Decentralized Computation Offloading and Resource Allocation in MEC by Deep Reinforcement Learning

Yeteng Liang, Yejun He*, Xiaoxu Zhong
Guangdong Engineering Research Center of Base Station Antennas and Propagation
Shenzhen Key Laboratory of Antennas and Propagation
College of Electronics and Information Engineering, Shenzhen University, 518060, China
Email: 1430396120@qq.com, heyejun@126.com*, 1031772642@qq.com

***Abstract*—Mobile edge computing (MEC) as a promising technology to relieve edge user equipment (UE) computing pressure by offloading part of a task, is able to reduce the execution delay and energy consumption effectively, and improve the quality of computation experience for mobile users. Nevertheless, we are facing a challenge of design of computation offloading and resource allocation strategy on a part of a task offloaded to MEC server. A task is divided into two sub-tasks firstly. Then one of the two sub-tasks is executed locally, and the other will be offloaded to MEC server that is located near the base station (BS). Based on dynamic offloading and resource allocation strategy, the best offloading proportion of a task, local calculation power and transmission power are investigated by deep reinforcement learning (DRL). In this paper, we propose two DRL-based approaches, which are named as deep Q network (DQN) and deep deterministic policy gradient (DDPG), to minimize the weighted sum cost including execution delay and energy consumption of UE. DQN and DDPG can deal with large scale state spaces and learn efficient offloading proportion of task and power allocation independently at each UE. Simulation results demonstrate that each UE can learn the effective execution policies, and the proposed schemes achieve a significant reduction in the sum cost of task compared with other traditional baselines.**

***Index Terms*—MEC, offloading proportion, power allocation, deep deterministic policy gradient (DDPG), deep Q network (DQN)**

I. Introduction

Mobile Edge computing (MEC) supports computationally intensive applications in mobile devices by offloading processing and storage to remote edge servers over the wireless networks. Due to limited computation resources, edge UEs (user equipments) can not complete some calculations on their own. Thus UEs have to offload part or all of a task to cloud computing server for calculation to reduce the execution delay and energy consumption. However, confronting overcrowding big data, the core network suffers from fatal congestion and execution delay, greatly reducing quality of service (QoS). To solve the above problems effectively, the technology of MEC was firstly proposed by the European Telecommunication Standardization Association (ETSI). MEC can provide cloud-based IT service environment, equipment management, computing and storage capabilities and other functions in the wireless access network close to the edge UEs [1].

In recent years, the optimization of the task offloading strategy becomes one of research hot-spots in MEC. The task offloading strategy of UE is mainly divided into binary offloading and partial offloading. The former means that all of a task of a UE are either processed locally or offloaded to MEC server for processing. The latter refers to that all of a task can be divided randomly or according to certain regulations, i.e., a part of which is left to be calculated locally while the other is offloaded to MEC server to execute.

References [2], [3] and [4] focused on the partial offloading strategy. Reference [2] proposed an optimal adaptive algorithm based on joint optimization method and [3] utilized dynamic voltage frequency scaling (DVFS) technique to minimize energy consumption. However, both are based on the single UE MEC system. Reference [4] proposed two traditional optimization algorithms called sequential quadratic programming (SQP) and deep neural network (DNN) algorithm to minimize energy consumption. However, the goal of these works is to minimize the energy consumption of UE, without considering the task execution delay, simultaneously.

References [5] and [6] adopted the trade-off between execution delay and energy consumption of task to solve the above problems. For the optimization framework of single UE MEC system, [5] proposed an offloading decision algorithm based on linear programming relaxation (LPR) and semi-definite relaxation (SDR) algorithm, where the performance of execution delay and energy consumption has been greatly improved. However, the UE's calculation frequency is still constant. In order to determine the calculation offloading strategy, the optimal frequency of local computation and optimal transmission power, [6] developed an online algorithm based on the Lyapunov optimization algorithm [7], which achieved optimal sum cost of task. However, the algorithm is not suitable for complex and real-time MEC system.

