# Deep Reinforcement Learning Based Task Offloading Algorithm for Mobile-edge Computing Systems

Hao Meng
Beijing Advanced Innovation Center for Future Internet Technology
Beijing University of Technology
Beijing, China
menghao0528@126.com

Daichong Chao
Beijing Advanced Innovation Center for Future Internet Technology
Beijing University of Technology
Beijing, China
chaodaichong@163.com

Qianying Guo
Beijing Advanced Innovation Center for Future Internet Technology
Beijing University of Technology
Beijing, China
guo_qianying@163.com

## ABSTRACT

Mobile-edge computing(MEC) is deemed to a promising paradigm. By deploying high-performance servers on the mobile access network side, MEC can provide auxiliary computing power for mobile devices, greatly reducing the computing pressure of mobile devices and improving the quality of the computing experience. In this paper, we consider the offloading problem of tasks in single-user MEC system. In order to minimize the mean energy consumption of mobile devices and the mean slowdown of tasks in the queue, we propose a deep reinforcement learning(DRL) based task offloading algorithm, and a new reward function is designed, which can guide the algorithm to optimize the trade-off between mean energy consumption and mean slowdown. The simulation results show that the deep reinforcement learning based algorithm outperforms the baseline algorithms.

## CCS Concepts

**• Networks → Network resources allocation • Computing methodologies→Policy iteration**

## Keywords

Mobile-edge computing(MEC), task offloading, deep reinforcement learning(DRL), mean energy consumption, mean slowdown.

## 1. INTRODUCTION

With the advent of the 5G era, a wide-range of new applications and services are emerging, e.g., augmented reality (AR), ultra-high-definition (UHD) and video virtual reality (VR), all of which are computation-intensive and delay-sensitive. This poses a big challenge to the battery life of mobile devices in spite of continuing improvements in battery technology. To cope with the above issue, a new service paradigm known as mobile edge computing (MEC) has been introduced. MEC integrates IT services into the edge of wireless access networks, creating a high bandwidth, low latency, and high performance service environment for users. By offloading the computation tasks from mobile devices to the nearby MEC servers, the experience of applications depending on the execution latency and energy consumption of mobile devices can be greatly improved.

*ICMAI 2019*, April 12–15, 2019, Chengdu, China
© 2019 Association for Computing Machinery
ACM ISBN 978-1-4503-6258-0/19/04…$15.00
http://doi.org/10.1145/3325730.3325732

Nevertheless, the efficiency of a MEC system mainly depends on the computation offloading scheme, in which many characteristics, e.g. computation task, wireless channel condition, energy consumption, should be taken into account. The discussion of this issue or similar may be found in the prior work. A dynamic offloading algorithm for MCC to achieve energy saving while satisfying given application execution time requirement is employed in [1], which is based on Lyapunov optimization method. A semi-Markovian decision process based control method for offloading for MCC is provided in [2], which aims to a balance between the application execution time and power consumption. In order for adapting to the characteristics of MEC, some other works have explored. In [3], the objective of this work focused on the full offloading decision is to minimize execution delay, which is solved by one-dimensional search algorithm. The computation offloading decision minimizing the energy consumption at the UE ends while satisfying the execution delay constraints is presented in [4] [5]. The former introduces an online learning strategy and an offline pre-calculated offline strategy corporately and the later proposes deterministic offline strategy and randomized offline strategy. However, the partial offloading manner is more flexible and realistic than full offloading. [6] [7] [8] aim to minimize minimization of the energy consumption while satisfying maximum allowable delay. [6] divides a task into a non-offloadable part and N offloadable parts and solves by an optimal adaptive algorithm based on a combinatorial optimization method. In [7], the offloading problem is formulated as binary programming model and execution correlation among divided parts is considered. The DVS technique is adopted and the variable substitution technique is combined with a locally optimal algorithm in [8]. A trade-off between mobile energy consumption and execution delay is delivered in [9] [10]. In [9], an iterative algorithm to seek out the optimal value of the number of bits to upload is presented. Reference [10] jointly optimizes task offloading scheduling and transmit power allocation for a single-user MEC system with multiple independent tasks by combining flow shop scheduling theory with convex optimization techniques. In a nutshell, few researchers attempt to adopt deep reinforcement learning methods to solve the scheduling and allocation problem in the MEC scenario with single-user.

