



# A Strategy of Multi-edge Collaborated Task Offloading Based on Edge Servers Placement in Metropolitan Area Networks

Haojie Peng<sup>1</sup>, Yongjie Xu<sup>1</sup>, Hui Li<sup>1,2,\*</sup>

<sup>1</sup>College of Electronics and Information Engineering, Nanjing University of Information Science and Technology, Nanjing, China

<sup>2</sup>College of Electronics and Information Engineering, Wuxi University, Wuxi, China

## Email address:

stud\_phj@163.com (Haojie Peng), 767275954@qq.com (Yongjie Xu), hitlihui1112@163.com (Hui Li)

\*Corresponding author

## To cite this article:

Haojie Peng, Yongjie Xu, Hui Li. A Strategy of Multi-edge Collaborated Task Offloading Based on Edge Servers Placement in Metropolitan Area Networks. *International Journal of Wireless Communications and Mobile Computing*. Vol. 10, No. 2, 2022, pp. 7-17.

doi: 10.11648/j.wcmc.20221002.11

**Received:** January 3, 2023; **Accepted:** January 16, 2023; **Published:** January 30, 2023

---

**Abstract:** At present, multi-access edge computing is applied to “cloud-edge-terminal” collaborative computing, in which the task of computing offloading is used to unload the computing tasks of “terminal” to “edge”, i. e. edge servers, but the placement of edge servers is often not considered, resulting in low efficiency of computing offloading. Therefore, this paper proposes a multi-edge collaborative task unloading strategy for the effective placement of edge servers in 5G metropolitan area networks (MAN). First, the strategy weights the average user delay, the average edge server load and the average edge server resource utilization, and improves the firefly algorithm (FA) to optimize the number and effective location of edge servers. Secondly, a multi-edge collaborative task unloading architecture is presented. The computing tasks in this architecture can be executed locally, on a local server, on a remote server or in the “cloud”. The delay and energy consumption of the four unloading modes are modeled respectively. Thirdly, a new variable, i. e. the maximum cooperation cost that the “terminal” can bear, is introduced to attract more remote edge servers to cooperate to complete the calculation of the “terminal” task. Furthermore, the immune particle swarm optimization (IPSO) algorithm was designed to solve the problems of the traditional PSO algorithm. The simulation results show that the improved firefly algorithm can find the optimal coordinates of the edge server, and then carry out the task unloading of the multi-edge server. Compared with the local offload strategy, immune algorithm (IA) and PSO, the total cost of the proposed IPSO algorithm is reduced by 66.7%, 54% and 45.5%, respectively. Therefore, the algorithm in this paper can improve the execution efficiency of computing tasks and effectively reduce the total cost of the whole system.

**Keywords:** Multi-edge Collaboration, Edge Server Placement, Task Offloading, Firefly Algorithm, IPSO Algorithm

---

## 1. Introduction

With the birth of 5G standard, 3GPP organization defines the combination of wireless communication and multi-access edge computing (MEC) [1]. MEC offloads user terminal tasks to radio access network (RAN) nearby connected edge servers (ES), thus reducing the time delay and energy consumption of terminal equipment, greatly improving the computing efficiency.

In recent years, MEC has become the research object of many scholars, and computational offloading is one of the most extensive research contents. Generally speaking, there are three optimization objectives of computational offloading, namely,

taking time delay as the optimization objective, taking energy consumption as the optimization objective, and taking the way of integrating time delay and energy consumption as the optimization objective [2, 3]. Li et al. used the gradient strategy algorithm in the queuing task scenario of single terminal and single edge server. Compared with greedy algorithm and exhaustive method, it can obtain a high-quality offloading scheme with shorter decision time and reduce the time cost by 50% compared with the gradient strategy algorithm [4]. Liu et al. cut the directed acyclic task sequence in a given order into basic units, and then decompose the basic units into sub-problems according to different scenarios to solve them,