To this end, we propose two deep Q network algorithms based on the principle of deep reinforcement learning and partial offloading. Each user can learn effective dynamic offloading strategies and power allocation to get the minimum the sum cost of task for multi UEs scenario. The major contributions of this paper are as follows.

- In our MEC system, each UE can independently make decisions consisting of offloading proportion of a task and power allocation based on time-varying wireless channel to minimize the sum cost.
- Based on DQN and DDPG, two DRL frameworks for computation offloading have been designed, and each UE is an agent that can learn the dynamic computation offloading and resource allocation policies according to each UE's local observation.
- The numerical results show the sum cost of the proposed algorithms is the lowest compared to other baselines under a reasonable running time of making a policy.

The rest of this paper is organized as follows. In Section II, the system model and the required parameter variables are introduced. In Section III, the optimization problem and two algorithms based on DDPG and DQN are formulated. Simulation results are publicized in Section IV. Finally, we conclude this paper in Section V.

II. System Model

A. Network Model

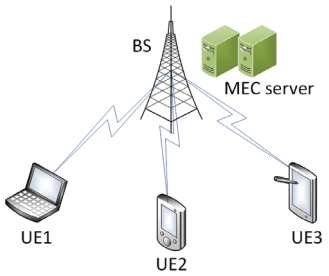

Fig. 1. A typical communication model of MEC system.

Fig. 1 shows a typical communication model of MEC system, which consists of one BS with N antennas and M UEs. The set of UE is denoted by $U = \{1, 2, ..., M\}$, and assuming that each UE m has one D_m-size task and the completed task in each step is divided into two parts. The α_m represents the offloading proportion of task for each UE m. The transmission power and local calculation power are denoted by $P_{o,m}$, $P_{l,m}$, respectively. B is defined as the bandwidth of each UE. The calculation frequency allocated to each UE by MEC server is a constant, which is denoted by f_s.

B. Computation Model

I) Local Computation Model: If local execution frequency of UE m is defined as $f_{l,m}$, the local execution delay is given by:

$$T_{l,m} = \frac{L(1-\alpha_m)D_m}{f_{l,m}} \tag{1}$$

$$f_{l,m} = \sqrt[3]{\frac{P_{l,m}}{\kappa}} \tag{2}$$

where L represents the number of computing cycles required to calculate one bit data, and κ is the effective switched capacitance depending on the chip architecture. The energy consumption of local execution for UE m can be expressed as:

$$E_{l,m} = P_{l,m}T_{l,m} \tag{3}$$

The total cost of local computation model for UE m is given by:

$$C_{l,m} = T_{l,m} + E_{l,m} \tag{4}$$

II) Offloading Computation Model: Partial task is offloaded to MEC server by BS, then MEC server executes task and returns a result. We ignore the return delay since the size of computation output is small compared with its input. The transmission rate can be expressed as:

$$R_m = B\log_2(1 + P_{o,m} * IPN_m) \tag{5}$$

$$IPN_m = \frac{1}{\sigma_R^2\left[\left(\boldsymbol{H}^H(t)\boldsymbol{H}(t)\right)^{-1}\right]_{mm}} \tag{6}$$

where IPN_m denotes the interference-plus-noise of UE m [8], the $N \times M$ channel matrix between M UEs and BS is defined as $\boldsymbol{H}(t) = [\boldsymbol{h}_1(t), \ldots, \boldsymbol{h}_M(t)]$, [M]mm denotes as the (m, m)-th element of matrix M, and $\boldsymbol{h}_m(t)$ represents the channel vector of UE m.

