Turkish Journal of Electrical Engineering & Computer Sciences
http://journals.tubitak.gov.tr/elektrik/
Research Article
Turk J Elec Eng & Comp Sci
(2023) 31: 498 – 515

doi:10.55730/1300-0632.3998

Task offloading and resource allocation based on DL-GA in mobile edge computing

Hang GU[1], **MinJuan ZHANG**[1,*], **WenZao LI**[2], **YuWen PAN**[3]
[1]School of information and Communication Engineering, North University of China, Taiyuan, China
[2]Educational Informationalization and Big Data Center, Education Department of Sichuan Province, Chengdu, China
[3]School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu, China

Received: 09.05.2022 • **Accepted/Published Online:** 13.03.2023 • **Final Version:** 28.05.2023

Abstract: With the rapid development of 5G and the Internet of Things (IoT), the traditional cloud computing architecture struggle to support the booming computation-intensive and latency-sensitive applications. Mobile edge computing (MEC) has emerged as a solution which enables abundant IoT tasks to be offloaded to edge services. However, task offloading and resource allocation remain challenges in MEC framework. In this paper, we add the total number of offloaded tasks to the optimization objective and apply algorithm called Deep Learning Trained by Genetic Algorithm (DL-GA) to maximize the value function, which is defined as a weighted sum of energy consumption, latency, and the number of offloaded tasks. First, we use GA to optimize the task offloading scheme and store the states and labels of scenario. Each state consists of five parameters: the IDs of all tasks generated in this scenario, the cost of each task, whether the task is offloaded, bandwidth occupied by offloaded task and remaining bandwidth of edge server. The labels are the tasks that are currently selected for offloading. Then, these states and labels will be used to train neural network. Finally, the trained neural network can quickly give optimization solutions. Simulation results show that DL-GA can execute 75 to 450 times faster than GA without losing much optimization power. At the same time, DL-GA has stronger optimization capability compared to Deep Q-Learning Network (DQN).

Key words: Mobile edge computing, task offloading, resource allocation, genetic algorithm, deep learning

1. Introduction

With the development and progress of 5G and Internet of things (IoT), many computation-intensive applications and services are developed, such as augmented reality (AR), intelligent transportation, and intelligent manufacturing and smart grid [1–6]. However, those applications and services require a network transmission environment with low latency [7]. For example, AR service is a latency-sensitive service which needs to ensure that all users interact with the AR environment efficiently and synchronously, which requires high computational power of the mobile device (MD) [2]. However, it is more common to upload these computationally intensive tasks to a powerful remote server for processing than to MDs with limited computational power, battery and storage capacity.

Mobile cloud computing (MCC) is a very popular solution with strong computing and storage capabilities [8]. However, since the remote servers are far away from MDs, the communication links are blocked when a

*Correspondence: zmj7745@163.com

large amount of data are transferred to MCC servers, resulting in high delay and high energy consumption, which negatively affects latency-sensitive applications [9]. To address this issue, mobile edge computing (MEC) is proposed as a reliable solution [10]. MEC allows MDs to offload their computing tasks to the MEC servers deployed near the MDs, which can effectively reduce response latency and energy consumption of MDs and provide a better user experience [11]. Since the edge servers are distributed at the edge of the network and provide data processing services in close proximity, they can effectively reduce the risk of user data privacy leakage and greatly improve data security [12]. In addition, the edge server and the external network can operate independently, which means it reduces the dependency on the external network and can provide reliable and stable communication links for MDs [13]. Despite these advantages, MEC still needs to figure out some obvious problems, which motivate our work. On the one hand, a disorderly task offloading strategy consumes a lot of unnecessary resources of the edge servers. On the other hand, only a fraction of computational tasks can be offloaded to the MEC server due to bandwidth and computational resource constraints [14].

In this paper, we aim to not only reduce the energy consumption and latency but also maximize the number of offloaded tasks by optimizing the task offloading strategy. We apply a deep learning trained by genetic algorithm (DL-GA) to optimize this problem. In this algorithm, deep neural network (DNN) is applied as experience collector in DL-GA. GA can be considered an "explorer" which obtains different optimization schemes in different environment and those optimization schemes can be understood as experience to train the DNN. Finally, the DNN trained by a large number of experiences can offer the optimization scheme rapidly. The main contribution of this paper are that the DL-GA algorithm is applied to solve the multiobjective optimization problem in MEC and the experiment results proves that DL-GA has a good optimization effect similar to GA and execution time of DL-GA is much faster than GA. Meanwhile, DL-GA has better optimization ability than deep Q-learning network (DQN).

The rest of the paper is organized as follows. The related work of resource allocation and task offloading is analyzed in Section 2. Section 3 introduces the system model and formulates the optimal problem. The detailed process of DL-GA is introduced in Section 4. The simulation results of the algorithm are analyzed in Section 5. Section 6 concludes this paper.

2. Related work

Resource allocation means that the MEC server allocates its own computing resources and storage to handle the computing tasks offloaded by the MDs. Task offloading strategy decides whether computing tasks should be offloaded. If the task decides not to be offloaded, it will be processed locally. If the task decides to be offloaded to the edge servers, then strategy needs to determine which edge server should process this task. The purpose of resource allocation and task offloading is to reduce response latency and energy consumption of MDs. There are some existing studies focusing on reducing the latency. Ning et al. [15] provided an iterative heuristic MEC resource allocation (IHRA) algorithm to reduce latency efficiently based on multiuser environment. Bai et al. [16] introduced intelligent reflecting surfaces (IRS) to optimize the proposed MEC system, which can reduce computational latency efficiently. Tiwary et al. [17] designed a noncooperative extensive game model. In this model, Karush–Kuhn–Tucker (KKT) conditions and Nash Equilibrium are applied to reduce average latency. Wang et al. [18] considered the impact of user equipment movement on the delay and designed a reinforcement learning-based online microservice coordination algorithm to reduce the overall latency. Similarly, there are also some researchers focusing on saving the energy consumption of UDs. Chen et al. [19] introduced an energy-efficient dynamic offloading algorithm (EEDOA) to minimize the energy consumption under a condition

where they try to stabilize the length of the offloading task queue at a low level in an unstable wireless channel environment. Zhao et al. [20] designed an algorithm based on dynamic programming to minimize the energy consumption of multiple mobile devices. Yang et al. [21] studied a DQN-based reinforcement learning algorithm to reduce the energy consumption and applied a conventional optimization algorithm as contrast. Certainly, the energy consumption and response latency can also be considered to be optimized simultaneously which is formulated as an NP-hard problem [22]. Nath et al. [23] introduced a deep deterministic policy gradient (DDPG) method to minimize the weighted sum of energy consumption, latency, and cost of cache contents. Tong et al. [24] designed a deep reinforcement learning (DRL) method to optimize the energy consumption and average response time, which works more effectively compared to reinforcement learning (RL). Zhang et al. [25] tried to minimize the cost which is weighted sum of energy consumption and response delay. They combined K-means clustering and GA to solve this multiobjective optimization problem.