In this paper, we consider a single-user MEC system that contains a mobile device and a MEC server. The task offloading problem on the mobile device will be investigated. To minimize the mean slowdown of tasks in the task buffer queue and the mean energy consuming of mobile device, we propose an online algorithm based on deep reinforcement learning method to schedule tasks to execution unit or transmission unit independently, thus translating the original problem into a learning problem. In our work, a system model is built to carry the reinforcement learning method, we also propose a new action space representation method in DRL, a new reward function is designed to guide the algorithm we proposed move towards the right objective. Simulation results show that compared with many traditional scheduling algorithms, the algorithm improves scheduling efficiency and has lower mean slowdown and energy consumption in mobile device.

The remaining of this paper is formed as follows: the background of reinforcement learning is introduced in Section 2; Section 3 presents the system model; The design of proposed algorithm is described in Section 4; the simulation results and conclusion are shown in Section 5 and Section 6.

## 2. BACKGROUND

In this section, we will briefly introduce the techniques and knowledge related to reinforcement learning (RL). RL is a product of multidisciplinary and multidisciplinary intersections. Its essence is to solve the problem of "Sequential Decision Making", in general, a reinforcement learning task can be characterized using Markov Decision Process (MDP). Specifically, in Figure 1, the agent is in an environment, it can only affect the environment by acting and perceive the current environment. when the agent performs an action, it will cause the environment to transfer to another state according to a certain probability, at the same time, the environment will feedback a reward to agent based on the reward function. Therefore, reinforcement learning mainly includes five elements: state space (S), action space(A), transfer probability($P_a(s,s')$), discount factor (γ) and reward function (R), the objective of it is to find an optimal strategy $\pi(s,a)$, so as to maximize the cumulative discounted reward $E[\sum_{t=0}^{\infty}\gamma^t r_t]$, where $\gamma$ is discount factor of future reward.

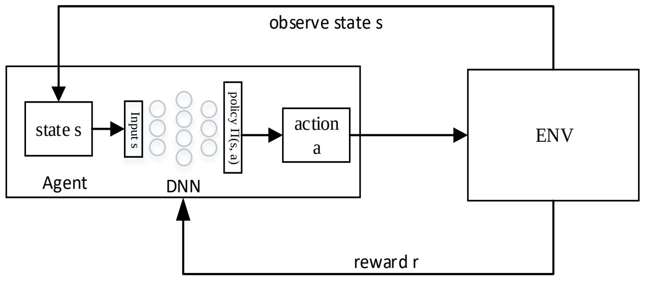

**Figure 1. Deep Reinforcement Learning model with DNN approximator.**

The policy mentioned above is a probability distribution over actions and a mapping relationship between state space and action space. In practice, the magnitude of this mapping relationship is very large, the traditional way of using the formal representation of state action pairs is infeasible, so using a function approximator is very common. A function approximator commonly has a controllable number of updatable parameters, θ. Hence, a policy with tunable parameter can be indicated as $\pi_\theta(s,a)$. In recent years, great breakthroughs have been made in deep neural network (DNN), so this paper uses a DNN as function approximator. But how to find the optimal policy by adjusting the parameters? we pay close attention to a class of reinforcement methods that performing gradient-descent on the parameters. The policy objective function of these methods can be represented as $J_{avR}(\theta)=\sum_s d^{\pi_\theta}(s)\sum_a \pi_\theta(s,a)R_s^a$, to maximizing the value of the objective function, the gradient given by [11] should be calculated:

$$\nabla_\theta E_{\pi_\theta}[\sum_{t=0}^{\infty}\gamma^t r_t]=E_{\pi_\theta}[\nabla_\theta log\pi_\theta(s,a)Q^{\pi_\theta}(s,a)] \qquad (1)$$