thus realizing the minimum time consumption [5]. Ketykoi et al. modeled the multi-user offloading problem as an NP-hard problem, and adopted the knapsack model to optimize the entire resource allocation and load balancing problem [6]. Wang et al. proposed a scheme for interference management, which is to allocate communication resources and computing resources under the condition of minimizing interference [7]. In order to maximize resource utilization, Ndikumana et al. combined MEC servers and proposed a collaborative cache unloading strategy [8]. Xu et al. proposed a non-static offloading strategy and used the deep learning method to allocate limited resources [9]. Mehrabi et al. proposed a three-node MEC architecture that minimizes system energy consumption while meeting the requirements of task completion deadline and task dependence, and the results show that this architecture is superior to the comparative offloading methods [10]. Guo et al. have proposed an index mechanism, which enables the terminal device to find appropriate servers for calculation and uninstallation, and thus reduces energy consumption by 7% [11]. Guo et al. have established two models with high computational efficiency, which respectively have negligible and non-negligible execution time of edge server to obtain the optimal solution in closed form, and use Johnson's algorithm to obtain the sub-optimal solution with low complexity [12]. Based on the artificial fish swarm algorithm, Zhang et al. took the link state of the forward network and the back network into consideration, saving 30% energy consumption compared with the comparison algorithm [13]. Xian et al. proposed a master-slave MEC system architecture and adopted greedy selection algorithm to achieve rational allocation of computing resources. Experimental results showed that the offloading strategy reduced the total system cost [14]. Huang et al. proposed an online offloading framework based on deep reinforcement learning, which can effectively reduce the computational complexity [15].

To sum up, the above studies are usually conducted when the location of edge servers is given. In the process of uninstallation, the location of edge servers is related to user

delay and load balance of each edge server, and even affects the energy consumption of users and service providers. In addition, most scholars only consider single-edge collaboration or cloud-edge collaboration. Therefore, this paper carries out innovative research in three aspects:

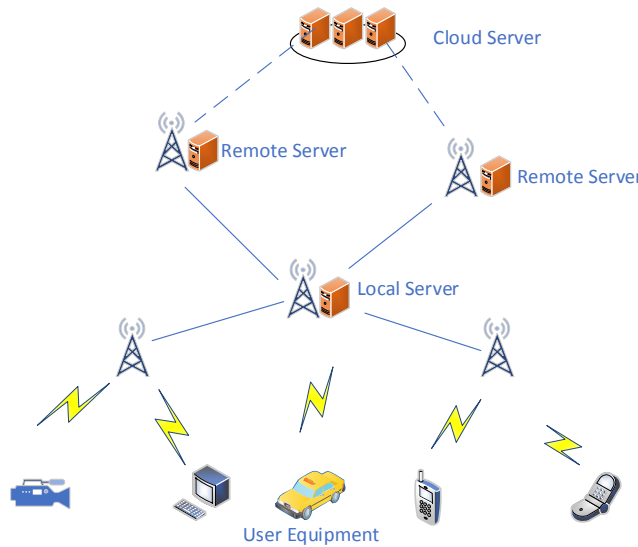
- 1) It first proposes to use the improved firefly algorithm for efficient and reasonable edge server placement, and then studies the efficient task unloading strategy based on it.
- 2) A new network architecture of multi-edge collaboration is adopted to make full use of idle resources of remote servers. Tasks in this architecture can be executed locally on a local server, on a remote server, or in the cloud server.
- 3) Different from traditional task attributes, a new variable, the maximum cooperation cost that the terminal can bear, is introduced to attract more remote edge servers to cooperate actively to complete the terminal task calculation. Further in view of the traditional particle swarm optimization (PSO) algorithm is easy to fall into local optimum prematurely, we adopted immune particle swarm optimization algorithm (IPSO) to solve the optimization goal, and got the best uninstall strategy, calculated the energy consumption and minimized the weighted sum of the delay.

## 2. System Model

### 2.1. Edge Server Placement Mode

#### 2.1.1. Edge Server Placement Problem

The problem of edge server placement description: Enter the number  $K$  of edge servers to be placed. Then use firefly algorithm to place edge servers near a given number of base stations, and assign a number of base stations to  $k$  edge servers according to the shortest distance principle. According to the problem description, three optimization models are given, namely, the user delay minimization model, the ES load balancing model and the ES resource utilization maximization model.



