} | s_t, a_t\Big] \tag{24}$$

The optimal state-action value function $Q^*(s_t, a_t)$ is the maximum among all policies and the optimal policy π^* can be calculated by maximizing $Q^*(s_t, a_t)$:

$$Q^*(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t) \tag{25}$$

$$\pi^*(s_t) = \arg\max_{a_t \in A} Q^*(s_t, a_t) \tag{26}$$

One of the challenges that reinforcement learning faces is the estimation of value functions, but the development of deep learning solves the problems. Deep reinforcement learning combines deep learning and reinforcement learning, which employs deep convolutional neural networks to approximate $Q(s_t, a_t)$. DDPG has been proposed in [12] to solve problems in continuous action spaces. As shown in Fig. 2, DDPG uses an actor-critic algorithm framework that reduces the difficulty of learning by separating the policy network (the actor function) and the Q-value network (the critic function). DDPG is a policy-base algorithm which parameterizes the policy and describes policy $\pi_\theta(s)$ with parametric linear or nonlinear function. In order to maximize cumulative returns $J(\pi_\theta) = \mathbb{E}_{T\sim\pi_\theta}\Big[R(T)\Big]$, where T represents the trajectory of one episode. According to [13], the policy gradient of the actor can be expressed as

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\rho^\mu}\Big[\nabla_\theta \mu_\theta(s) \nabla_a q^\mu(s, a)|_{a=\mu_\theta(s)}\Big] \tag{27}$$

Then we can update the network parameter with a learning rate α as:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\mu_\theta) \tag{28}$$

Actor network is used for the interaction with the environment and critic network is responsible for policy evaluation. During the learning process, actor network adjusts the parameter θ to control the action, and critic network guides the actor network to converge toward a larger cumulative return. The detailed steps are shown in Algorithm 1.

3.2 The DDPG Framework

In this section, we will employ the DDPG algorithm to minimize the overall cost of transmission and computation. Thus, the three elements of the DDPG algorithm can be defined as follows:

- System State: At the beginning of time slot t, the bits of user k's tasks $l_k(t)$ will be updated with the rate of poisson distribution λ_k. Meanwhile, channel

Algorithm 1. The DDPG-based Computational Offloading

Randomly initialize the actor network μ_{θ^μ} and the critic network Q_{θ^Q} with weights θ^μ and θ^Q;
Initialize target network μ and Q with weights $\theta^{\mu'} \leftarrow \theta^\mu$, $\theta^{Q'} \leftarrow \theta^Q$;
Initialize the experience replay buffer B;
for each episode $m = 1, 2, ..., M$ **do**
  Reset simulation parameters for the environment;
  Randomly generate an initial state s_1;
  for each time slot $t = 1, 2, ..., T$ **do**
    Select an action $a_t = \mu(s_t|\theta^\mu) + \nabla\mu$ to determine the power for transmission and computation;
    Execute action a_t and receive reward r_t and observe the next state s_{t+1};
    Store the tuple (s_t, a_t, r_t, s_{t+1}) into B;
    Sample a random mini-batch of N transitions (s_t, a_t, r_t, s_{t+1}) from B;
    Update the critic network by minimizing the loss L :
    $L = \frac{1}{N}\sum_{t=1}^{N}\left(r_t + \max_{a\in A} Q(s_t^{'}, a|\theta^{Q'}) - Q(s_t, a_t|\theta^Q)\right)^2$;
    Update the actor network by using the sampled policy gradient:
    $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_{t=1}^{N} \nabla_\alpha Q(s_t, a|\theta^Q)|_{a=a_t} \nabla_{\theta^\mu}\mu(s_t|\theta^\mu)$;
    Update the target networks by:
    $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$;
    $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$;
  end for
end for

vectors $\mathbf{g}_{k,q}(t)$ and $\mathbf{g}_{q,c}(t)$ will also be estimated. Thus the system state can be denoted as

$$s(t) = \left\{ l_k(t), \mathbf{g}_{k,q}(t), \mathbf{g}_{q,c}(t) \right\} \tag{29}$$

- System Action: In the system, the agent has to decide the allocated powers for local execution and transmission, MEC servers' computation and transmission. The weight of the bits of tasks computed in the MEC server will also be included. So the system action can be expressed as

$$a(t) = \left\{ p_k^l(t), p_{k,q}^o(t), p_{q,c}^o(t), p_{k,q}^{mec}(t), \alpha_q(t) \right\} \tag{30}$$

