Multi-agent Reinforcement Learning for Joint Wireless and Computational Resource Allocation in Mobile Edge Computing System

Yawen Zhang(✉), Weiwei Xia, Feng Yan, Huaqing Cheng, and Lianfeng Shen

National Mobile Communications Research Laboratory,
Southeast University, Nanjing 210096, China
{220170890,wwxia,feng.yan,220170869,lfshen}@seu.edu.cn

Abstract. Mobile edge computing (MEC) is a new paradigm to provide computing capabilities at the edge of pervasive radio access networks in close proximity to intelligent terminals. In this paper, a resource allocation strategy based on the variable learning rate multi-agent reinforcement learning (VLR-MARL) algorithm is proposed in the MEC system to maximize the long term utility of all intelligent terminals while ensuring the intelligent terminals' quality of service requirement. The novelty of this algorithm is that each agent only needs to maintain its own action value function so that the computationally expensive issue with the large action space can be avoided. Moreover, the learning rate is changed according to the expected payoff of the current strategy to speed up convergence and get the optimal solution. Simulation results show our algorithm performs better than other reinforcement learning algorithm both on the learning speed and users' long term utilities.

Keywords: Mobile edge computing · Joint resource allocation · Multi-agent reinforcement learning · Variable learning rate

1 Introduction

With the development of Internet, mobile intelligent terminals are becoming more and more popular and its function is more and more various. New applications such as face recognition, image recognition, automatic driving, video chat and augmented reality (AR) keep emerging [8]. However, these emerging applications require that mobile devices should have abundant computational resources and storage resources, while the resources of intelligent terminals are limited.

Mobile edge computing (MEC) has developed rapidly in recent years, as it provides a large amount of computational resources to bridge the gap between

This work was supported by the National Natural Science Foundation of China (No. 61741102, 61601122, U1805262).

J. Zheng et al. (Eds.): ADHOCNETS 2019, LNICST 306, pp. 149–161, 2019.
https://doi.org/10.1007/978-3-030-37262-0_12

the demands of applications and the restricted capacity of intelligent terminals. MEC servers are deployed at the base stations (BSs) in close proximity to mobile subscribers to execute latency-sensitive services, thereby extending computing, storage and data processing capabilities of intelligent terminals [5].

The joint computational and wireless resources allocation in mobile edge computing systems has been a key point attracting great interests in the MEC system in recent years [7]. To address the problem that the users in the MEC system typically suffer from unfair resource allocation, an approach is proposed in [17] to maximize the overall network throughput under the constraint of each user's minimum transmission rate. In [16], the offloading selection, radio resource and computational resource allocation are jointly optimized to minimize the energy consumption on smart mobile devices in a multi-mobile-users MEC system. In [9], three models, namely local compression, edge cloud compression and partial compression offloading are studied and compared. However, in their papers, the MEC servers need to know the global information.

Reinforcement Learning (RL) interacts with the environment and improves the behavior through trial and error to obtain the optimal solution. RL algorithm is a learning method that requires less prior knowledge and has been widely studied in the artificial intelligence community [1]. Therefore, there are also some literatures on the use of RL to solve the problem of resource allocation. In [14], an intelligent agent is designed to develop a real-time adaptive policy for computational resource allocation in order to improve the average end-to-end reliability by employing a deep reinforcement learning method. In [15], a dynamic offloading framework was formulated as a multi-label classification problem and the deep supervised learning method is developed to minimize the computational overhead. The Q-learning based and deep reinforcement learning based schemes are proposed respectively in [6] to tackle the resource allocation in wireless MEC system. However, the literatures above only study the situation after offloading without considering the necessity of offloading. Besides, Q-learning algorithm will cause the large state and action spaces, which leads to high computational complexity.

In this paper, variable learning rate multi-agent reinforcement learning (VLR-MARL) algorithm is proposed and the main contributions of this paper are as follows:

(1) Wireless and computation resources are jointly allocated in this paper in order to maximize the utility by increasing the throughput and reducing the cost of each user. Moreover, the necessity of offloading is also considered in this paper.
(2) Multi-agent reinforcement learning (MARL) method is used to reduce the learning time and speed up the search for the optimal strategy through multiple agents parallel processing.
(3) The learning rate is changed according to the expected payoff of the current policy and each agent only needs to maintain its own action value function, which reduces the complexity of this algorithm.