**Figure 1.** System model of multi-edge collaborated task offloading based on ES Placement.

As shown in figure 1, there are user devices, base stations, edge servers and cloud servers in the edge server placement model in this paper. The topological network where edge servers are placed can be considered as a dichotomous network  $G=(V \cup S, L)$  consisting of many base stations and a group of candidate positions of edge servers, where  $V$  is the collection of base stations,  $S$  is the set of edge servers,  $L$  is a collection of links for many base stations and edge servers. The description of the edge server in this paper is homogeneous, and the calculation and storage capacity are the same. A base station only propagates data to an edge server, the two base stations are not connected to each other, unless at least one of the two base stations is the location of the edge server, and all the base stations are covered by the edge server.

### 2.1.2. Problem Definition

A set  $\{S_1, S_2, \dots, S_k\}$  of  $k$  edge servers is used to represent  $K$  potential locations.  $\{r_1, r_2, \dots, r_R\}$  is the set of  $R$  RANs and  $\{u_1, u_2, \dots, u_U\}$  is the set of requests of  $r_j$  access users.  $d(r_j, s_i)$  represents the distance between  $r_j$  and  $s_i$ .

In 5C-MEC, the model of placing edge servers considers the access distance between RAN and its corresponding ES to be approximately as delay, and considers the access distance between RAN and its corresponding ES to be minimized. The average user delay is described as follows:

$$D[X] = \sum_{s_i \in S} \sum_{r_j \in C_{s_i}} d(r_j, s_i) / (|R| - K) \quad (1)$$

where  $X$  represents the placement of  $k$  ES locations and the allocation scheme of all RAN, and  $C_{s_i}$  represents the set of all RAN positions corresponding to a  $s_i$  under  $X$  scheme. A single ES load is defined as the sum of all transmission loads of its corresponding RAN.

$$W_{s_i} = \sum_{r_j \in C_{s_i}} T(r_j) \quad (2)$$

in which

$$T(r_j) = T(u_1) + T(u_2) + \dots + T(u_{|U_j|}) \quad (3)$$

where  $C_{s_i}$  represents the set of all RAN corresponding to an ES under placement scheme  $X$ , and  $T(\cdot)$  represents the load time of each user or RAN. The ES load balancing model is the standard deviation of each ES load.

$$W[X] = \sqrt{\sum_{s_i \in S} (W_{s_i} - \bar{W})^2 / |S|} \quad (4)$$

where  $\bar{W}$  represents the average load of all ES under a certain placement plan  $X$ . The maximum resource utilization model of ES is as follows:

$$H[X] = \sum_{s_i \in S} \frac{W_{s_i} - W_{TH}}{W_{TH}} \times 100\% / K \quad (5)$$

where  $W_{TH}$  indicates the load threshold of a single  $s_i$ . Combined with the model of user average delay minimization, ES load balancing and ES resource utilization maximization, the weighted system cost function is as follows:

$$P = 0.33(D_{nor}[X] + W_{nor}[X] + (1 - H_{nor}[X])) \quad (6)$$

where  $D_{nor}[X]$ ,  $W_{nor}[X]$  and  $H_{nor}[X]$  represent the average user delay, ES load standard deviation and ES average resource utilization after the normalization of a certain placement scheme  $X$ . In this paper, the average user delay, ES load standard deviation and ES average resource utilization data are linearly normalized [16], which can be described as follows:

$$v_{nor} = (v_i - v_{\min}) / (v_{\max} - v_{\min}) \quad (7)$$

## 2.2. Multi-Edge Task Offloading Model

### 2.2.1. Model Construction

Multi-edge collaborative task offloading problem description: edge servers are divided into local edge servers and remote edge servers based on the original "cloud-edge-terminal" system according to the physical distance between the terminal generating the task and the edge server. In this system, tasks can optionally be offloaded to remote edge servers as well as local edge servers and the cloud server. Because the remote edge server is closer to the user terminal than the cloud server, it greatly reduces the transmission delay and makes full use of idle computing resources. Multi-edge task offloading studies the user terminal offloads tasks to multi-edge servers including remote edge servers, and studies the user delay minimization model and system energy consumption minimization model.

### 2.2.2. Problem Definition

In the task unloading model, we assume that there are a total of  $N$  end users, and each user only produces one task, there are  $N$  tasks in total, and each task  $L_i$  can be represented as the set  $L_i = \{td_i, cd_i, rd_i, co_i^{max}\}$ , where  $td_i$ ,  $cd_i$ ,  $rd_i$  and  $co_i^{max}$  are respectively the amount of data transmitted for  $L_i$ , the amount of computation task  $L_i$ , the amount of data returned as a result, and the maximum cost that the local device can accept when requesting local or remote edge servers to cooperate to complete tasks.  $f_1$ ,  $f_2$  and  $f_3$  are the computing capabilities of user terminals, edge servers, and cloud servers respectively. For the convenience of the study, the computing power of the local edge server and the remote edge server is assumed to be the same.