- Reward Function: We set the deadline t and assume that the transmission and computation must be accomplished within t. The overall cost of transmission and computation is regarded as the feedback of the system. In this way, the reward function $R(t)$ at time slot t can be written as

$$R(t) = -R_k^l * C^{user} - \sum_{m=1}^{l_k - R_k^l} P_k(m)(\alpha_q m C^{mec} + (1-\alpha_q) m C^{cloud}) - \omega(l_k - R_k^l - m) \tag{31}$$

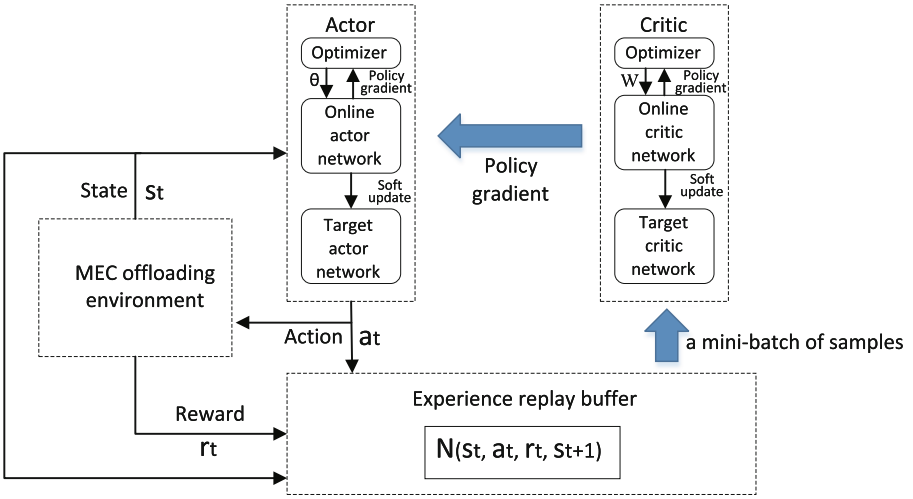

Fig. 2. The structure of DDPG

where ω is nonnegative weighted factor, and the agent will receive a negative reward if the transmission and computation are not completed within the deadline.

4 Numerical Results

In this section, we will present the numerical results to illustrate the performance of the proposed DDPG algorithm for power allocation. We assume that the bandwidth of the system is 1 MHz, the path loss factor α is 2. On the other hand, the maximum transmission and computation power, that is, the constraint of actions is 2 W. Additionally, the required CPU cycles per bit C_k and C_q is 500 cycles/bit, and $k = 10^{-27}$. Additionally, the cost of local computation C^{user} is 10, the cost of one task bit offloading including transmission and computation to MEC servers is 8, and the cost for one task bit transmitting to the core network is 15.

According to the results of the actual debugging experiment, the general training process of the deep reinforcement learning algorithm is summarized as follows: Firstly, the agent interacts with the environment with random actions, and explores the environment as much as possible. Then the agent chooses action based on a specific policy (obtained by the actor network or the greedy policy), and trains the agent according to the next state and reward that the environment feedbacks. Finally, the agent evaluates the performance of the learned policy and continues to train the network after evaluation. In order to receive the optimal policy, we set the learning rate of actor and critic as 0.0001 and 0.001 respectively. In Fig. 3, we show the training process of DDPG-based power allocation algorithm with $\lambda_{q,k} = 0.3$ and $\lambda_{q,k} = 2.5$. For different connect frequencies, the