The total delay of offloading execution includes transmission delay and computation delay in MEC server, which is denoted by:

$$T_{o,m} = \frac{\alpha_m D_m}{R_m} + \frac{L\alpha_m D_m}{f_s} \tag{7}$$

The energy consumption of offloading execution for UE m can be written as:

$$E_{o,m} = P_{o,m}\frac{\alpha_m D_m}{R_m} \tag{8}$$

The total cost of offloading computation model is given by:

$$C_{o,m} = T_{o,m} + E_{o,m} \tag{9}$$

Thus, the battery energy B^t of UE m can be obtained by:

$$B_m^{t+1} = \min\{\max\{B_m^t - (E_{l,m}+E_{o,m}) + e_m^t, 0\}, B_m^{max}\} \tag{10}$$

where B_m^{max} represents the maximum battery capacity of UE m, and e_m^t denotes the energy gain of UE m from surrounding environment in step t, including solar radiation and electrified wire netting.

Since the local and offloading executions are done simultaneously, the total execution delay cost should be the maximum delay among the local computation and offloading computation. Combining the above formulas, the sum cost of

executing one task for UE m in MEC system can be expressed as:

$$C_m^{sum} = \omega max\{T_{l,m}, T_{o,m}\} + (1-\omega)(E_{l,m} + E_{o,m}) + \delta_m^t \quad (11)$$

where $\omega \in [0,1]$ denotes the weighted parameter of execution delay for UE m. The δ_m^t denotes the penalty of the failed execution for the task.

III. Problem Formulation and Proposed Approaches

In this section, we formulate the execution issue of task as minimizing the weighted sum cost of UE by optimizing the task offloading and resource allocation policy. Since the formulation is non-convex, it is hard to deal with the issue by conventional algorithms. Thus we simplify the issue and propose two near-optimal task offloading and resource allocation algorithms based on DQN and DDPG.

A. Problem Formulation

The optimization problem is formulated as:

$$\begin{aligned} \mathcal{P}_1 : & \min_{\alpha_m, P_{l,m}, P_{o,m}} C_m^{sum} \\ s.t.\ & C1 : \alpha_m,\ \omega \in [0,1] \\ & C2 : 0 \le P_{l,m},\ P_{o,m} \le P_m^{max} \\ & C3 : 0 \le (E_{l,m} + E_{o,m}) \le B_m^{max} \end{aligned} \quad (12)$$

where $C2$ indicates that the local execution power and the transmission power can not exceed the maximum power of P_m^{max}. In order to minimize the task maximum execution delay for UE m, we simplify (12). Since the local execution and offloading execution are simultaneously performed, we have:

$$T_{l,m} = T_{o,m} \quad (13)$$

where (13) indicates that when the local execution delay is equal to the offloading execution delay, the $\max\{T_{l,m}, T_{o,m}\}$ can obtain the minimum value. Furthermore, the relationship between the optimal offloading proportion of task and the optimal local execution frequency as well as the optimal transmission rate can be written as:

$$\alpha_m = \frac{LR_m f_s}{LR_m f_s + LR_m f_{l,m} + f_{l,m} f_s} \quad (14)$$

Thus, $P1$ is revised as:

$$\begin{aligned} \mathcal{P}_2 : & \min_{P_{l,m}, P_{o,m}} C_m^{sum} = \min_{P_{l,m}, P_{o,m}} \{\omega \max\{T_{l,m}, T_{o,m}\} \\ & + (1-\omega)(E_{l,m} + E_{o,m}) + \delta_{t,m}\} \\ s.t.\ & \omega \in [0,1],\ C2,\ C3 \end{aligned} \quad (15)$$

Problem 2 can be solved by finding out the optimal local execution power and the optimal transmission power.

B. The Deep Reinforcement Learning Framework

Three key elements of DRL are as follows.

State: Each UE is an agent that can make decisions independently after training. We presume that the state of UE is only determined by collecting local observation of MEC system. Before forming a new state, the UE m will receive the last IPN_m from BS in step t, i.e., $IPN_m(t-1)$, UE m channel vector $h_m(t)$ of coming uplink transmission can be estimated by channel correlation. Thus the state of UE m can be written as $s_{m,t} = [IPN_m(t-1), h_m(t)]$.