Where $Q^{\pi_\theta}(s,a)$ is the expected cumulative discounted reward from choosing action $a$ in state $s$, $\gamma$ is discount factor. In the basic Monte Carlo method [12], the agent selects and samples multiple trajectories and uses the calculated cumulative discount reward as an unbiased estimate of $Q^{\pi_\theta}(s,a)$, and then use gradient descent to update the parameters:

$$\theta \leftarrow \theta+\alpha\sum_t \nabla_\theta log\,\pi_\theta(s,a)(v_t-b_t) \qquad (2)$$

where α indicates the step size. $v_t$ is the cumulative discounted reward, which is obtained on a trajectory using the Monte Carlo Method. $b_t$ is the average value of multiple sampling for each $v_t$, it can reduce the variance of parameter gradient in DNN training.

## 3. SYSTEM MODEL

Figure 2 shows the structure of a single-user MEC system constructed in this paper. It mainly consists of a mobile device and a MEC server, which communicate through wireless transmission channel. The mobile device runs multiple independent, computationally intensive tasks, and is mainly composed of a task buffer queue, a task scheduling unit (task scheduler), a transmission unit including data transmitting (TX) and receiving (RX) module, and a local execution unit. Specifically, the task buffer queue Q is used to cache the pending tasks of the mobile device; the scheduler schedules tasks from task buffer queue Q to execution unit or transmission unit according to optimization objectives; The transmission unit and local execution unit can handle multiple tasks simultaneously, different colors in the figure represent different tasks, and each task requires a different proportion of resources. For example, we assume that the transmission unit and the execution unit have resources of $N_t$ and $N_e$ units respectively, in the transmission unit, green task transmission takes up three units of bandwidth, and the transmission of magenta task need to occupy four units of bandwidth; in the execution unit, the execution of purple and yellow task requires four and three units of CPU resources respectively. In addition, white unit indicates that the resource is idle. The MEC server is deployed in the vicinity of the base station, using the wireless channel of the base station to establish connections and communicate with mobile device. Assuming that the MEC server is equipped with sufficient computing resources, it can handle multiple computing tasks in parallel, that is, the task arrives at the server without caching and can be executed immediately, so the waiting delay on the MEC server can be ignored. Besides, the computational results of tasks are assumed to be very small and thus the feedback delay can be ignored.

In MEC system, the task buffer queue can be represented as $Q=\{J_1,J_2,\dots,J_i,\dots,J_N\}$, each task in $Q$ is characterized by a six-tuple, $J_i=\{t_{enter,i},t_{start,i},\ t_{fin,i},t_{proc,i},r_{exe,i},r_{trans,i}\}$, where $t_{enter,i}$ is the time step for task $J_i$ to enter the queue, $t_{start,i}$ and $t_{fin,i}$ are the time steps for the start and completion of task $J_i$, $t_{proc,i}$ indicates the time required for task $J_i$ to be processed on the mobile device, $r_{exe,i}$ and $r_{trans,i}$ indicates the expected proportion of local execution unit and transmission unit occupied by task $J_i$. We assume that the energy required to be consumed by the local execution unit and the transmission unit at full load is $p_e$ and $p_t$, respectively. Hence, If the current system resource occupancy is $U_e$ and $U_t$, then the energy consumption of mobile device at the current time step can be expressed as:

$$\mathrm{E} = \frac{U_e}{N_e} * p_e + \frac{U_t}{N_t} * p_t \tag{3}$$

The slowdown of task $J_i$ can be calculated as:

$$T_{slowdown,i} = \frac{t_{fin,i} - t_{enter,i}}{t_{fin,i} - t_{start,i}} \tag{4}$$

The mean slowdown of tasks in the queue and energy consumption in mobile device are important indicators for evaluating the performance of algorithms in single-user MEC systems. In this paper, we propose a deep reinforcement learning based algorithm, aiming at minimizing the mean slowdown of tasks in the queue and energy consumption in mobile device by optimizing task offloading policy, and reducing the energy consumption of mobile devices while speeding up the processing speed of tasks in the queue.