However, the above studies did not consider the impact of the number of offloaded tasks on the user experience. Since the limited resources of MEC servers cannot support all tasks when the number of tasks increases gradually in relevant scenarios, the number of offloaded tasks is also an important optimization objective. Liu et al. [26] designed an extended marriage algorithm (EMA) to maximize the number of offloaded tasks before deadline, but the optimization both energy consumption and latency was not considered in this paper. Li et al. [27] designed further a novel genetic algorithm named M-COGA which tried to maximize the weighted sum of the number of offloaded tasks and the cost where cost represents the weight combination of energy consumption and latency. Although the M-COGA has good optimization effect, it still needs to overcome some defects, that is, the iterative convergence speed of genetic algorithm will be slower and the algorithm execution time will be longer in the face of more complex scenarios, which greatly affects the user experience.

In [28], the DL-GA was proposed to solve the path planning of unmanned aerial vehicle (UAV). This algorithm overcomes the disadvantage of long iterative convergence time of GA. Therefore, we will apply DL-GA in MEC to solve the multiobjective optimization problem, that is, to maximize the weighted sum of the number of offloaded tasks and the cost where cost represents the weight combination of energy consumption and latency.

3. System model and problem formulation

In this section, we will model both MDs and MEC servers. We will then formulate this multiobjective problem. All notations used commonly are shown in Table 1.

3.1. System model

The MEC model consisting of n MDs and m base stations (BSs) is shown in Figure 1. Each BS is deployed with an edge server to process the data received by the BS, which constitutes a complete MEC server.

In this research, we assume that each MD has only one task. In other words, MD_i generates a task T_i, where MD_i denotes the id of the MD which generated task T_i. MD_i can process T_i locally if it has enough resources. The local processing time of a task T_i can be described as

$$t_i = \frac{d_i}{f_i}, \tag{1}$$

where d_i represents the required local CPU cycles for task T_i, and f_i represents CPU cycles of MD_i per

Table 1. Commonly used terms in the MEC model.

Notation	Definition
n	Number of MD
m	Number of BS
MD_i	Id of MD
T_i	Id of task
t_i	Local processing time of T_i
d_i	Required local CPU cycles for T_i
f_i	CPU cycles of MD_i per second
$\mathcal{J}_i$	Energy consumption coefficient per CPU cycle
e_i	Local energy consumption of MD_i
B	Channel bandwidth of BS
S	Average power of signal
N	Average power of noise
C_B	Channel capacity
B_i	Bandwidth of subchannel
$C_s(i)$	Channel capacity occupied by offloaded task T_i
$Cost(i)$	Weighted sum of energy consumption and latency
λ_t	Weight parameter of t_i
λ_e	Weight parameter of e_i
N_t	Total number of offloaded tasks
C_v	Total cost of offloaded tasks
$\theta(N_t)$	Normalized N_t
$\theta(C_v)$	Normalized C_v
λ_{N_t}	Weight parameter of N_t
λ_{C_v}	Weight parameter of C_v
$\phi(N_t, C_v)$	system effectiveness
L_i	Whether to offload T_i
$min(N_t)$	Theoretically minimum number of offloaded tasks
$max(N_t)$	Theoretically maximum number of offloaded tasks
$min(C_v)$	Theoretically minimum total cost of offloaded tasks
$max(C_v)$	Theoretically maximum total cost of offloaded tasks
$min(Cost)$	Theoretically minimum cost
$max(Cost)$	Theoretically maximum cost

second. Then, the local energy consumption of MD_i for computing the task T_i can be described as

$$e_i = \mathcal{J}_i d_i, \tag{2}$$

where $\mathcal{J}_i$ is the energy consumption coefficient per CPU cycle. Generally, the time and energy consumption required for each CPU cycle will not be exactly the same. However, it is difficult to accurately calculate the time and energy consumption of processing a task due to the complex working mechanism of the CPU. Therefore, f_i and $\mathcal{J}_i$ can be understood as a coefficient in this model.

In this MEC server, BS is configured to receive the data to be offloaded from MDs through wireless transmission and send it to the edge server for processing. The maximum transmission rate of BS can be

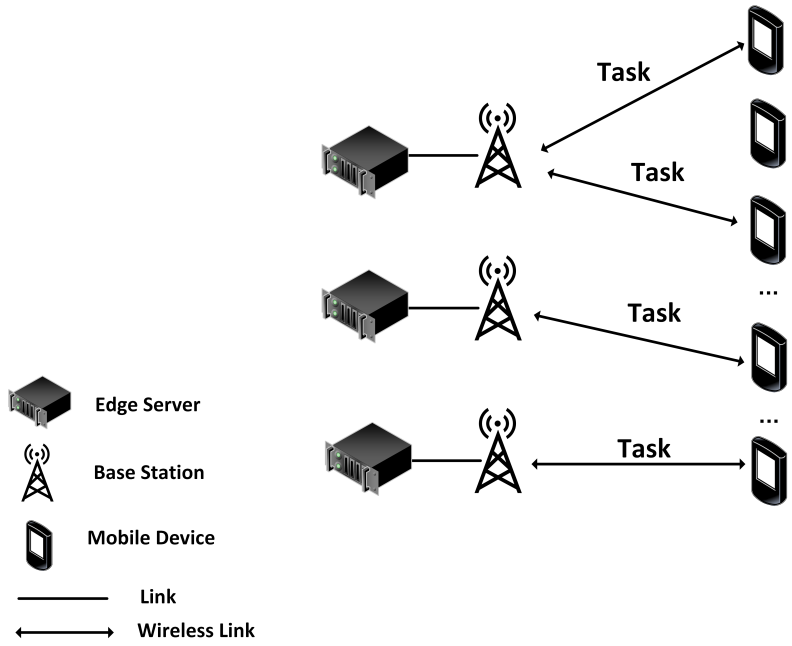


Figure 1. The task offloading in the MEC system.

assumed as

$$C_B = B\log_2(\frac{S}{N}), \tag{3}$$

where B is the channel bandwidth, S present the average power of signal and the N represents the average power of noise. C_B is also called channel capacity. Usually, the actual value is often smaller than the theoretical value C_B due to the influence of the real environment [27]. When task T_i will be offloaded to the BS, the task T_i needs to occupy a certain channel capacity called subchannel, which can be expressed as

$$C_s(i) = B_i\log_2(\frac{S_i}{N}), \tag{4}$$

where B_i is the bandwidth of this subchannel, and the $C_s(i)$ indicates the channel capacity occupied by offloaded task T_i. From Eqs. (3) and (4), limited channel capacity cannot carry unlimited tasks transmission, that is, $C_B \geq \sum_{i=1}^{n} C_s(i)$. Generally, if the channel quality is bad, it is more suitable to process task locally rather than offloading to the MEC server, because a large part of energy is wasted in data transmission. If the channel quality is efficient, it is more suitable to offload all tasks to the MEC [29]. However, the influence of channel quality on offloading strategy will not be discussed in this paper.