The rest of paper is organized as follows: in Sect. 2, the system model is described in detail. In Sect. 3, the VLR-MARL based resource allocation policy is proposed. Section 4 presents the simulation results. Finally, we conclude the paper in Sect. 5.

2 System Model

The MEC system studied in this paper is illustrated in Fig. 1, which includes a base station, MEC Servers and intelligent terminals. The intelligent terminals are connected to the BS through the wireless links and the MEC server is deployed inside the BS or connected to the BS through the optical fiber. The MEC server is a relatively small pool of resources with limited communication and computational resources.

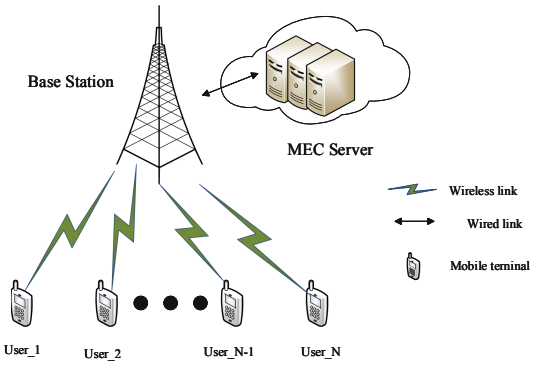

Fig. 1. System model.

In this paper, we consider that there are N users as $\mathcal{N} = \{1, 2, 3, \cdots, N\}$. Each user has computational intensive tasks that need to be offloaded to the MEC server. We can divide the wireless channel into K subcarriers, where $\mathcal{K} = \{1, 2, 3, \cdots, K\}$. Assuming that the subcarriers are orthogonal to each other, the users that choose different subcarriers do not interfere with each other. The connection between the user n and the subcarrier k is defined as c_n^k. When the user n utilizes the subcarrier k, $c_n^k = 1$, and $c_n^k = 0$ otherwise. Each user can only access to no more than one subchannel, which means

$$\sum_{k=1}^{K} c_n^k \leq 1, \forall n \in N \tag{1}$$

2.1 Communication Model

Since many users share the same channel, we need to take channel interference into account when transferring computing tasks to the MEC server through the

wireless channel. The uplink data rate of each user can be expressed as follows:

$$r_n = \omega \log_2 \left(1 + \frac{p_n \mathrm{c}_n^k g_{n,s}}{\sigma_0 + \sum\limits_{i \in N, i \neq n} p_i \mathrm{c}_i^k g_{i,s}} \right) \tag{2}$$

where ω and σ_0 represent channel bandwidth and background noise power, p_n is the transmission power of user n which is determined according to some power control algorithms such as [13] and [3]. Further, $g_{n,s}$ is the channel gain between user n and BS s, which is written as

$$g_{n,s} = l_{n,s}^{-\alpha} \tag{3}$$

where $l_{n,s}$ is the distance between user n and BS s and α is the path loss factor.

We assume that each user n has a computation-intensive task $J_n = \{b_n, d_n\}$. Here b_n denotes the size of input data (including task code and input parameters) and d_n denotes the numbers of CPU cycles required to complete the task J_n. A terminal can apply the methods in [4] to obtain the information of b_n and d_n. In order to offload the task to the MEC server, additional wireless transfer time is required as

$$t_n^{up} = b_n / r_n \tag{4}$$

The energy consumption generated during this period can be expressed as

$$e_n^c = p_n \frac{b_n}{r_n} + L_n \tag{5}$$

where L_n represents the energy consumption generated for a period of time after the completion of data uploading. After the MEC server receives the computational task, it will execute the task.

2.2 Computation Model

We assume that f_n^c is the computational resources (the number of CPU cycles per second) allocated to user n from the MEC server, so the task execution time for user n can be expressed as

$$t_n^{exe} = d_n / f_n^c \tag{6}$$

Therefore, the total overhead of user n can be given as

$$K_n^c = \lambda_n^t (t_n^{up} + t_n^{exe}) + \lambda_n^e e_n^c \tag{7}$$

The time it takes for the MEC server to transmit the results back to the user is negligible because the results are usually much smaller than the uplink data.

If the user does not upload the task to the MEC server, but decide to compute it locally, the cost only includes the computational time and energy consumption.