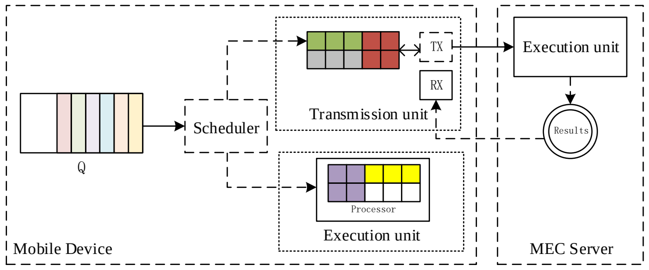

**Figure 2. The structure of single-user MEC system.**

## 4. DESIGN OF ALGORITHM

In this paper, the deep reinforcement learning algorithm is used to design and analyze the algorithm from four aspects based on the system model, which will be elaborated below.

**State space.** The state of single-user MEC system can be denoted as S, which consists of the resource occupancy status of local execution unit and transmission unit in mobile device and the resource requirement status of tasks in task buffer queue, similar to the state space described in [13], but not quite the same. In particular, we use a matrix to describe the resource occupancy status of the execution unit or transmission unit in mobile device, as shown in Figure 3(a), the horizontal axis represents the time, the vertical axis represents the resource occupancy status, and the whole matrix represents the resource allocation and occupancy status of the execution unit or transmission unit in the future T-time step. Similarly, as depicted in Figure 3(b), the state of the tasks in the queue is also represented by a matrix of the same dimension. The difference is that the matrix shown in Figure 3(a) represents the allocation and occupancy state of the future T-step resources, while the task matrix corresponds to each task one by one, only expressing the expected amount of resources occupied and the length of time steps consumed by the task.

Since the input dimension of deep neural network is fixed, but the amount of tasks in $Q$ is constantly changing, so we only consider the state of the first $D$ tasks in the queue $Q$. Intuitively, it's enough to focus on the first- arrival tasks, because reasonable policies are likely to prefer tasks that have been waiting longer, and this operation also provides convenience for modeling the action space below.

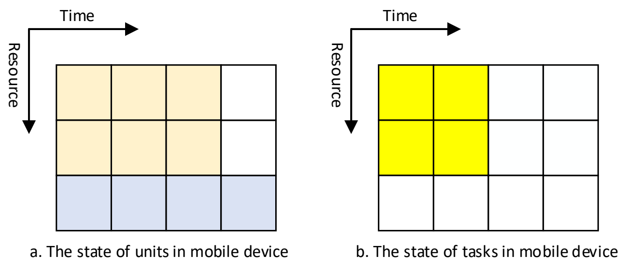
Time
Resource
Time
Resource
a. The state of units in mobile device
b. The state of tasks in mobile device

**Figure 3. State representation.**

**Action space.** In a single-user MEC system, we define scheduling a task from task queue $Q$ to the local execution unit or to the transmission unit as an action. As can be seen from the definition of action, an action A contains two sub-operations $\{A_1, A_2\}$: $A_1$ selects a task $J_i$ from the task queue $Q$; $A_2$ sends the task $J_i$ selected by $A_1$ to the execution unit or the transmission unit. We define the depth of sub-operation $A_1$ as $D$, so the $A_1$ can be denoted as $A_1 = \{\emptyset, 1, 2, \ldots, i, \ldots, D\}$, where $\emptyset$ indicates that no action is taken at the current time step, i means to select i-th task from task queue $Q$ for scheduling. Meanwhile, $A_2$ can be expressed as $A_2 = \{0,1\}$, where 0 means to dispatch task $J_i$ to the local execution unit, while 1 means to schedule task $J_i$ to the transmission unit. So the action space can be represented as:

$$\mathrm{A} = A_1 \times A_2 = \{(a_1, a_2) | a_1 \in A_1 \land a_2 \in A_2\} \tag{5}$$