Action: The action of UE m includes $P_{l,m}$ and $P_{o,m}$ which can be selected according to current state $s_{m,t}$. As a result, the action of UE m is expressed as $a_{m,t} = [P_{l,m}, P_{o,m}]$.

Reward: The UE m will get a reward $r(s_{m,t}, a_{m,t})$ after executing one step where the UE m selects certain action $a_{m,t}$ to execute. In general, the reward is related to the objective function. In order to learn the near-optimal task offloading and resource allocation policy in our model, we refer to the minimizing the weighted sum cost as the reward. Thus, the reward of each step t can be denoted as $r_{m,t} = C_m^{sum}$.

C. Deep Q Network Approach

The DQN algorithm is originated from the Q-learning algorithm which is a classical reinforcement algorithm. The agent needs to calculate and store a Q-value in a Q-table in executing each step for Q-learning, and each state-action pair will have a corresponding Q-value. Thus the relationship between reward $r(s,a)$ and Q-value can be expressed as $Q(s,a) = r(s,a) + \gamma * \max Q(s',a')$. The difference between DQN algorithm and Q-Learning is that DQN's Q-value is calculated not directly by the state-action pair but by a neural network. Since the DQN algorithm does not have a Q-table to store current all Q-values, the agent needs to use experience replay method to update the action-value function. The pseudo code of the task offloading and recourse allocation algorithm based on DQN is shown in Algorithm 1.

D. Deep Deterministic Policy Gradient approach

Although DQN algorithm can solve the problem of high dimensional state spaces, it is difficult to deal with high dimensional and continuous action spaces. Thus we adopt the DDPG to extend DRL algorithm to continuous action spaces. Although DDPG algorithm borrows the idea of experience replay and target network of DQN, it cannot directly adopt the Q-learning framework since the greedy strategy of Q-learning cannot be implemented simply and quickly in the continuous action space. Therefore, the actor-critic algorithm framework based on determining the action strategy is used by DDPG. Different from DQN algorithm, which directly copies parameters of the Q network to the target network periodically, namely hard updates, DDPG adopts soft target updates to ensure that the parameters can be updated slowly and improve learning stability. The pseudo code of the task offloading and recourse allocation algorithm based on DDPG is shown in Algorithm 2.

Algorithm 1 The Task Offloading And Recourse Allocation Algorithm based on DQN

for each UE $m \in M$ do
  Initialize experience replay buffer D_m as C
  Initialize action-value function Q and target action-value function Q' with random weights θ_m, θ'_m
  for episode $e = 1,\ 2,\ 3, \cdots,\ E$ do
    Randomly initialize $s_{m,1}$
    for step $t = 1,\ 2,\ 3, \cdots, T$ do
      Select an action $a_{m,t}$ with probability ε
      otherwise $a_{m,t} = \arg max_a Q(s_{m,t}, a_m; \theta_m)$
      Execute action $a_{m,t}$ and then calculate reward $r_{m,t}$ and observe next state $s_{m,t+1}$
      Store $(s_{m,t}, a_{m,t}, r_{m,t}, s_{m,t+1})$ in D_m
      Sample a random mini-batch of $(s_{m,j}, a_{m,j}, r_{m,j}, s_{m,j+1})$ from D_m
      Set $y_{m,j} =$
      $\begin{cases} r_{m,j} & \text{for t=T} \\ r_{m,j} + \gamma_m \max_{a'} Q'(s_{m,j+1}, a'_m; \theta'_m) & \text{otherwise} \end{cases}$
      Perform gradient descent on $(y_{m,j} - Q(s_{m,j}, a_{m,j}; \theta_m))^2$ with respect to θ_m
      Update $\theta'_m = \theta_m$
    end for
  end for
end for

IV. Simulation Results

In this section, the performance of our proposed task offloading and resources allocation algorithm is presented by Python software. For comparison, we provide four baselines as well: "Full Local" means that all of a task for UE is executed in locally. "Full Offload" means that all of a task for UE is offloaded to MEC server to execute. "Random Offload" represents that the $P_{l,m}$ and $P_{o,m}$ are random. "SA" means that each UE obtain the optimal action to minimize the weighted sum cost by adopting simulation annealing algorithm in each step.