Let f_n^m be the computational capability of the user n. The local computational time of the task $J(n)$ can be given as

$$t_{n,local}^{exe} = d_n / f_n^m \tag{8}$$

Then the local energy consumption can be denoted as

$$e_n^{local} = \gamma_n d_n \tag{9}$$

where γ_n is the coefficient denoting the consumed energy per CPU cycle, which can be obtained by the measurement method in [12].

According to the Eqs. (8) and (9), then the local overhead can be expressed as

$$K_n^{local} = \lambda_n^t t_{n,local}^{exe} + \lambda_n^e e_n^{local} \tag{10}$$

where $\lambda_n^t, \lambda_n^e \in [0, 1]$ represents the weighting parameters of computational time and energy consumption for intelligent terminal n.

The utility function for each user should be related to the data transfer rate and resource overhead. Therefore, the utility function for the nth user can be given as:

$$u_n = \rho_i r_n - \upsilon_i(\lambda_n^t(t_n^{up} + t_n^{exe}) + \lambda_n^e e_n^c) \tag{11}$$

The first item in the above formula represents the data transmission rate provided to the user, and the second item represents the total cost incurred in offloading the task to the MEC server. ρ_i and υ_i are coefficients of both items.

2.3 Problem Formulation

The resource allocation in MEC system is formulated as an optimization problem. The objective of this paper is to maximize the utility function of all agents. When the cost of offloading is less than the cost of local computing, it can be considered that the resource allocation is reasonable and can meet users overhead requirement, otherwise the offloading is unreasonable. Each user can only choose one subcarrier. Under this constraint, the problem is formulated as

$$\max U = \sum_n u_n \tag{12}$$

$$s.t. K_n^{exe} < K_n^{local} \tag{13}$$

$$\sum_{k=1}^{K} c_n^k = 1, \forall n \in N \tag{14}$$

3 Multi-agent Reinforcement Learning Based Resource Allocation Algorithm

In this section, reinforcement learning is applied to solving the problem (12)–(14) to obtain the joint strategy of wireless and computational resource allocation. In the following, we first present the basic model of RL, then propose MARL to speed up the rate of convergence by parallel computation, and at last add variable learning rate to reduce the complexity of state and action space.

3.1 Basic Model

In the MEC network, each user acts as an agent. In each time slot, the agent chooses an action from action space. After applying an action, the agent receives a reward or punishment from the environment.

(1) State Space: The degree of satisfaction of user n can be defined as $s_n(t)$, so state space is expressed as

$$\mathcal{S}(t) = \{s_1(t), s_2(t), \ldots, s_N(t)\} \tag{15}$$

where $s_n(t) = \{0, 1\}$, when $s_n(t) = 0$, it means that the offloading cost of the user is more expensive than the local computation. On the contrary, if it is $s_n(t) = 1$, it means that the offloading is reasonable and the user meets the needs of overhead costs.

(2) Action Space: Each user selects computational and wireless resources, so the action space can be represented as:

$$a_n(t) = \{b_n(t), c_n(t)\} \tag{16}$$

where $c_n(t)$ represents computational resources obtained from the server and $b_n(t)$ represents wireless resources. The computational resources provided by the server have been decentralized as $\{4000, 4500, 5000, 5500\}$, therefore $c_n(t)$ represents the computing resource arbitrarily selected on behalf of the user. Besides, $b_n(t) = \left\{b_n^1(t), b_n^2(t), ..., b_n^K(t)\right\}$, $b_n^k(t) = 1$ represents that user n chooses subchannel k.

(3) Reward: When the user chooses the action $a_n(t)$ by observing the state $s_n(t)$, it will obtain an immediate reward $r_n(t)$ as

$$r_n(t) = u_n(t) - \Phi \tag{17}$$

where $\Phi > 0$ is a fixed cost when the user chooses an action. Multiple agents collectively explore the environment and refine wireless and computational resource allocation based on their own observations of the environment state. While the resource allocation problem may appear a competitive game, in order to make all agents cooperate with each other, the same reward is used for all agents. As such, the reward includes the instantaneous utilities of all intelligence terminals. In the meantime, to achieve the objective that all users offload successfully as much as possible, the reward is set to add a constant number Ω, that is greater than the largest utility. The reward of the user at each time step t is defined as

$$R_n(t) = \begin{cases} \sum\limits_n r_n(t), if \sum\limits_n s_n(t) < N \\ \sum\limits_n r_n(t) + \Omega, f \sum\limits_n s_n(t) = N \end{cases} \tag{18}$$