The action space is composed of Cartesian products of sub-operation sets $A_1$ and $A_2$. Each action a is represented by a two-tuple $(a_1, a_2)$, which means that the scheduler dispatches the task $a_1$ from the task queue $Q$ to the $a_2$ sub-queue.

In addition, with reference to [13], at each time step, the time stops until the void action or invalid action is selected by the scheduler. This not only satisfies our established requirements, but also reduces the complexity of the action space.

**Rewards.** The goal of our deep reinforcement learning based algorithm is to minimize the mean energy consumption of mobile device and the mean slowdown of the tasks in the queue, so we define the reward as:

$$\mathrm{R} = R_t + R_e \tag{6}$$

Where $R_t$ is the reward for execution delay, which can be calculated as follows:

$$R_t = \beta \cdot \sum_{i \in J} \frac{-1}{t_{proc,i}} \tag{7}$$

where $J$ represents the set of all unfinished tasks in a mobile device, including tasks waiting to be scheduled for execution in $Q$ and tasks processed in execution unit or transmission unit. As we can see, when set the discount factor $\gamma = 1$, maximizing the cumulative discounted reward is equivalent to minimizing the average slowdown.

$R_e$ is the reward of the energy consumption generated in the current time step, the formula can be expressed as follows:

$$R_e = (1 - \beta) \cdot (-1) * E \tag{8}$$

where E is defined in Formula 3, $\beta$ is trade-off factor, which satisfies [0,1], indicating the importance of $R_t$ and $R_e$.

**Approximator.** we use DNN as function approximator. It consists of an input layer, two hidden layers and an output layer. The dimension of the input layer is the same as the dimension of the state space. The two hidden layers can be represented by H1 and H2 respectively, and their dimensions are defined as the smallest power of 2 that greater than the input dimension. The dimension of the output layer is the same as the dimension of the action space and Softmax[14] is used to output the probability distribution of actions in the current state. It should be noted that the action space defined in this paper is a two-tuple, so we use the product of two sub-action space's dimensions as the dimension of the neural network output layer. At the same time, after each hidden layer, we use ReLU[15] as the activation function, so the network structure of DNN can be shown as: input->H1->relu->H2->relu->output->softmax.

## 5. SIMULATION RESULTS

We carried out simulation experiments to verify the performance of the algorithm. For task$J_i = \{t_{enter,i}, t_{start,i}, t_{fin,i}, t_{proc,i}, r_{exe,i}, r_{trans,i}\}$, the values of $t_{enter,i}$, $t_{start,i}$ and $t_{fin,i}$ are dynamically allocated when the single-user MEC system runs, the $t_{proc,i}$, $r_{exe,i}$, $r_{trans,i}$ are assumed to be uniformly distributed, for example, we assume that the probability that $t_{proc,i}$ obeys $rand([1,3])$ is 0.8, and the probability of obeying $rand([10,15])$ is 0.2, $r_{exe,i} \sim rand([N_e/2, N_e])$, $r_{trans,i} \sim rand([N_t/2, N_t])$, where $N_e = N_t = 8$. Besides, we set $p_e = 15, p_t = 10, \beta = 0.9$, $D = 5$.

With the definition above, we can calculate that the dimension of action space is 12, so we set the output dimension of DNN as 12. The dimension of H1 and H2 can be set to 32 and 20. Assuming that the action value of Monte Carlo sampling is c, then the sub-operation $(a_1, a_2)$ can be calculated according to the following formula: $a_1 = \lfloor c/2 \rfloor, a_2 = c \% 2$. In addition, the generation of tasks in MEC system obeys the Bernoulli process.