When a task is selected for local calculation, the delay caused is mainly calculation delay, and there is no transmission delay. The energy consumption is the energy consumption of the processor during the calculation of the

task.  $t_i^1$  represents the local computing delay, and  $e_i^1$  represents the energy consumption of the local device.

$$t_i^1 = cd_i / f_1 \quad (8)$$

$$e_i^1 = p_1 \times t_i^1 \quad (9)$$

where  $p_1$  is the power of the local device processor. When a task is uninstalled to the local server, the delay is caused by the transmission delay, calculation delay, and results return delay. The power consumption is mainly generated when data is uploaded and downloaded and when the server processes tasks. According to Shannon theory, the wireless transmission rate of the local device and server is as follows:

$$c_1(p) = B_1 \lg \left[ 1 + \frac{\rho_1^2 (1/d_1)^{\rho_2} p}{B_1 \rho_0} \right] \quad (10)$$

where  $B_1$  is the communication bandwidth between the local device and the local server;  $\rho_1$  and  $\rho_2$  are the fading factor and loss factor of the path;  $\rho_0$  is the noise power spectrum density;  $d_1$  is the nearest distance between the local device and the local edge server.  $t_i^{e1}$  is latency of uninstalling tasks to the local server.

$$t_i^{e1} = \frac{td_i}{c_1(p_s)} + \frac{cd_i}{f_2} + \frac{rd_i}{c_1(p_x)} \quad (11)$$

where  $p_s$  and  $p_x$  represent the upload power and download data of the task. And  $e_i^{e1}$  represents the energy consumption of uninstalling the task to the local server.

$$e_i^{e1} = p_s \times \frac{td_i}{c_1(p_s)} + p_e \times \frac{cd_i}{f_2} + p_x \times \frac{rd_i}{c_1(p_x)} \quad (12)$$

where  $p_e$  indicates the calculated power of the local server. When a task is uninstalled from a local server to a remote server, the delay mainly includes transmission delay, calculation delay and return delay. The energy consumption includes computing energy consumption, upload energy consumption, and download energy consumption. The wireless transmission rate of the local device and server is as follows:

$$c_2(p) = B_2 \lg \left[ 1 + \frac{\rho_1^2 (1/d_2)^{\rho_2} p}{B_2 \rho_0} \right] \quad (13)$$

where  $d_2$  is the Euclidean distance between the two servers. The delay for uninstalling a task to a remote server is as follows:

$$t_i^{e2} = \frac{td_i}{c_2(p_s)} + \frac{cd_i}{f_2} + \frac{rd_i}{c_2(p_x)} \quad (14)$$

If the task is uninstalled to a remote server, the energy consumption is as follows:

$$e_i^{e2} = p_s \times \frac{td_i}{c_2(p_s)} + p_e \times \frac{cd_i}{f_2} + p_x \times \frac{rd_i}{c_2(p_x)} \quad (15)$$

When unloading the task to the cloud, the delay mainly includes: transmission delay, calculation delay and return delay. The energy consumption includes computing energy consumption, upload energy consumption, and download energy consumption. The wireless transmission rate between the local device and the cloud server is as follows:

$$c_3(p) = B_3 \lg \left[ 1 + \frac{\rho_1^2 (1/d_3)^{\rho_2} p}{B_3 \rho_0} \right] \quad (16)$$

where  $B_3$  is the communication bandwidth between the local device and the cloud server, and  $d_3$  is the distance between the local device and the cloud server. The delay for uninstalling a task to the cloud server is as follows:

$$t_i^c = \frac{td_i}{c_3(p_s)} + \frac{cd_i}{f_3} + \frac{rd_i}{c_3(p_x)} \quad (17)$$

The energy consumption for uninstalling a task to the cloud server is as follows:

$$e_i^c = p_s \times \frac{td_i}{c_3(p_s)} + p_e \times \frac{cd_i}{f_3} + p_x \times \frac{rd_i}{c_3(p_x)} \quad (18)$$