3.2. Problem formulation

The main purpose of resource allocation and task offloading is to reduce energy consumption and response latency of MDs. We denote $Cost(i)$ as the weighted sum of energy consumption and response latency, which can be described as

$$Cost(i) = \lambda_t t_i + \lambda_e e_i. \tag{5}$$

Tasks with high *Cost* will be preferentially offloaded to BS. λ_t and λ_e are the weight parameter which influence the optimization ability. In other words, if we focus on reducing the UDs' response latency, we can set $\lambda_t > \lambda_e,\ \lambda_t, \lambda_e \in [0,1]$ and vice versa.

In addition to the $Cost(i)$, the MD can also decide whether to request offloading task. We denote $L_i \in \{0,1\}$. MD_i determines to request offloading T_i if $L_i = 1$, and if $L_i = 0$, the task T_i will be processed locally. Thus, the task is defined by several important attributes, i.e. $T_i = \{MD_i, Cost(i), C_s(i), L_i\}$. In this study, we assume that all offloaded tasks can be fully processed. In other words, we assume that the resources of MEC server are sufficient to calculate all uploaded tasks. Therefore, we want to offload more tasks simultaneously with limited channel capacity while guaranteeing a high *Cost* of tasks, which will be formulated as

$$\begin{aligned} &\max \quad n \text{ and } \sum_{i=1}^{n} L_i * Cost(i)) \\ &s.t. \quad C_B \geq \sum_{i=1}^{n} L_i * C_s(i) \end{aligned} \tag{6}$$

In order to better analyze the research results, we use system effectiveness $\phi(N_t, C_v)$ to quantify the above optimization objectives, which is defined as

$$\phi(N_t, C_v) = \lambda_{N_t}\theta(N_t) + \lambda_{C_v}\theta(C_v), \tag{7}$$

where $\lambda_{N_t}, \lambda_{C_v} \in (0,1)$ denote weight coefficients and θ is the normalization function that keeps the two parameters at the same dimensions. $\theta(N_t)$ represents the total normalized number of offloaded tasks which is defined as

$$\theta(N_t) = \frac{N_t - min(N_t)}{max(N_t) - min(N_t)}, \tag{8}$$

where N_t is the actual total number of offloaded tasks, $min(N_t)$ is the theoretically minimum number of offloaded tasks and $max(N_t)$ is the theoretically maximum number of offloaded tasks. And $\theta(C_v)$ is the total normalized cost of offloaded tasks, which is defined as

$$\theta(C_v) = \frac{C_v - min(C_v)}{max(C_v) - min(C_v)}, \tag{9}$$

where C_v is the actual total cost of offloaded tasks, $min(C_v)$ is the theoretically minimum total cost of offloaded tasks and $max(C_v)$ is the theoretically maximum cost of offloaded tasks. $min(C_v)$ and $max(C_v)$ are defined as

$$\begin{cases} max(C_v) = max(N_t) * max(Cost), \\ min(C_v) = min(N_t) * min(Cost), \end{cases} \tag{10}$$

where $max(Cost)$ and $min(Cost)$ are the maximum and minimum theoretical cost value.

4. Resource allocation based on DL-GA

In this section, we introduce DL-GA, which is an algorithm that can be divided into two steps. We first use GA to collect states and labels as a dataset, which is then trained by a deep neural network.

4.1. Genetic algorithm

GA is one of the heuristic algorithms with strong generalization and local search ability. In this system, the algorithm is divided into the following steps.

4.1.1. Establish initial population

First of all, an initial population is established as a matrix P

$$P = \begin{bmatrix} \tau_{1,1} & \tau_{1,2} & \cdots & \tau_{1,q} \\ \tau_{2,1} & \tau_{2,2} & \cdots & \tau_{2,q} \\ \cdots & \cdots & \cdots & \cdots \\ \tau_{M,1} & \tau_{M,2} & \cdots & \tau_{M,q} \end{bmatrix}, \tag{11}$$

where M represents the number of individuals. Each row of this matrix represents a possible offloading strategy for task set $T = \{T_1, T_2, \cdots, T_q\}$, which means $\tau_{M,i} \in T,\ i \in (1, q)$. After establishing the population, the fitness value of each row will be calculated. Fitness is the evaluation criterion of each row which represents the individual viability. In GA, the system effectiveness is called fitness value. The fitness computation can be described by Algorithm 1.

Algorithm 1 Fitness computation

Input: Population P
Output: Fitness set F

Initialize population with matrix P
Initialize empty fitness set F
for $k = 1, M$ **do**
    Initialize channel capacity of BSs
    Initialize total cost $C_v = 0$ and total number $N_t = 0$
    for $j = 1, n$ **do**
        Select maximum remaining channel capacity of BSs as $C_{res}(B)$
        if $C_{res}(B) \geq C_s(\tau_{k,j})$ **then**
            $C_v += Cost(\tau_{k,j})$
            $N_t += 1$
            $C_{res}(B) -= C_s(\tau_{k,j})$
        else
            break
        end if
    end for
    $F.append(\lambda_{N_t}\theta(N_t) + \lambda_{C_v}\theta(C_v))$
end for
return F

4.1.2. Roulette algorithm (RA)

After calculating the fitness value of each row by Algorithm 1, a set F will be obtained, and then the survival probability of each individual can be calculated as

$$P_r(i) = \frac{F(i)}{\sum_{m=1}^{M} F(m)}, \tag{12}$$

A new survival probability set $P_r(i = 1, 2, 3, \cdots, M)$ can be generated by equation (12). And next, the population will be randomly selected for M times, and a new population matrix P_{IN} will be formed. As individual with greater fitness are easier to survive, individual with greater P_r are more likely to be selected multiple times. In order to create next generation individuals to continue the selection, the crossover operation (CO) and mutation operation (MO) will be used after that.