A. Parameters Setup

The energy harvesting follows the Poisson distribution. In addition, $B = 3$ MHz, $f_s = 2$ GHz, $L = 500$, and $P_m^{max} = 2$ W.

For DQN, we adopt the neural network with four layers namely input layer, output layer and two hidden layers, and there are 400 and 300 neurons in the two hidden layers, respectively. We assume the capacity of $C = 100000$, the mini-batch size is set as 60, and the learning rate is $\lambda = 0.001$.

For DDPG, the fully connected network with four layers including input layer, output layer and two hidden layers are adopted in the actor network and critic network. 400 and 300 neurons are selected in the two hidden layers, respectively. The activation functions of two hidden layers utilize the ReLu. The Sigmoid is used in the output layer to bound the actions. Furthermore, the soft update rate is $\tau = 0.001$, and the temporal correlated noise adopts the Ornstein Uhlenbeck process with $\sigma = 0.12$ and $\theta = 0.15$.

Algorithm 2 The Task Offloading And Recourse Allocation Algorithm based on DDPG

for each UE $m \in M$ do
  Initialize actor network $\mu(s|\theta_m^\mu)$ and critic network $Q(s, a|\theta_m^Q)$ with θ_m^μ and θ_m^Q
  Initialize actor-target networks μ' and critic-target networks Q' with weights $\theta_m^{\mu'} \leftarrow \theta_m^\mu$ and $\theta_m^{Q'} \leftarrow \theta_m^Q$
  Initialize experience replay buffer D_m as C
  for episode $e = 1,\ 2,\ 3, \cdots,\ E$ do
    Randomly generate a random process N for action exploration
    Randomly initialize state $s_{m,1}$
    for step $t = 1,\ 2,\ 3, \cdots, T$ do
      Select action $a_{m,t} = \mu(s_{m,t}|\theta_m^\mu) + N_{m,t}$ by running the current policy and using exploration noise $N_{m,t}$
      Execute action $a_{m,t}$ and then calculate reward $r_{m,t}$ and observe next state $s_{m,t+1}$
      Store $(s_{m,t}, a_{m,t}, r_{m,t}, s_{m,t+1})$ in D_m
      Sample a random mini-batch of $(s_{m,i}, a_{m,i}, r_{m,i}, s_{m,i+1})$ from D_m
      Set $y_{m,i} =$
      $r_{m,i} + \gamma_m Q'\left(s_{m,i+1}, \mu'\left(s_{m,i+1}|\theta^{\mu'_m}\right)|\theta^{Q'_m}\right)$
      Update critic network by minimizing the loss:
      $L = \frac{1}{N}\sum_i^N \left(y_{m,i} - Q\left(s_{m,i}, a_{m,i}|\theta_m^Q\right)\right)^2$
      Update actor network by using the sampled policy gradient:
      $\nabla_{\theta_m^\mu} J \approx \frac{1}{N}\sum_{i=1}^N \nabla_a Q\left(s_{m,i}, a|\theta_m^Q\right)\Big|_{a=a_{m,i}} \nabla_{\theta_m^\mu}\mu\left(s_{m,i}|\theta_m^\mu\right)$
      Update actor-target network and critic-target network by $\theta_m^{\mu'} \leftarrow \tau\theta_m^\mu + (1-\tau)\theta_m^{\mu'}$ and $\theta_m^{Q'} \leftarrow \tau\theta_m^Q + (1-\tau)\theta_m^{Q'}$
    end for
  end for
end for

B. Single User Equipment Scenario

In single UE scenario, the ω of the training and testing process is successively set as 0.5 and 0.8, respectively. Assuming the task size of training is 10 Mbits, and the task size of testing is from 10 to 15 Mbits. 1000 episodes are used in training and each episode consists of 200 steps. 200 episodes are used in testing and each episode includes 100 steps.