3.2 Multi-agent Reinforcement Learning Policy

Reinforcement learning often has the characteristics of delay in return, so a function is defined to indicate the long-term influence on the strategy chosen in its current state. This function is called the value function [10]. The expression of value function is

$$V^{\pi}(s) = \mathbb{E}_{\pi}[\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s] \tag{19}$$

where $\mathbb{E}(x)$ denotes the expectation of x. The above formula represents a cumulative expectation of the reward value obtained by the strategy in the initial state s, and γ^i is the discount factor to measure the importance of the reward in the value function. In general, the further away from the current state, the smaller the effect of the reward.

The current value function can be estimated by the value function of the subsequent state and the formula can be obtained as

$$V^{\pi}(s) = \pi(s) \sum_{s' \in S} p(s, s')[r_0 + \gamma V^{\pi}(s')] \tag{20}$$

where $p(s, s')$ represents the state transition probability from state s to state s' and $\pi(s)$ denotes the policy of agent i at state s. All subsequent states can be obtained from the model formula $\pi(s)$ and action set. When these conditions are unknown, the subsequent state can be obtained only by trials and sampling.

A policy usually corresponds to multiple execution actions. The value function can be decomposed into the expression related to each action, and the action value function can be obtained as

$$Q(s, a) = r_s^a + \gamma \sum_{s \in S} p(s, s') \sum_{a, \in A} \pi(a|s') Q(s', a') \tag{21}$$

where r_s^a denotes the reward when agent i chooses action a at state s, $p(s, s')$ represents the state transition probability from state s to state s' and $\pi(a|s')$ denotes the policy of agent i at state s' choosing action a.

In order not to get stuck in a locally optimal solution, we adopt $\varepsilon - greedy$ strategies [2] to explore the environment. ε is a very small number, as the probability of picking a random action and $1 - \varepsilon$ denotes the probability value of selecting the optimal action.

$$\pi(a|s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A(s)|}, a = \arg\max Q(s, a) \\ \frac{\varepsilon}{|A(s)|}, a \neq \arg\max Q(s, a) \end{cases} \tag{22}$$

One popular reinforcement learning method is Q-learning [11]. In Q-learning, the optimal Q-value function can be obtained from the Bellman's equation. The Q-value of the current state can be calculated using the Q-value of the next state. There is a difference between the calculated Q-value and the original value in

this state, which is called incremental Q-value. Then, we can use it to update the Q-value, which is given as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right) \tag{23}$$

where α represents the learning rate and usually $\alpha = 0.1$, $\max_{a'} Q(s',a')$ represents the maximum Q-value of all possible actions in the next state.

In this paper, all users are assumed to be agents. All the actions of agents constitute the joint action space set $A = a_1 \times a_2 \times \ldots a_n$. The return function of each Agent is calculated based on the joint action space and all agents' policies are combined into joint strategy h. The Q-value of each Agent in state s is

$$Q_n^h(s,a) = \mathbb{E}\{\sum_{k=0}^{\infty} \gamma^k r_{i,k+1} | s_0 = s, a_0 = a, h\} \tag{24}$$

This Q-value iteration algorithm is guaranteed to converge to the optimal point with $Q(s,a) \rightarrow Q^*(s,a)$ as $iteration \rightarrow \infty$ [10]. Because Q-learning is off-policy, which means the agent learning and interacting with the environment are different. The behavior policy is used to interact with the environment to generate datas in order to make decisions in the training process. The target policy is constantly studying and optimizing using the datas generated by the behavior policy. The target policy is greedy while choosing behavior uses $\varepsilon - greedy$ strategy. The maximum action value is used to calculate the expected return of the next state, however the current policy does not always select the optimal action. The strength of the off-policy is that by separating the target policy from the behavior policy, the global optimal value can be obtained while maintaining exploration.

3.3 Variable Learning Rate-MARL Algorithm

To reduce the space complexity of multi-agent problem and improve convergence, we propose a VLR-MARL algorithm. The learning rate changes by the principle that learn fast while losing and learn slowly while winning. The principle helps convergence by giving other agents more time to adapt to strategy changes that initially seem beneficial, while allowing agents to adapt more quickly to other agents' harmful strategy changes. This algorithm requires two learning parameters δ_{lose} and δ_{win}. If the agent is currently determined to be winning, δ_{win} is used, otherwise, δ_{lose} is used. It is based on virtual game, and replaces the unknown equilibrium strategy with the approximate equilibrium average greedy strategy.