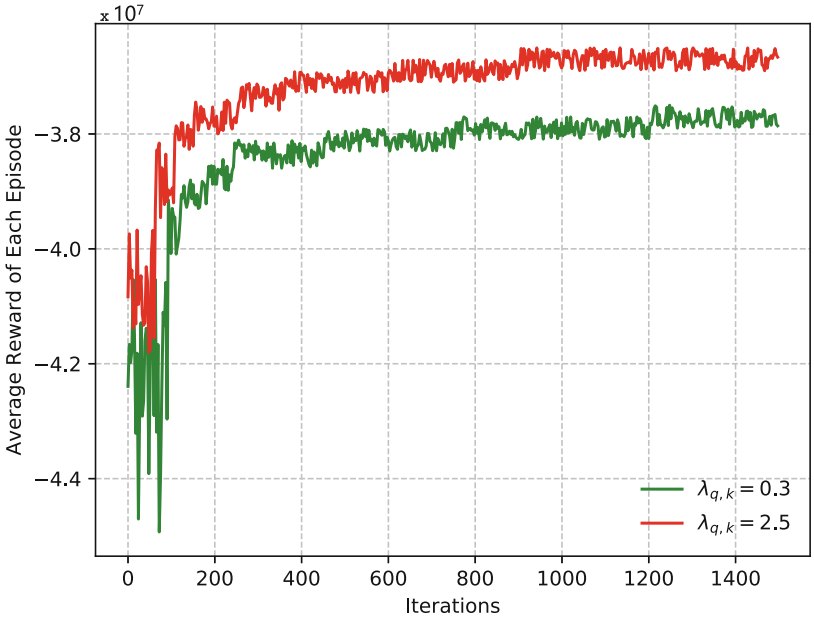

Fig. 3. Average reward of each episode for $\lambda_{q,k} = 0.3$ and $\lambda_{q,k} = 2.5$.

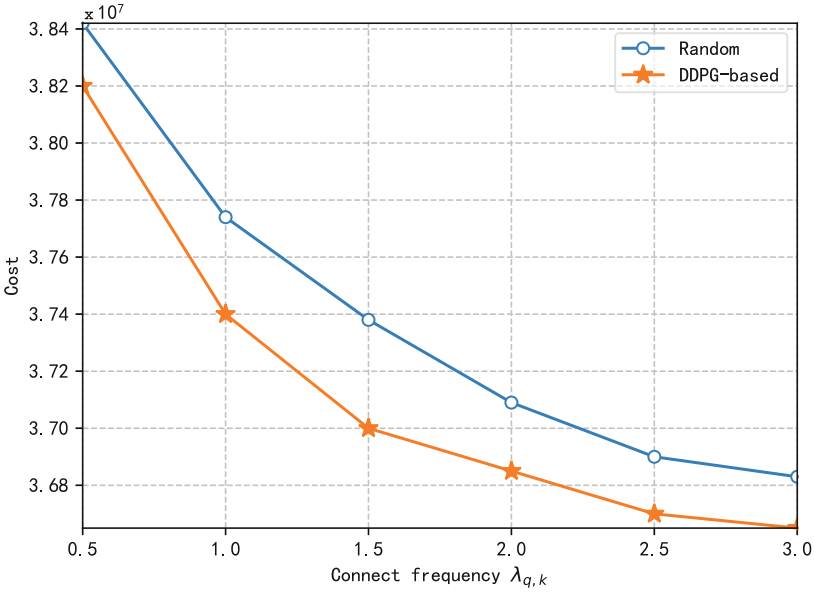

Fig. 4. Overall cost for DDPG-based and random power allocation with different connect frequency $\lambda_{q,k}$.

average reward both increases as the training time goes by, and reach a stable value. On the other hand, when the connect frequency is low, there are fewer tasks that the requested user can offload and then the overall cost increase.

We compare the performance of our proposed algorithm and random power allocation algorithm with different connect frequencies $\lambda_{q,k}$ in Fig. 4. The impact of connect frequency on the overall cost is shown in the figure. When the connect frequency increases, more tasks can be offloaded to MEC servers and the core network so the overall cost becomes lower. As expected, the DDPG-based algorithm presents a better power allocation scheme than random power allocation scheme.

It can be observed from Fig. 5 that when the task arrival rate increases, the overall cost will increase as well, which means that more tasks can be processed within the capabilities of the MEC offloading system. We can also find that the performance of our proposed DDPG-based algorithm acts better than random power allocation algorithm since the DDPG agent can adjust the power allocation and learn the optimal policy according to the current state after training.