As $\omega = 0.5$, we can observe that the SA algorithm obtains the best result of the sum cost in Fig. 2, and the result of DDPG-based algorithm approximates the behavior of the SA. This fact indicated the training network of DDPG need to be well seated and trained. It can be seen from TABLE I that the running time of the latter to make an action is only 2.5% of the former, which means nearly 40 times quicker speed. The reason why the DDPG-based algorithm has a fast action speed compared to the SA algorithm is that the SA has to go through multiple iterations to jump out of the local optimal solution probabilistically and eventually obtain the global optimal solution, where the computation process is timeconsuming.Due to

the limited number of discrete action of DQN, the sum cost of DQN-based algorithm is higher than that of the DDPG-based and the SA algorithm. However, its running time of making an action is only 2.8% of the SA, and it achieves nearly 38 times quicker speed compared with the SA. In Fig. 3, although the DDPG-based algorithm does not have the best execution delay, it slightly compromises in the execution delay to bring about the second lowest energy consumption after the SA in Fig. 4. Since the energy consumption for most of tasks for "Full Local" is more than battery energy, the collected energy is exhausted in each step so that the phenomenon of energy consumption does not change in "Full Local" as shown in Fig. 4.

TABLE I
RUNNING TIME OF MAKING AN ACTION FOR UE.

ω	SA	DDPG	DQN	DDPG ÷ SA	DQN ÷ SA
0.5	18.6ms	0.47ms	0.52ms	0.025	0.028
0.8	20.9ms	0.50ms	0.57ms	0.024	0.027

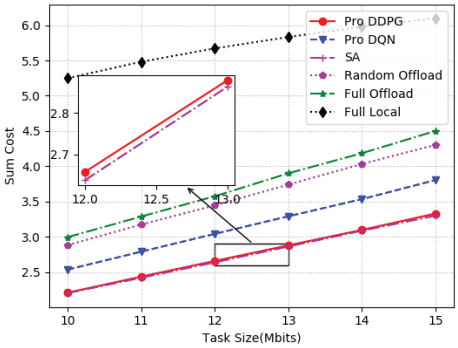

Fig. 2. The weighted sum cost for single UE with $\omega = 0.5$.

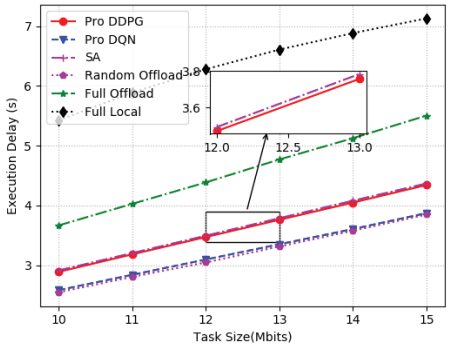

Fig. 3. The execution delay for single UE with $\omega = 0.5$.

When $\omega = 0.8$, we can observe that the sum cost of the DDPG-based algorithm still approximates the SA and outperforms other baselines from Fig. 5. However, the running time of the former to make an action is only 2.4% of the latter. In Fig. 7, we can see that although the energy consumption of DDPG and DQN at $\omega = 0.8$ is greater than that of DDPG and

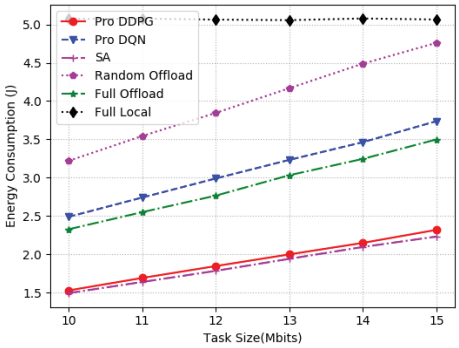

Fig. 4. The energy consumption for single UE with $\omega = 0.5$.