The agent will explore action a_k when it transfers from stage s_k to s_{k+1} and has the reward function R. Then its average estimation strategy update can be expressed as:

$$\bar{\pi}_i(s_k,a_k) = \bar{\pi}_i(s_k,a_k) + \frac{1}{C(s)}[\pi_i(s_k,a_k) - \bar{\pi}_i(s_k,a_k)] \tag{25}$$

where $C(s)$ denotes the number of occurrences of state s. The strategy can be given as

$$\pi_i(s_k, a_k) = \pi_i(s_k, a_k) + \Delta_{sa_i} \tag{26}$$

$$\Delta_{sa_i} = \begin{cases} -\delta_{sa_i} a_i \neq \arg\max_{a'} Q(s, a') \\ \sum_{a' \neq a_i} \delta_{sa'}, others \end{cases} \tag{27}$$

The learning rate that is used to upgrade the policy depends on whether the agent is currently determined to be winning or losing. This is determined by comparing whether the current expected value is greater than the current expected value of the average strategy. If the expectation of the current policy is smaller, then the larger learning rate δ_{lose} is used.

$$\delta_{sa_i} = \min(\pi(s_k, a_k), \frac{\delta}{|A_i| - 1}) \tag{28}$$

$$\delta = \begin{cases} \delta_{win}, \sum_{a_i \in A_i} \pi_i(s_k, a_k) Q_i > \sum_{a_i \in A_i} \bar{\pi}_i(s_k, a_k) Q_i \\ \delta_{lose}, others \end{cases} \tag{29}$$

The detailed procedure of the proposed VLR-MARL algorithm is presented in Algorithm 1.

4 Performance Evaluation

In this section, we evaluate the performance of our VLR-MARL algorithm and provide the numerical results compared with other algorithms.

In this simulation, we suppose that there are 20, 30 or 40 users within a radius of 100 m, randomly distributed around the base station. The total number of channels is $K = 5$ with channel bandwidth $W = 5$ MHz. The user's transmitted power is 100 mW. The path loss coefficient of the channel gain is $\alpha = 4$ and the background noise power σ_0 is -100 dBm. The size of all users' input data b_n for the computation task J_n is 5000 kB and the number of CPU cycles d_n required to complete the task is 1000 Mc. The available computational resources in MEC servers is $f_n^c = \{5000, 8000, 10000, 12000\}$ M/s. The local computational capability of each user is 500 M/s.

4.1 Convergence of Proposed Algorithm

The convergence of the proposed VLR-MARL algorithm is shown in Fig. 2. The initial action is chosen randomly and the stable state is achieved within 20 iterations. Figure 2 also shows the performance with different learning rate δ_{win} and δ_{lose}. When the $\frac{\delta_{win}}{\delta_{lose}} = \frac{1}{2}$, VLR-MARL algorithm reaches the optimal utility a bit faster. Therefore, the learning rate is chosen to be $\delta_{win} = 0.005$ and $\delta_{lose} = 0.0025$ in the next simulation.

Figure 3 shows the offloading success rate of different numbers of users. The state function of the users shows the offloading situation. When $s_n(t) = 1$, the task offloading is known as successful. When the number of users increases, the rate of convergence will slow down.

Algorithm 1: VLR-MARL Algorithm for Resource Allocation in MEC Network

Input: The number of populations $\mathcal{N}$, the distance d between each user and MEC Server, Max_iteration, the number of wireless channel $\mathcal{K}$

Output: Optimal sequence of actions required to maximize the users' utility

Initialize: $Q_i(s,a_k)=0$, $\pi_i(s,a_k)=\frac{1}{|A_i|}$, $\bar{\pi}_i(s,a_k)=\frac{1}{|A_i|}$, δ_{lose}, δ_{win}, $C(s)=0$, $\alpha,\gamma,\varepsilon\in[0,1]$; **for** *iteration* $=1$ *to Max_iteration* **do**

  for *user*$_i$ *to user*$_N$ **do**

    user$_i$ take $\varepsilon-greedy$ policy to choose action a_k based on the current state s

    Calculate the next reward value r_i

    Calculate the next state s'