When a computing task requires large computing resources, if the task is executed locally, the terminal energy consumption will increase. If the task is unloaded to the "cloud", since the "cloud" is far away from the user layer, the task completion delay will be large. Therefore, this paper introduces collaborative computing into the unloading calculation model. In order to encourage more edge servers to share computing resources, terminal users should pay a certain cost to attract servers to help users complete tasks' computing.  $co_i$  is proportional to the amount of task data because a task with a larger amount of data requires more transmission time. Therefore, the calculation paradigm for cooperative calculation cost  $co_i$  is as follows:

$$co_i = ktd_i \quad (19)$$

where  $k$  is the scale coefficient, and represents the unit price of the task that the user requests the server to cooperate. In summary, the total delay and energy consumption of task unloading can be expressed as follows:

$$T_{\text{sum}} = \sum_{i=1}^M t_i^1 + t_i^{e1} + t_i^{e2} + t_i^c \quad (20)$$

$$E_{\text{sum}} = \sum_{i=1}^M e_i^1 + e_i^{e1} + e_i^{e2} + e_i^c \quad (21)$$

The total cost of the system is defined as the objective function, which is expressed as the weighted sum of delay and energy consumption. Since cooperation costs should be considered, the objective function should follow certain

constraints, so the complete objective function is as follows:

$$g = w_t T_{\text{sum}} + (1 - w_t) E_{\text{sum}} \quad (22)$$

$$s.t. \quad co_i \leq co_i^{\max} \quad (23)$$

Equation (22) is the optimization objective of this paper, and (23) represents the cost constraint paid by the user when the task is unloaded on the edge computing server. If the calculation task is sensitive to delay, the time weight  $w_t$  can be appropriately increased; if the energy consumption is expected to be reduced, the  $w_t$  can be reduced.

### 3. Edge Server Placement Method Based on Firefly Algorithm

First, firefly algorithm (FA) [17] simulates individual fireflies in nature by searching points in space. Secondly, the search and position updating process in the optimization process was simulated as the mutual attraction and movement process among fireflies in nature. The objective function value of the problem to be solved is simulated into the luminance information of individual firefly. Finally, the process of finding the optimal feasible solution in the iterative process was simulated as the process of individual firefly survival of the fittest. Clustering problem is an important application field of FA, which has excellent performance. In this paper, the FA was improved to obtain the minimum average time delay of users and the scheme of ES placement and RAN allocation with good system performance.

#### 3.1. Definition of Firefly Individual

According to the above analysis, the placement of edge server is a discrete optimization problem. In this paper, an individual firefly is regarded as the deployment of  $k$  edge servers, and a reasonable number of firefly population is set for iterative optimization.

#### 3.2. Firefly Luminescence Intensity

Firefly luminescence intensity is related to the value of the objective function to be optimized. In this paper, FA was improved and the average time delay of all users under the ES placement and RAN assignment scheme was set as the firefly luminescence intensity. The greater the individual luminescence intensity, the smaller the average time delay of users under the placement and assignment scheme.

#### 3.3. Firefly Position Update

The position of the first occurrence of  $n$  firefly individuals is random,  $n$  fireflies randomly select the locations of  $k$  edge servers in turn, traverse RAN, select the nearest edge server to match, and obtain  $n$  individual luminescence intensity.  $n/2$  individuals with high brightness are reserved for the second iteration, so that the individual cluster center moves, and firefly position update is completed.

#### 3.4. Fitness Curve

Fitness function controls the iterative updating of population. 5G network combined with MEC pays more attention to user service. In this paper, the average user time delay is used as the fitness value of FA.

#### 3.5. Description of the FA Algorithm

The steps of the FA algorithm are as follows:

*Step 1:* Initialize the relevant data, including the number of clusters  $K$ , the number of fireflies  $n$ , whether it is the first iteration  $T$ , and the number of iterations  $m$ .

*Step 2:* In the first iteration, all  $n$  fireflies randomly selected ES addresses, and RAN was assigned to ES successively according to the principle of shortest distance allocation to get the luminescence intensity of fireflies.

*Step 3:* Keep the first two fireflies with strong luminous intensity among  $n$  fireflies, and update the firefly population. Select one of the first two individuals with high brightness each time, and complete elite individual replication. Iterate  $n$  times to generate elite firefly population.