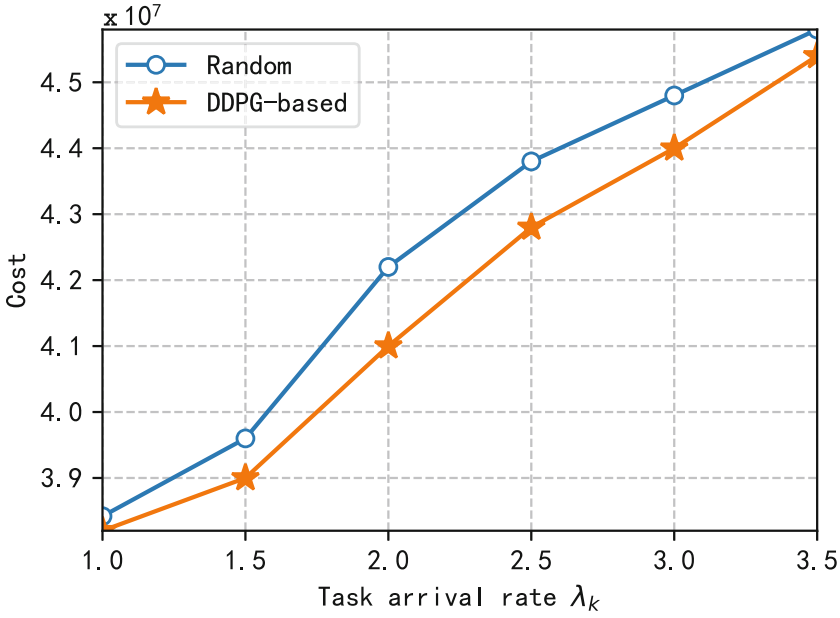

Fig. 5. Overall cost for DDPG-based and random power allocation with different task arrival rate λ_k.

5 Conclusion

In this paper, we developed a mobility-aware computation offloading system, where tasks can be processed locally and offloaded to MEC servers. By considering the limited resources at MEC servers, the unprocessed tasks can be handed over to the core network. Then we derive the offloading and computation rates under the condition of time-varying wireless channels and mobile users. We adopt the deep reinforcement learning algorithm named DDPG to learn the optimal power allocation policy with minimizing the overall consumption cost. Finally, it can be observed from numerical simulations that the DDPG-based algorithm presents a better power allocation scheme than existing method.

Acknowledgement. This work was supported in part by the National Science Foundation China under Grant 61771345 and Grant 61571330, and in part by the Fundamental Research Funds for the Central Universities.

References

1. Wang, Y., Chen, I.R., Wang, D.C.: A survey of mobile cloud computing applications: perspectives and challenges. Wirel. Pers. Commun. **80**(4), 1607–1623 (2015)
2. Yi, S., Li, C., Li, Q.: Mobile edge computing: a survey on architecture and computation offloading. IEEE Commun. Surv. Tutor. 1 (2017)
3. Ren, J., et al.: Exploiting mobile crowdsourcing for pervasive cloud services: challenges and solutions. IEEE Commun. Mag. **26**(1), 1–12 (2017)
4. Zhang, K., Mao, Y., Leng, S., He, Y., Zhang, Y.: Mobile-edge computing for vehicular networks: a promising network paradigm with predictive offloading. IEEE Commun. Mag. **12**(2), 36–44 (2017)
5. Liu, J., Mao, Y., Zhang, J., et al.: Delay-optimal computation task scheduling for mobile-edge computing systems. In: IEEE International Symposium on Information Theory, Barcelona, Spain, pp. 1451–1455 (2016). https://doi.org/10.1109/ISIT.2016.7541539
6. Zhang, K., Mao, Y., Leng, S., et al.: Optimal delay constrained offloading for vehicular edge computing networks. In: IEEE International Conference on Communications, Paris, France, pp. 1–6 (2017). https://doi.org/10.1109/ICC.2017.7997360
7. Du, J., Zhao, L., Feng, J., Chu, X.: Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee. IEEE Trans. Commun. **66**(4), 1594–1608 (2018)
8. Guo, H., Liu, J., Zhang, J., Sun, W., Kato, N.: Mobile-edge computation offloading for ultra-dense IoT networks. IEEE Internet Things J. **5**(6), 4977–4988 (2018)
9. Tan, L.T., Hu, R.Q.: Mobility-aware edge caching and computing in vehicle networks: a deep reinforcement learning. IEEE Trans. Veh. Technol. **67**(11), 10190–10203 (2018)
10. Chen, Z., Wang, X.: Decentralized computation offloading for multi-user mobile edge computing: a deep reinforcement learning approach. arXiv preprint arXiv: 1812.07394 (2015)
11. Ross, S.M.: Introduction to Probability Models, 11th edn. Academic Press, San Francisco (2000)

12. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning. arXiv preprint arXiv: 1509.02971 (2015)
13. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M.: Deterministic policy gradient algorithms. In: International Conference on Machine Learning, pp. 387–395 (2014)