4.1.3. Crossover operation

Crossover and subsequent mutation are important methods to ensure that GA has good global and local search ability. CO randomly selects two adjacent rows from P_{IN}, which is described as row $P_j = [\tau_{j,1}, \cdots, \tau_{j,q}]$and $P_k = [\tau_{k,1}, \cdots, \tau_{k,q}]$. And we randomly select an index $i, i \in (1, q)$ to exchange $\tau_{j,i}$ and $\tau_{k,i}$. Then, we exchange the $\tau_{j,g}$and $\tau_{k,h}$ where $\tau_{j,g} = \tau_{k,i}$ and $\tau_{k,h} = \tau_{j,i}$. A crossover probability P_c will be introduced to replace the random selection of individuals, which means we determine whether the two rows need crossover by producing a random number compared with P_c. After repeating the above steps *M-1* times, a new matrix P_{INC} will be generated and continue the mutated.

4.1.4. Mutation operation

MO randomly exchanges two elements in one row. In other words, MO interchanges the $\tau_{s,j}$and $\tau_{s,k}$ in the same row from P_{INC}. Here, a mutation probability P_m is also introduced to replace the random selection of row. After repeating M times of mutation operation, the new matrix P_{INM} can be obtained. Besides, in order to preserve the optimal individual in the initial population, an individual with the largest fitness value in initial matrix P was selected and added it to the matrix P_{INM} at the *M+1* rows.

The row with the lowest fitness value from P_{INM} will be deleted and a new matrix P_{new} of size $M \times q$ will be obtained. Then iterating will continue until we find a stable individual with best fitness.

4.2. The deep learning trained by genetic algorithm

Eventually, we can get a queue set $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$ with best fitness after computing through Algorithm 2, which is the optimal scheme of task offloading in that current environment. If the tasks of MDs are changed, we have to get result by recomputing Algorithm 2, which spends too much time and influences the experience of users. It indicates that GA is not applicable to time-sensitive problems. Therefore, we try to apply DL-GA to solve this problem due to the faster computation speed.

In DL-GA, the result obtained by GA can be understood as experience. And the convolution neural network (CNN) is chosen to study the experience because of the excellent feature extraction ability [28]. After sufficient training, CNN can quickly solve the optimization schemes according the different parameters.

It is necessary to reshape our experience into a state matrix and label because of the application of CNN. The state matrix stores the features as input, and label stores an optimized offloading task queue as output. The state matrix with size of $\sqrt{n} \times \sqrt{n} \times 4$ was shown as in Figure 2.

Figure 2 shows that the state matrix can be understood as four layers and each layer is $\sqrt{n} \times \sqrt{n}$, $\sqrt{n} \in N+$. The first three layers are used to store attributes of tasks, and last layer stores relevant features of the BS. In order to train the CNN more efficiently, the range of values stored in this matrix will be set from 0 to 1. Therefore, the first layer stores $C'_s(i)$ which is normalization values of $C_s(i)$. The second layer stores $Cost'(i)$ which is normalization values of $Cost(i)$. The third layer stores L_i which determines whether it is

Algorithm 2 Offloading strategy based on GA in MEC

Input: Population (P)
Output: Individual with best fitness

Initialize population with matrix P and channel capacity of BSs
while $x < 800$ **do**
    Calculate fitness value by Algorithm 1
    Find individual with best fitness *Best*
    function RA
        Initialize a new empty matrix P_{IN}
        $pro = 0, i = 0$
        repeat
            randomly generated seed $P_{seed}, P_{seed} \in (0, 1)$
            repeat
                $pro+ = P_r(i)$
                if $pro > P_{seed}$ **then**
                    $P_{IN}.append(P(i))$
                    break
                else
                    $i + 1$
                end if
            until M **times**
            return P_{IN}
        until M **times**
    end function
    function CO
        repeat
            random a *seed*
            if $seed < P_c$ **then**
                Crossover operation
            end if
        until $M - 1$ **times**
    end function
    function MO
        repeat
            random a *seed*
            if $seed < P_m$ **then**
                Mutation operation
            end if
        until M **times**
    end function
    Calculate fitness value by Algorithm 1
    Find individual with best fitness value as *NewBest*
    if $NewBest = Best$ **then**
        $x+ = 1$
    else
        $x = 0$
    end if
end while
return individual with best fitness value *NewBest*

Figure 2. Matrix with collected features.

offloaded or not. L_i will be randomly assigned a value in the experiment. The fourth layer is used to collect the remaining channel capacity of m BSs. The first three columns of the fourth layer represent the remaining channel capacity of BSs set $\mathcal{B} = \{\beta_1, \cdots, \beta_m\}$, respectively. In other words, the first column of the fourth layer stores the $C_{res}(\beta_1)$, the second column of the fourth layer stores the $C_{res}(\beta_2)$, the third column of the fourth layer stores the $C_{res}(\beta_3)$, and the rest of the fourth layer has a value of 0.

To train the CNN better, it is important to collect enough state matrices and labels from the optimal results by GA. Next, we will introduce how to collect state matrices and labels from an assumed optimal result $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$. At the beginning, a state matrix $Ft_1 = [C_s'(i), Cost'(i), L_i^{\{1\}}, C_{res}^{\{1\}}(\beta_i)]$ and an empty label set $\mathcal{L} = \{\}$ are initialized. The $C_{res}(\beta_1)$, $C_{res}(\beta_2)$, $C_{res}(\beta_3)$ are initialized to 1 before offloading tasks, and then $\mathcal{L}$ adds a label τ_1 from set τ described as $\mathcal{L} = \{\tau_1\}$, which means the MEC system will offload task τ_1 in the current state Ft_1. When the first task τ_1 is offloaded, the system selects a BS with maximum remaining channel capacity to receive the task τ_1 and process. Then the working BS will subtract the channel capacity occupied by task τ_1, which is described as

$$C_{res}(\beta_i) = C_{res}(\beta_i) - \frac{C_s(\tau_1)}{C_B(\beta_i)}, \tag{13}$$

where β_i is the BS with maximum remaining channel capacity. $C_s(\tau_1)$ is the channel capacity occupied by task τ_1, $C_B(\beta_i)$ is the maximum channel capacity of β_i. After offloading the task τ_1, the L_i of task τ_1 changes from 1 to 0, and then a new state matrix $Ft_2 = [C_s'(i), Cost'(i), L_i^{\{2\}}, C_{res}^{\{2\}}(\beta_i)]$ with the same scale as Ft_1 is produced to store the new feature which is changed after offloading a task. And the label $\mathcal{L}$ also adds a new task τ_2 described as $\mathcal{L} = \{\tau_1, \tau_2\}$, which means that the system will offload the task τ_2 in the changed states Ft_2. This process will be described in Algorithm 3.