DQN at $\omega = 0.5$, the execution delays of DDPG and DQN are both lower after the SA in Fig. 6. The reason is that the execution delay of the sum cost will be given more penalty at $\omega = 0.8$. Moreover, although the gap between the energy consumptions of DDPG and DQN at $\omega = 0.8$ is narrower than that of DDPG and DQN at $\omega = 0.5$ in Fig. 7, the lowest execution delay is achieved for the DDPG-based algorithm under a reasonable running time in Fig. 6.

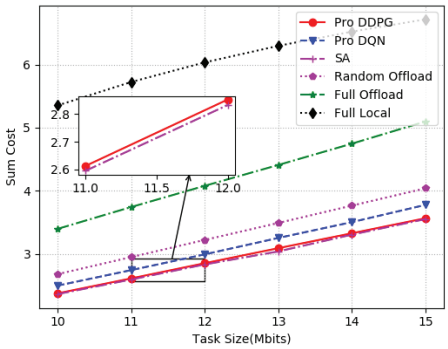

Fig. 5. The weighted sum cost for single UE with $\omega = 0.8$.

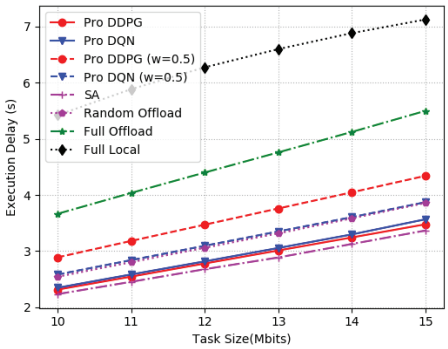

Fig. 6. The execution delay for single UE with $\omega = 0.8$.

C. Multi User Equipment Scenario

Since the running time of making an action for the SA is too long and unable to meet requirements of MEC system, it

TABLE II
TESTING RESULT FOR MULTI UES WITH $\omega = 0.5$.

	Sum Cost			Execution Delay(Unit: s)			Energy Consumption(Unit: J)		
	UE 1	*UE 2*	*UE 3*	*UE 1*	*UE 2*	*UE 3*	*UE 1*	*UE 2*	*UE 3*
Pro DDPG	**2.212**	**2.654**	**3.095**	2.889	3.468	4.045	**1.535**	**1.840**	**2.145**
Pro DQN	2.530	3.132	3.708	2.574	3.261	3.713	2.485	3.003	3.702
Full Local	5.246	5.677	5.969	5.426	6.272	6.889	5.065	5.082	5.048
Random Offload	2.882	3.462	4.038	**2.538**	**3.059**	**3.579**	3.227	3.865	4.497
Full Offload	2.999	3.582	4.206	3.667	4.389	5.139	2.332	2.775	3.274

TABLE III
TESTING RESULT FOR MULTI UES WITH $\omega = 0.88$.

	Sum Cost			Execution Delay(Unit: s)			Energy Consumption(Unit: J)		
	UE 1	*UE 2*	*UE 3*	*UE 1*	*UE 2*	*UE 3*	*UE 1*	*UE 2*	*UE 3*
Pro DDPG	**2.377**	**2.853**	**3.328**	**2.315**	2.777	**3.242**	2.623	3.155	3.670
Pro DQN	2.504	2.996	3.521	2.348	**2.741**	3.297	3.125	4.019	4.416
Full Local	5.352	6.034	6.522	5.422	6.273	6.883	5.073	5.078	5.077
Random Offload	2.681	3.226	3.756	2.543	3.064	3.571	3.235	3.872	4.492
Full Offload	3.397	4.087	4.737	3.665	4.407	5.115	**2.325**	**2.809**	**3.224**