    Update $Q_i(s,a_k)$

    $$:Q_i(s,a_k)\leftarrow Q_i(s,a_k)+\alpha\left(r+\gamma\max_{a'}Q(s',a')-Q_i(s,a_k)\right)$$

    Update $C(s)=C(s)+1$

    Update the average estimation strategy $\bar{\pi}_i(s,a_k)$

    $$=\bar{\pi}_i(s_k,a_k)+\frac{1}{C(s)}[\pi_i(s_k,a_k)-\bar{\pi}_i(s_k,a_k)]$$

    Determine whether the agent is wining and choose

    $$\delta=\begin{cases}\delta_{win}, \sum\limits_{a_i\in A_i}\pi_i(s_k,a_k)Q_i>\sum\limits_{a_i\in A_i}\bar{\pi}_i(s_k,a_k)Q_i\\ \delta_{lose}, others\end{cases}$$

    Update the strategy function $\pi_i(s,a_k)$

  iteration $=$ *iteration* $+1$

Return the action space of all users

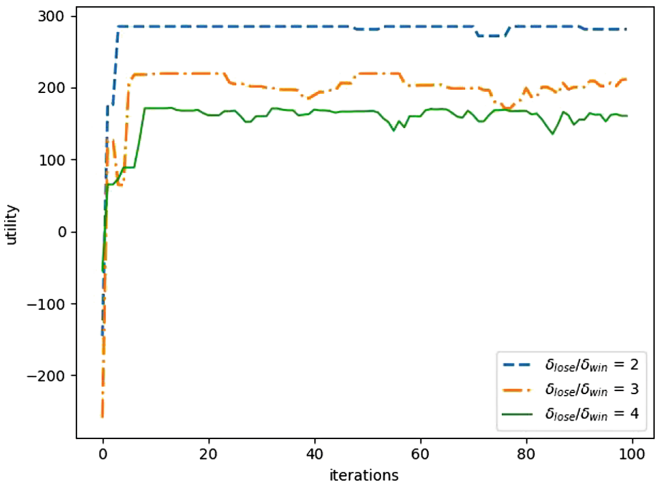

Fig. 2. Utility of users

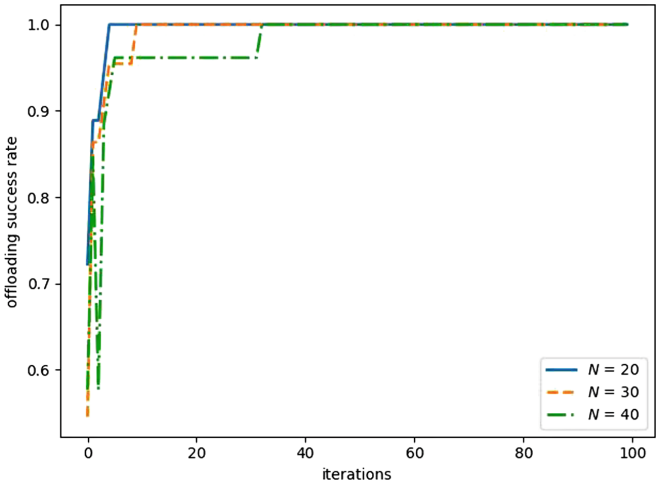

Fig. 3. Offloading success rate

4.2 Comparison to Other Algorithms

Then, the convergence efficiency are analyzed with various MARL algorithms. Figure 4 shows the learning curves of centralized single agent Q-learning algorithms in [6], Q-learning MARL algorithms and VLR-MARL algorithms proposed in this paper. As can be seen, Q-learning reaches the optimal solution more slowly than the proposed VRL-MARL algorithm. Moreover, the utility of Q-learning single agent reinforcement learning (SARL) and Q-learning-MARL are lower than VRL-MARL. Therefore, the VRL-MARL algorithm performs better both on the learning speed and users' utilities.