After repeating the above step, we will get a state matrix set $Ft = \{Ft_1, Ft_2, \cdots, Ft_{S-1}\}$ and the label $\mathcal{L} = \{\tau_1, \tau_2, \cdots, \tau_{S-1}\}$. And the element of both Ft and $\mathcal{L}$ correspond in order and are combined into a set of training sets as the input of the CNN. However, training CNN needs large datasets. Thus, we need to repeat Algorithm 3 to generate more datasets.

After collecting the relevant data, a CNN will be constructed with the structure shown in Figure 3. Here, the classification result represents the label of the task being offloaded, and then we can input the state matrix to get the best task queue to be offloaded. Since the CNN computation process does not require iteration and convergence, it can quickly compute a reasonable result. The task offloading process of DL-GA can be described by the algorithm 4

In Algorithm 4, this trained CNN may choose the MD that does not need to offload the task to the BS. This is because the optimization capability of the CNN relies on the large amount of experience gained from

Algorithm 3 datasets collection

Input: Optimal scheme τ from GA
Output: Fitness set F

Obtain individual with maximum fitness from Algorithm 2 (Supposing it is τ)
Initialize state matrix Ft_1
Store $C_s'(i), Cost'(i), L_i$ and $C_{res}(\beta_i)$ in the Ft_1
$\mathcal{L}.append(\tau_1)$ and $S = 1$
while ture **do**
    selecting the $C_{res}(\beta_i)$ with maximum remaining channel capacity
    if $C_{res}(\beta_i) \geq \frac{C_s(\tau_S)}{C_B(\beta_i)}$ **then**
        $C_{res}(\beta_i) = C_{res}(\beta_i) - \frac{C_s(\tau_S)}{C_B(\beta_i)}$
        $L_i = 0,\ L_i \in \tau_S$
        $\mathcal{L}.append(\tau_{S+1})$
        Copy matrix Ft_S to Ft_{S+1}
        Restore new $C_{res}(\beta_i)$ and L_i in Ft_{S+1}
        $S = S + 1$
    else
        break
    end if
end while

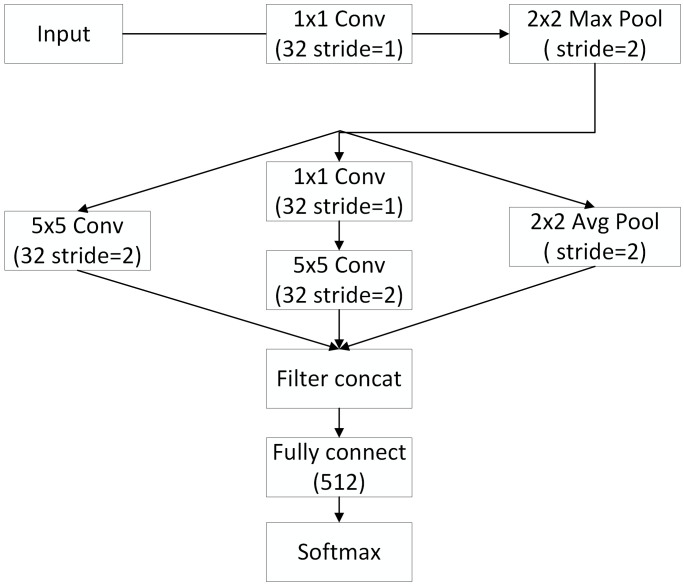


Figure 3. Matrix with collected features.

the GA rather than theoretical computations. In this case, the CNN's offloading strategy is designed to select a task with the minimum channel capacity to offload to the BS.

5. Simulation result

In this simulation, there are 100 MDs covered by 3 BSs and each BS can link all 100 MDs within the same distance. In other words, we ignore the effect of transmission distance on transmission rate. Since the $C_{res}(\beta_i)$ stored in matrix is normalized to a percentage value, each BS can be designed to have the same channel

Algorithm 4 Offloading strategy by CNN.

Input: An initialization state matrix Ft_1
Output: Optimal scheme $\mathcal{L}$

Initialize n UDs including $C'_s(i), Cost'(i), L_i$ and $C_{res}(\beta_i)$
Initialize m BSs including $C_B(\beta_i)$ of BSs
Reshape data above into matrix Ft_1
Initialize task queue $\mathcal{L}$ and $x = 1$
while ture **do**
    Input Ft_x to CNN and get the task τ_x. select the $C_{res}(\beta_i)$ with maximum remaining channel capacity
    if $L_i = 1,\ L_i \in \tau_x$ **then**
        if $C_{res}(\beta_i) \geq \frac{C_s(\tau_x)}{C_B(\beta_i)}$ **then**
            $C_{res}(\beta_i) = C_{res}(\beta_i) - \frac{C_s(\tau_x)}{C_B(\beta_i)}$
            $L_i = 0,\ L_i \in \tau_x$
            $\mathcal{L}.append(\tau_x)$
            $x = x + 1$
            Reshape new matrix Ft_x
        else
            return $\mathcal{L}$
        end if
    else
        Select task τ_j with minimum channel capacity
        if $C_{res}(\beta_i) \geq \frac{C_s(\tau_x)}{C_B(\beta_i)}$ **then**
            $C_{res}(\beta_i) = C_{res}(\beta_i) - \frac{C_s(\tau_x)}{C_B(\beta_i)}$
            $L_i = 0,\ L_i \in \tau_j$
            $\mathcal{L}.append(\tau_j)$
        else
            return $\mathcal{L}$
        end if
    end if
end while

bandwidth with no effect on the simulation results. We use a probability of 0.5 to determine the L_i of each task T_i. The parameter of channel bandwidth is assumed as $B = 5$ MHz [30], the average power of signal $S = 100$ mw, and the average power of noise $N = -100$ dbm [31]. All important simulation parameters and hyperparameter are shown in Table 2.

The diversity and validity of datasets are one of the key factors for the successful operation of DL-GA. We have to repeat GA in different initialization environments in order to collect sufficient datasets for the CNN.

Figure 4 shows one of the convergence curves of optimal GA-based results, which indicates that the fitness can converge efficiently. In this MEC system, fitness presents the system effectiveness defined by equation (7). Due to the large difference between the theoretical extremes and the random values, the fitness values usually appear small.

After training the CNN based on a sufficient dataset, the experimental procedure is described as follows. At first, we initialize the MDs and BSs. Then, we reshape them into an initialized state matrix. Finally, this matrix is fed to the trained CNN and an optimal task offloading scheme is obtained by the algorithm 4. Several factors are used to evaluate the DL-GA, including the total number of selected offloading tasks, the total cost of selected offloading tasks, the system effectiveness, and the solving time. To demonstrate the superiority

Table 2. Experience parameters and hyperparameter setting.