*Step 4:* Firefly position move is the original position change of all ES. RAN is traversed. According to the condition that the longitude and latitude of RAN and each ES are less than 0.01 degrees, one of the RAN that meets the condition is randomly selected as the position of the new ES, and the ES position move is completed.

*Step 5:* Repeat Step 4  $n$  times to complete the position movement of firefly population and get the luminescence intensity of firefly population.

*Step 6:* Compare whether the number of iterations meets the maximum number of iterations. If yes, go to Step 7; otherwise, go to Step 3.

*Step 7:* Output the brightest firefly individual of the last iteration and the average user delay, ES load balancing standard deviation and ES average resource utilization in this scheme.

### 4. Multi-edge Offloading Strategy Based on Edge Server Placement

PSO has always been favored by scholars to solve objective optimization problems, but the traditional PSO is prone to premature convergence and falling into local optimum in practical engineering optimization. To overcome these shortcomings, this paper introduces the immune mechanism in biology into particle swarm optimization, and optimizes the objective function based on immune particle swarm optimization.

#### 4.1. Traditional PSO Algorithm

It is a candidate solution to the particle optimization problem. Particles have two parameters, namely, velocity and position. Each particle has its fitness function, which is generally an objective optimization function, which is used to evaluate the quality of particles. In the iteration, the particle updates itself through the individual extreme value  $pbest$  and

the global extreme value  $gbest$  until it meets the conditions. The following formulas are the particle velocity and position update criteria.

$$v_i^d = wv_i^{d-1} + c_1r_1(pbest_i^d - x_i^d) + c_2r_2(gbest_i^d - x_i^d) \quad (24)$$

$$x_i^d = x_i^{d-1} + v_i^d \quad (25)$$

where  $w$  is the inertia factor, which indicates the particle's trust in its own motion state.  $c_1$  and  $c_2$  are learning factors, and  $r_1$  and  $r_2$  are two independent random numbers with values of  $[0,1]$  [18].

#### 4.2. Coding Methods

This paper adopts four offloading methods, with 1, 2, 3 and 4 representing local offloading, local server offloading, remote server offloading and cloud server offloading. Because there are  $n$  tasks and the priority between tasks needs to be reflected, this paper adopts 2-bit floating-point number encoding. The integer part indicates the task number, the first decimal place indicates the unloading mode, and the second decimal place indicates the priority of the task—that is, the larger the numerical value, the more priority the task is to be executed.

#### 4.3. Fitness Function

It can be known from the system model that the optimization objective is to minimize the total cost of the system, so this paper takes the objective function as the fitness function. The larger the fitness function, the lower the total cost of the system and the more effective the unloading scheme.

$$fitness = -[wT_{sum} + (1-w)E_{sum}] \quad (26)$$

#### 4.4. Immune Particle Swarm Optimization

Based on the PSO algorithm, the artificial immune algorithm is introduced, especially the immune memory mechanism and the concentration regulation mechanism in the immune system. Immune memory mechanism is defined as: plasma cells secreting antibodies that can bind to antigens in the immune system are stored as memory cells. When the body encounters the same antigen again, memory cells will differentiate into plasma cells and produce a large number of antibodies. In this paper, the idea is introduced into the PSO algorithm, and the particles with high fitness generated by each iteration are saved as memory cells. When encountering particles with low fitness, memory cells can be substituted.

Different from the traditional particle swarm optimization algorithm, the quality of particles (antibodies) is determined by affinity as follows:

$$p(x_i) = \alpha f(x_i) + (1-\alpha)d(x_i) \quad (27)$$

where  $\alpha$  is the scaling factor,  $p(x_i)$  is the affinity value,  $f(x_i)$  is the fitness value and  $d(x_i)$  is the concentration value. In order to avoid falling into the local optimal solution, we should try

our best to ensure the diversity of the population and prevent a large number of particles with high affinity value. Therefore, the variable  $d(x_i)$  concentration is introduced. The calculation method is shown in (21) and (22).

$$\rho(x_i) = \sum_{j=1}^N \begin{cases} 1 & 0.95 < \frac{x_i}{x_j} < 1 \\ 0 & \text{else} \end{cases} \quad (28)$$

$$d(x_i) = \rho(x_i)/N \quad (29)$$

where  $\rho(x_i)$  represents the similarity, if each dimension of the particle is similar, it is judged to be similar particle.  $N$  is the number of particles in the solution space.