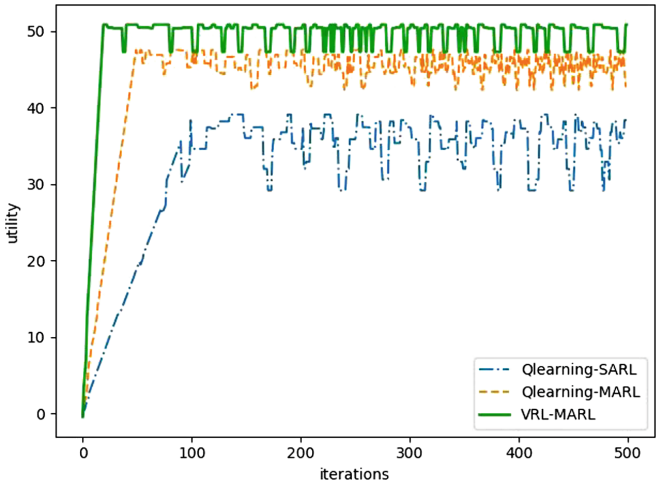

Fig. 4. Learning curves of different algorithms

5 Conclusion

In this paper, we propose a MARL framework to obtain the optimal resource allocation strategy in the MEC system. The optimization issue has been designed to obtain the maximum long-term reward while guaranteeing that users' offloading is reasonable. Considering the non-convex and combinatorial characteristics of this joint optimization problem, we have proposed the VLR-MARL strategy by jointly choosing channels and allocating computational resource to users. Based on the theory that learn slowly while winning and learn fast while losing, our strategy can efficiently provide a near-optimal solution with a small amount of iterations. Simulation results are given to indicate the convergence of the proposal method and better performance compared with other reinforcement learning methods.

References

1. Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: A brief survey of deep reinforcement learning. arXiv preprint arXiv:1708.05866 (2017)
2. Bu, L., Babu, R., De Schutter, B., et al.: A comprehensive survey of multiagent reinforcement learning. IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.) **38**(2), 156–172 (2008)
3. Chiang, M., Hande, P., Lan, T., Tan, C.W., et al.: Power control in wireless cellular networks. Found. Trends® Networking **2**(4), 381–533 (2008)
4. Cuervo, E., et al.: MAUI: making smartphones last longer with code offload. In: Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, pp. 49–62. ACM (2010)
5. Lan, Z., et al.: A hierarchical game for joint wireless and cloud resource allocation in mobile edge computing system. In: 10th International Conference on Wireless Communications and Signal Processing (WCSP), pp. 1–7. IEEE (2018)
6. Li, J., Gao, H., Lv, T., Lu, Y.: Deep reinforcement learning based computation offloading and resource allocation for MEC. In: IEEE Wireless Communications and Networking Conference (WCNC), pp. 1–6. IEEE (2018)
7. Mach, P., Becvar, Z.: Mobile edge computing: a survey on architecture and computation offloading. IEEE Commun. Surv. Tutorials **19**(3), 1628–1656 (2017)
8. Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K.B.: A survey on mobile edge computing: the communication perspective. IEEE Commun. Surv. Tutorials **19**(4), 2322–2358 (2017)
9. Ren, J., Yu, G., Cai, Y., He, Y.: Latency optimization for resource allocation in mobile-edge computation offloading. IEEE Trans. Wireless Commun. **17**(8), 5506–5519 (2018)
10. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction (2011)
11. Watkins, C.J., Dayan, P.: Q-learning. Mach. Learn. **8**(3–4), 279–292 (1992)
12. Wen, Y., Zhang, W., Luo, H.: Energy-optimal mobile application execution: taming resource-poor mobile devices with cloud clones. In: Proceedings IEEE Infocom, pp. 2716–2720. IEEE (2012)
13. Xiao, M., Shroff, N.B., Chong, E.K.P.: A utility-based power-control scheme in wireless cellular systems. IEEE/ACM Trans. Networking **11**(2), 210–221 (2003)

14. Yang, T., Hu, Y., Gursoy, M.C., Schmeink, A., Mathar, R.: Deep reinforcement learning based resource allocation in low latency edge computing networks. In: 15th International Symposium on Wireless Communication Systems (ISWCS), pp. 1–5. IEEE (2018)
15. Yu, S., Wang, X., Langar, R.: Computation offloading for mobile edge computing: a deep learning approach. In: IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), pp. 1–6. IEEE (2017)
16. Zhao, P., Tian, H., Qin, C., Nie, G.: Energy-saving offloading by jointly allocating radio and computational resources for mobile edge computing. IEEE Access **5**, 11255–11268 (2017)
17. Zhu, Z., et al.: Fair resource allocation for system throughput maximization in mobile edge computing. IEEE Access **6**, 5332–5340 (2018)