Parameters	Value
m	100
n	3
B	5 MHz
S	100 mw
N	-100 dbm
d_i	random$(0, 3 \times 10^8)$ Hz
f_i	1 GHz
J_i	8.9×10^{-12} J/cycle
$C_s(i)$	random$(0.175 \times 10^5, 0.5 \times 10^5)$
M	80
P_m	0.7
P_c	0.3
λ_{N_t}	0.7
λ_{C_v}	0.3
λ_t	0.3
λ_e	0.7
Activation function	Relu
Loss function	Crosse-entropy
Mini-Batch size	1024
Optimizer	Adam
Number of datasets	424536

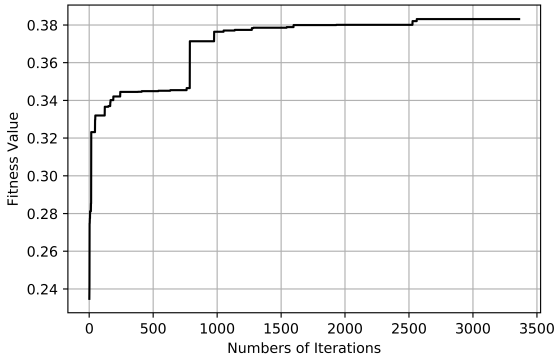


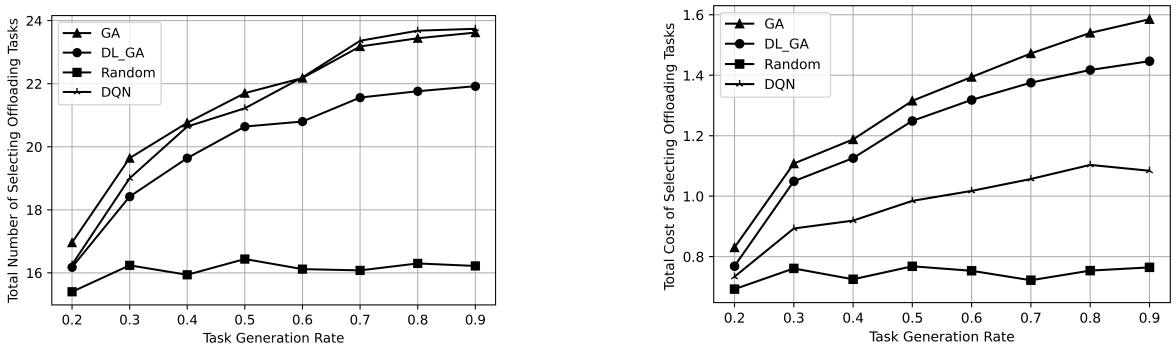
Figure 4. Convergence curve.

of DL-GA, GA, DQN, and random selection are used as comparisons. DQN is another commonly used task offloading algorithm, which consists of DL and RL. The following are the results of GA, DL-GA, DQN, and random selection based on the average value after 100 executions, respectively.

In GA, the value of L_i is randomly assigned as 0 or 1 with probability 0.5. In other words, the generation rate is set to 0.5, which means the experience of CNN learning is also based on the task generation rate of 0.5. Therefore, we set the axis X to the task generation rate from 0.2 to 0.9 to demonstrate the fitting ability of

DL-GA in the simulation experiment. When the task generation rate is 0.1, the channel capacity of all BSs is sufficient to offload all generated tasks simultaneously, which is not meaningful task offloading. Figures 5 and 6 shows the performance of GA, DL-GA ,DQN, and random choosing.

From Figures 5 and 6, as the task generation rate increases, the total number and cost of offloading tasks selected using GA, DL-GA, and DQN are much larger than those of random selection, which leads to much higher effectiveness of the three algorithms than the results of random selection. Although the total number of offloading tasks using DQN is larger than that of DL-GA, the total cost of offloading tasks using DQN is much smaller than that of DL-GA. This means that the offloading strategy of DQN is to offload as many tasks as possible without considering the cost of the tasks. Therefore, this DQN algorithm is not effective in reducing the energy consumption and delay of MD compared to DL-GA. The system effectiveness of DQN is similar to that of DL-GA due to the large weight of the number of tasks in the system effectiveness equation 7 in the experiment.



(a) The total number of selecting offloading tasks (b) The total cost of selecting offloading tasks

Figure 5. The total number and cost of offloading tasks in several algorithms.

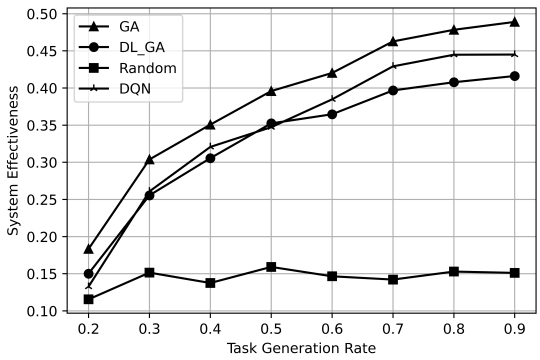


Figure 6. The system effectiveness of several algorithms.

In addition, the total cost and number of using GA is slightly more than that of using DL-GA. We can observe that the system efficiency of GA is higher than that of DL-GA, but the change is not significant. However, from Figure 7 and Table 3 we observe that the average solution time of GA is about 75–450 times

higher than that of DL-GA, which shows the superiority of DL-GA. The experimental results show that DL-GA exhibits faster algorithm execution than GA at the cost of slightly weaker optimization.

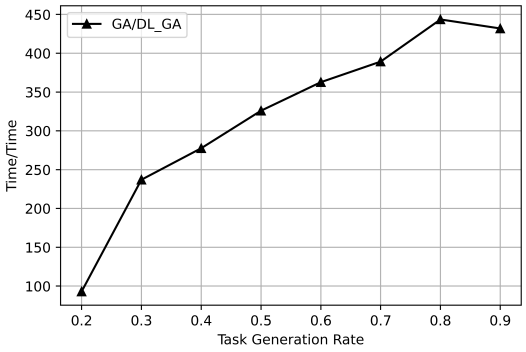


Figure 7. Ratio of average solving time of GA and DL-GA.

Table 3. Average solving time of GA and DL-GA.

Task Generation Rate	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
DL-GA(s)	0.165	0.119	0.123	0.129	0.125	0.128	0.129	0.129
GA(s)	15.36	28.14	34.15	42.06	45.22	49.70	57.21	55.72

In order to better demonstrate the optimization capability of DL-GA in different environments, we use the number of BSs as the environment variable to test the optimization capability of DL-GA, GA, DQN, and random selection. As the number of base stations increases, the total number of tasks that can be handled by the edge server increases. We increase the number of BSs sequentially from 1 to 6 and set the task generation rate to 0.5.