Three baseline offloading algorithms are compared: All in mobile device, All in MEC server, and Random. All in mobile device means that scheduler schedules all tasks in the task queue to the execution unit of mobile device for processing; All in MEC server indicates that the scheduler only schedules tasks to the transmission unit and then the tasks in transmission unit would be offloaded to MEC server for execution; Random represents that the scheduler schedules tasks in the task queue to the transmission unit or execution unit randomly. To facilitate simulation, we schedule tasks from the task buffer queue according to the order of task arrival time and assume that the execution time in the local execution unit is twice that of the transmission unit in the mobile device.

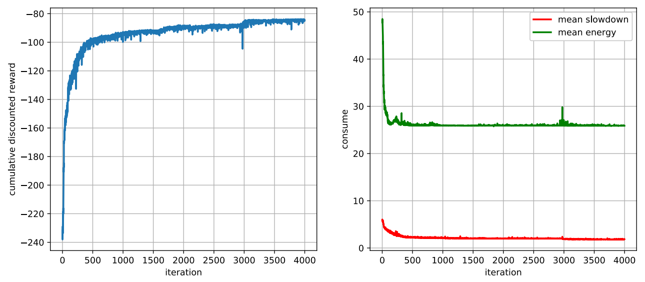

**Figure 4. The discounted total rewards and the mean slowdown and mean energy consumption across iteration.**

In the simulation, the policy gradient method is used to train the deep reinforcement learning model. As can be seen from Figure 4, as the iteration progresses, the reward gradually rises from -238.06 to nearly -83.70, indicating that the policy $\pi_\theta(s, a)$ is moving towards the direction that making the objective function better guided by reward function. Meanwhile, with the increase of reward, the mean slowdown and mean energy consumption are gradually decreasing, the slowdown decreased from 6.04 to about 1.79, with a decrease of 4.25, and the energy consumption also decreased from 48.49 to 25.75. As the number of training iteration increases, the model converges gradually, and the rewards does not continue to rise, but swings back and forth at the highest point. Similar to rewards, the mean slowdown and mean energy consumption swings near the lowest point. Figure 5 shows the comparison between our proposed algorithm and the basic algorithm on mean slowdown and mean energy consumption. As we can see, with the increase of task generation rate, the algorithm we proposed is superior to baselines in terms of mean slowdown, for mean energy consumption, in our simulation environment, offloading all tasks to the MEC performs the least energy consumption. It can be seen that as the task generation rate increases, our algorithm effectively balances the mean slowdown and mean energy consumption.

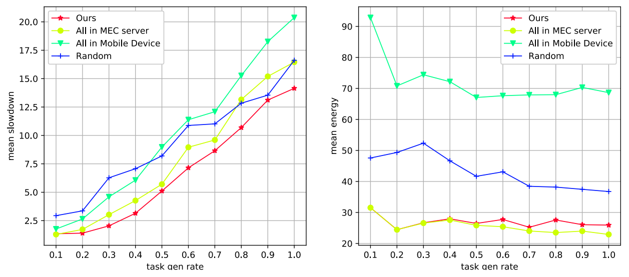

**Figure 5. the comparison of mean slowdown and mean energy consumption.**

## 6. CONCLUSION

Aiming at the problem of task offloading in single-user MEC system, a task offloading algorithm based on deep reinforcement learning is proposed in this paper, the goal is to minimize the mean energy consumption of mobile devices and the mean slowdown of tasks in the queue. The simulation results show that the proposed algorithm can learn the approximate optimal task offloading strategy after training, and is superior to the traditional task offloading algorithm in performance. In the future research, factors such as multi-core CPU and their power, the power transmission unit and the distance of mobile device from the MEC server will be considered to be closer to actual needs.