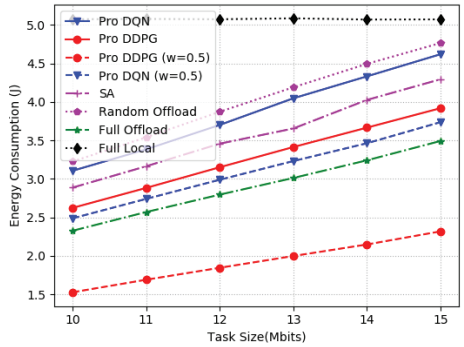

Fig. 7. The energy consumption for single UE with $\omega = 0.8$.

is not considered in multiuser MEC scenario. Assuming three UEs are located in MEC system, and the task sizes of testing for three UEs are set as 10, 12, 14 Mbits, respectively.

When $\omega = 0.5$, the sum cost of the DDPG-based algorithm is the lowest compared to the DQN-based algorithm and three baselines as shown in TABLE II. It can be observed that the DDPG-based algorithm can achieve the lowest energy consumption for all UEs by slightly compromising execution delay.

If $\omega = 0.8$, it can be seen from TABLE III that the execution delay of DDPG for UE 1 and UE 3 are lower than DQN and three baselines when more penalty is added to execution delay. However, the UE 2 achieves the lowest execution delay by using DQN-based algorithm, which indicates that the exploration of DDPG needs to be improved further. In a word, two strategies of the DDPG-based and DQN-based outperform three baselines in the sum cost for all UEs.

V. CONCLUSION

In this paper, in order to minimize the weighted sum cost of UE in MEC system, solve continuous behavior space problem, and fully utilize the advantages of parallel computing between the MEC server and UE, we propose the task offloading and resource allocation algorithms based on DQN and DDPG, which can allocate powers of local execution and that of transmission adaptively, and then determine the offloading proportion of task. It is performed in single UE and multi UEs MEC systems, respectively. Numerical results show that the performance of our proposed algorithms effectively reduce the weighted sum cost of UE and outperform three baselines except the SA.

ACKNOWLEDGMENT

This work was supported in part by National Natural Science Foundation of China under Grant 60972037, in part by Mobility Program for Taiwan Young Scientists under Grant RW2019TW001, and in part by Shenzhen Science and Technology Program under Grants GJHZ20180418190529516 and JSGG20180507183215520.

REFERENCES

[1] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," IEEE Communications Surveys Tutorials, vol. 19, no. 3, pp. 1657C1681, 2017.

[2] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," IEEE Transactions on Emerging Topics in Computing, pp. 1C1, 2019.

[3] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, Mobile-edge computing: Partial computation offloading using dynamic voltage scaling, IEEE Transactions on Communications, vol. 64, no. 10, pp. 4268C4282, 2016.

[4] J. Li and T. Lv, "Deep neural network based computational resource allocation for mobile edge computing," in Proc. 2018 IEEE Globecom Workshops (GC Wkshps), pp. 1C6, 2018.

[5] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," IEEE Transactions on Communications, vol. 65, no. 8, pp. 3571C3584, 2017.

[6] Y. Mao, J. Zhang, S. H. Song, and K. B. Letaief, "Power-delay tradeoff in multi-user mobile-edge computing systems," in Proc. 2016 IEEE Global Communications Conference (GLOBECOM), pp. 1C6, 2016.

[7] W. Wu, Q. Yang, B. Li, and K. S. Kwak, "Adaptive resource allocation algorithm of lyapunov optimization for time-varying wireless networks," IEEE Communications Letters, vol. 20, no. 5, pp. 934C937, 2016.

[8] H. Q. Ngo, E. G. Larsson, and T. L. Marzetta, "Energy and spectral efficiency of very large multiuser mimo systems," IEEE Transactions on Communications, vol. 61, no. 4, pp. 1436C1449, 2013.