From Figure 8, we can see that the system effectiveness of all three algorithms, DL-GA, GA, and DQN, is much higher than random selection as the number of BSs increases. However, we can see from Figure 9 that the total cost of using DL-GA is much larger than that of DQN when the number of offloading tasks using the DQN strategy is slightly larger than that of using DL-GA. This indicates that when the number of BSs increases, DL-GA is still better than DQN in reducing energy consumption and delay.

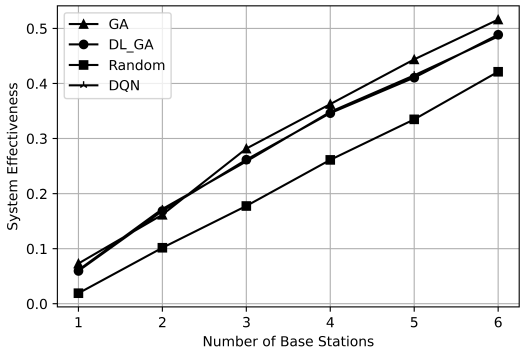
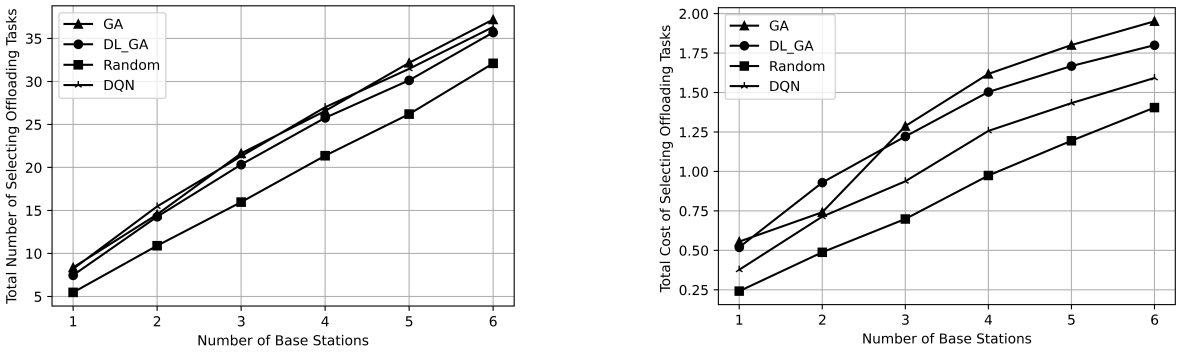


Figure 8. The system effectiveness at different numbers of BSs.



(a) The total number at different numbers of BSs (b) The total cost at different numbers of BSs

Figure 9. The total number and cost of offloading tasks at different numbers of BSs.

Overall GA has better optimization effect than DL-GA. However, when the number of BSs gradually increases, the number of offloading tasks that the BSs can accommodate at the same time also increases, so the complexity of the environment for the GA also increases, which can largely affect the iterative convergence time of the GA. Therefore, it can be seen from Figure 10 that the overall solution speed of GA tends to increase as the environment complexity increase. This experiment also demonstrates the superiority of the DL-GA algorithm.

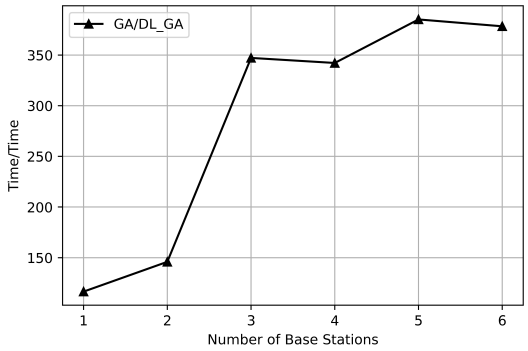


Figure 10. Ratio of average solving time of GA and DL-GA at different number of BSs.

6. Conclusion

Based on the consideration that the number of offloading tasks also affects the user experience, we aim to maximize the weighted sum of the total number of offloading tasks and the total cost. We apply DL-GA to solve this optimization problem. In this approach, we collect a large number of optimal results by GA and classify these results into states and labels. Then, we use these states and labels to train a CNN. After that, the trained CNN will quickly give the optimal solution. Simulation experiments show that the average solution time of DL-GA is much less than that of GA, and DL-GA still has similar optimization capability as GA, which proves the superiority of DL-GA for application in MEC environment.

However, DL-GA also has a drawback that the output of the neural network depends heavily on the training of the dataset. Therefore, once the input matrix structure changes, for example, a new environmental

factor is added, the optimization capability of DL-GA is much weaker. At this point, the CNN can only be retrained using the GA.

References

[1] Ren J, He Y, Huang G, Yu G, Cai Y et al. An Edge-Computing Based Architecture for Mobile Augmented Reality. In IEEE Network 2019; 33 (4): 162-169. doi: 10.1109/MNET.2018.1800132.

[2] Ren P, Qiao X, Huang Y, Liu L, Pu C et al. Edge AR X5: An Edge-Assisted Multi-User Collaborative Framework for Mobile Web Augmented Reality in 5G and Beyond. IEEE Transactions on Cloud Computing 2022; 10 (4): 2521-2537. doi: 10.1109/TCC.2020.3046128

[3] Zhao J, Li Q, Gong Y, Zhang K. Computation Offloading and Resource Allocation For Cloud Assisted Mobile Edge Computing in Vehicular Networks. In IEEE Transactions on Vehicular Technology 2019; 68 (8): 7944-7956. doi: 10.1109/TVT.2019.2917890

[4] Zhang Y, Wang P, Huang H, Zhu Y, Xiao D et al. Privacy-Assured FogCS: Chaotic Compressive Sensing for Secure Industrial Big Image Data Processing in Fog Computing. In IEEE Transactions on Industrial Informatics 2021; 17 (5): 3401-3411. doi: 10.1109/TII.2020.3008914

[5] Wang T, Shen X,Obaidat MS, Liu X, Wan S. Edge-Learning-Based Hierarchical Prefetching for Collaborative Information Streaming in Social IoT Systems. In IEEE Transactions on Computational Social Systems 2022; 9 (1): 302-312. doi: 10.1109/TCSS.2020.3041171

[6] Zhang L, Hao J, Zhao G, Wen M, Hai T et al. Research and Application of AI Services Based on 5G MEC in Smart Grid. IEEE Computing, Communications and IoT Applications (ComComAp) 2020; pp. 1-6. doi: 10.1109/ComComAp51192.2020.9398885