## 7. REFERENCES

[1] Huang, D. , Wang, P. , and Niyato, D. . 2012. A dynamic offloading algorithm for mobile computing. *IEEE Transactions on Wireless Communications*, 11, 6, 1991-1995. DOI= http://dx.doi.org/10.1109/ISIT.2016.7541539

[2] Chen, S. , Wang, Y. , and Pedram, M. . 2013. A semi-Markovian decision process based control method for offloading tasks from mobile devices to the cloud. *IEEE Global Communications Conference*. IEEE. DOI= http://dx.doi.org/10.1109/GLOCOM.2013.6831512

[3] Liu, J. , Mao, Y. , Zhang, J. , and Letaief, K. B. . 2016. Delay-optimal computation task scheduling for mobile-edge computing systems. *2016 IEEE International Symposium on Information Theory*. IEEE. DOI= http://dx.doi.org/10.1109/ISIT.2016.7541539

[4] Kamoun, M. , Labidi, W. , and Sarkiss, M. . 2018. Joint resource allocation and offloading strategies in cloud enabled

cellular networks. *IEEE International Conference on Communications*. IEEE. DOI= http://dx.doi.org/10.1109/ICC.2015.7249203

[5] Labidi, W. , Sarkiss, M. , and Kamoun, M. . 2015. Energy-optimal resource scheduling and computation offloading in small cell networks. *International Conference on Telecommunications*. IEEE. DOI= http://dx.doi.org/10.1109/ICT.2015.7124703

[6] Cao, S. , Tao, X. , Hou, Y. , and Cui, Q. . 2016. An energy-optimal offloading algorithm of mobile computing based on HetNets. *International Conference on Connected Vehicles and Expo.* IEEE. DOI= http://dx.doi.org/10.1109/ICCVE.2015.68

[7] Deng, M. , Tian, H. , and Fan, B. . 2016. Fine-granularity based application offloading policy in cloud-enhanced small cell networks. *IEEE International Conference on Communications Workshops*. IEEE. DOI= http://dx.doi.org/10.1109/ICCW.2016.7503859

[8] Wang, Y. , Sheng, M. , Wang, X. , Wang, L. , and Li, J. . 2016. Mobile-edge computing: partial computation offloading using dynamic voltage scaling. *IEEE Transactions on Communications*, 1-1. IEEE. DOI= http://dx.doi.org/10.1109/TCOMM.2016.2599530

[9] Muñoz, Olga, Pascual-Iserte, A. , and Vidal, J. . 2014. Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading. *IEEE Transactions on Vehicular Technology*, 64, 10, 4738-4755. IEEE. DOI= http://dx.doi.org/10.1109/TVT.2014.2372852

[10] Mao, Y. , Zhang, J. , and Ben Letaief, K. . 2017. Joint Task Offloading Scheduling and Transmit Power Allocation for Mobile-Edge Computing Systems. *Wireless Communications and Networking Conference*. IEEE. DOI= http://dx.doi.org/10.1109/WCNC.2017.7925615

[11] Sutton, R. , and Barto, A. . 1998. Reinforcement learning - an introduction. *IEEE Transactions on Neural Networks*, 9, 5, 1054-1054. IEEE. DOI= http://dx.doi.org/10.1109/TNN.1998.712192

[12] R. S. Sutton, D. A. McAllester, S. P. Singh and Y. Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. *Neural Information Processing Systems*, 12. MIT.

[13] Mao, H. , Alizadeh, M. , Menache, I. , and Kandula, S. . 2016. Resource Management with Deep Reinforcement Learning. *Acm Workshop on Hot Topics in Networks*. DOI= http://dx.doi.org/10.1145/3005745.3005750

[14] Priddy, K. L. , and Keller, P. E. . 2005. Artificial Neural Networks: An Introduction.

[15] Glorot, Xavier. Bordes, Antoine and Bengio, Y. 2010. Deep Sparse Rectifier Neural Networks. *Journal of Machine Learning Research*. 15.