[7] Li B, Li Z, Chen F, Zhou H, Wang Y et al. Research on The Requirements and Deployment of 5G MEC in Power Grid Applications. IEEE 2nd International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA) 2021; pp. 351-355, doi: 10.1109/ICIBA52610.2021.9687891

[8] Qureshi SS, Ahmad T, Rafique K, Shuja-ul-islam. Mobile cloud computing as future for mobile applications - Implementation methods and challenging issues. IEEE International Conference on Cloud Computing and Intelligence Systems 2011; pp. 467-471. doi: 10.1109/CCIS.2011.6045111

[9] Huang Y, Lin B, Zheng Y, Hu J, Mo Y et al. Cost Efficient Offloading Strategy for DNN-Based Applications in Edge-Cloud Environment. IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), 2019, pp. 331-337, doi: 10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00056

[10] Bonomi F, Milito R, Zhu J, Addepalli S. Fog computing and its role in the Internet of Things. In Proc. 1st MCC Workshop Mobile Cloud Comput 2012; pp. 13–16.

[11] Bozorgchenani A, Mashhadi F, Tarchi D, Salinas Monroy S. A. Multi-Objective Computation Sharing in Energy and Delay Constrained Mobile Edge Computing Environments. in IEEE Transactions on Mobile Computing 2021; 20 (10): 2992-3005. doi: 10.1109/TMC.2020.2994232

[12] Li B, Chen T, Giannakis GB. Secure Mobile Edge Computing in IoT via Collaborative Online Learning. In IEEE Transactions on Signal Processing 2019; 67 (32): 5922-5935. doi: 10.1109/TSP.2019.2949504

[13] Zhuang W, Ye Q, Lyu F, Cheng N, Ren J. SDN/NFV-Empowered Future IoV With Enhanced Communication, Computing, and Caching. in Proceedings of the IEEE 2020; 108 (2): 274-291. doi: 10.1109/JPROC.2019.2951169

[14] Satyanarayanan M. The Emergence of Edge Computing. Computer 2017; 50(1), 30–39. doi:10.1109/mc.2017.9

[15] Ning Z, Dong P, Kong X, Xia F. A Cooperative Partial Computation Offloading Scheme for Mobile Edge Computing Enabled Internet of Things. In IEEE Internet of Things Journal 2019; 6 (3): 4804-4814. doi: 10.1109/JIOT.2018.2868616

[16] Bai T, Pan C, Deng Y, Elkashlan M, Nallanathan A et al. Latency Minimization for Intelligent Reflecting Surface Aided Mobile Edge Computing. In IEEE Journal on Selected Areas in Communications 2020; 38 (11): 2666-2682. doi: 10.1109/JSAC.2020.3007035

[17] Tiwary M, Puthal D, Sahoo KS, Sahoo B, Yang LT. Response time optimization for cloudlets in Mobile Edge Computing. Journal of Parallel and Distributed Computing 2018; 119: 81–91. doi:10.1016/j.jpdc.2018.04.004

[18] Wang S, Guo Y, Zhang N, Yang P, Zhou A et al. Delay-Aware Microservice Coordination in Mobile Edge Computing: A Reinforcement Learning Approach. In IEEE Transactions on Mobile Computing 2021; 20 (3): 939-951. doi: 10.1109/TMC.2019.2957804

[19] Chen Y, Zhang N, Zhang Y, Chen X, Wu W et al. Energy Efficient Dynamic Offloading in Mobile Edge Computing for Internet of Things. In IEEE Transactions on Cloud Computing 2021; 9 (3): 1050-1060. doi: 10.1109/TCC.2019.2898657

[20] Zhao T, Zhou S, Song L, Jiang Z, Guo X et al. Energy-optimal and delay-bounded computation offloading in mobile edge computing with heterogeneous clouds. In China Communications 2020; 17 (5): 191-210. doi: 10.23919/JCC.2020.05.015

[21] Yang Y, Hu Y, Gursoy MC. Deep Reinforcement Learning and Optimization Based Green Mobile Edge Computing. 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC) 2021; pp. 1-2. doi: 10.1109/CCNC49032.2021.9369566

[22] Li X, Zhao L, Yu K, Aloqaily M, Jararweh Y. A cooperative resource allocation model for IoT applications in mobile edge computing. Computer Communications 2021; 173: 183-191.

[23] Nath S, Wu J. Deep reinforcement learning for dynamic computation offloading and resource allocation in cache-assisted mobile edge computing systems. In Intelligent and Converged Networks 2020; 1 (2): 181-198. doi: 10.23919/ICN.2020.0014

[24] Tong Z, Deng X, Ye F, Basodi S, Xiao X et al. Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment. Information Sciences 2020; 537: 116-131. doi:10.1016/j.ins.2020.05.057

[25] Tang H, Wu H, Zhao Y, Li R. Joint Computation Offloading and Resource Allocation Under Task-Overflowed Situations in Mobile-Edge Computing. In IEEE Transactions on Network and Service Management 2022; 19 (2): 1539-1553. doi: 10.1109/TNSM.2021.3135389

[26] Liu C, Li K, Liang J, Li K. COOPER-SCHED: A Cooperative Scheduling Framework for Mobile Edge Computing with Expected Deadline Guarantee. In IEEE Transactions on Parallel and Distributed Systems 2019; pp. 1-1. doi: 10.1109/TPDS.2019.2921761

[27] Li W, Wang F, Pan Y, Lei Z, Liu J et al. Computing Cost Optimization for Multi-BS in MEC by Offloading. Mobile Networks and Applications 2022; 27: 236–248. doi:10.1007/s11036-020-01627-y

[28] Pan Y, Yang Y, Li W. A Deep Learning Trained by Genetic Algorithm to Improve the Efficiency of Path Planning for Data Collection With Multi-UAV. In IEEE Access 2021; 9: 7994-8005. doi: 10.1109/ACCESS.2021.3049892

[29] Muñoz O, Pascual-Iserte A, Vidal J. Joint allocation of radio and computational resources in wireless application offloading. 2013 Future Network & Mobile Summit 2013; pp. 1-10.

[30] Chen X. Decentralized Computation Offloading Game for Mobile Cloud Computing. In IEEE Transactions on Parallel and Distributed Systems 2015; 26 (4): 974-983. doi: 10.1109/TPDS.2014.2316834

[31] Chen X, Jiao L, Li W, Fu X. Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing. In IEEE/ACM Transactions on Networking 2016; 24 (5): 2795-2808. doi: 10.1109/TNET.2015.2487344