In this paper, the particle population is divided into two populations  $a$  and  $b$  according to the maximum particle concentration. Population  $a$  is particles with high affinity. Let these particles iterate according to the traditional particle swarm optimization method, while population  $b$  is particles with low affinity. Vaccinate the particles of population  $b$ , and let the particles with high affinity in memory cells increase in value to replace them. Here, the number of particles in population  $a$  and  $b$  is obtained by the following two formulas:

$$N(a) = d_{\max} N \quad (30)$$

$$N(b) = N - N(a) \quad (31)$$

where  $N(a)$  and  $N(b)$  represent the number of particles in population  $a$  and  $b$ , respectively, and  $d_{\max}$  is the maximum concentration of particles. Controlling the number of particles in two populations is conducive to maintaining the diversity of particles and making the population evolve in a good direction.

#### 4.5. Algorithm Flow

Flow chart shows the corresponding relationship between immune particle swarm optimization and unloading problem. When using immune particle swarm optimization algorithm to solve the problem, the objective optimization problem is abstracted as antigen, and the candidate solution of the problem is abstracted as antibody (particle). The optimal antibody is searched by heuristic search in the solution space by solving the affinity, and the optimal antibody is decoded to output the optimal solution of the objective problem. Antibody (particle)  $X = \{x_1, x_2, \dots, x_b, \dots, x_N\}$  is used to indicate the result of task scheduling, and when  $x_i=1$ , the task is executed locally.

**Table 1.** Relationship between IPSO algorithm parameters and unloading policy ones.

IPSO Algorithm Parameters	Unloading Policy Ones
IPSO Algorithm	Unloading Optimization Problem
Particle Position	Unloading Strategy
Fitness Value	Total System Cost

The steps of the IPSO algorithm are as follows:

*Step 1:* The particle swarm with an initial population of  $N$  includes position and velocity.



*Step 2:* The affinity of each particle is calculated according to (27), and the particle whose affinity is equal to the extreme value is regarded as a memory cell as a vaccine.

*Step 3:* Order the particles in descending order of their affinity.

*Step 4:* Population  $a$  and  $b$  are divided into two groups according to the maximum particle concentration. Population  $a$  is iterated according to (24) and (25), and population  $b$  is vaccinated.

*Step 5:* Population  $a$  and population  $b$  merge to form population  $c$ .

*Step 6:* The population is updated according to the velocity and position formulas of the PSO algorithm.

*Step 7:* Judge whether the group extreme value meets the loop exit condition. If so, output the result; otherwise, go to Step 2.

#### 4.6. Algorithm Complexity Analysis

The population of IPSO algorithm is set as  $M$  and the number of iterations is set as  $Y$ . Firstly, the time complexity analysis of the operation of a single iteration is carried out, and then it is extended to the whole process. Firstly, the population is initialized and the affinity of each antibody is calculated, and its time complexity is  $O(M)$ . The affinity value of the antibody is sorted by the way of selection from largest to smallest, and its time complexity is  $O(M^2)$ . The velocity and position of each antibody are updated according to (24) and (25), and its time complexity is also  $O(M)$ . To sum up, the complexity of IPSO algorithm is  $O(YM^2)$ .

## 5. Experimental Analysis

In this paper, Python 3.10 is used for simulation experiments. First, simulation experiments are conducted for placing efficiently edge servers, and the optimal number and location of placed edge servers are obtained. Core parameters were set as follows: maximum iteration times 120, population number  $n=4$ ,  $W_{TH}=1 \times 10^6$  min. Moreover, three major algorithms, K-Means [19] algorithm, Top-K algorithm [20] and Random algorithm [21], are compared with the improved firefly algorithm in this paper.

Scene simulation: based on the system model, a rectangular area of  $50 \times 50$  km<sup>2</sup> is established as the simulation scene of the experiment. The unit of horizontal and vertical coordinates is km. In the simulation scene, there are 3000 base stations obeying the Poisson Distribution. It is assumed that these base stations can cover all users in this area. Based on the reference [18], it is assumed that the data amount transmitted by user tasks is randomly generated at [1,600] KB, and the returned data amount is randomly generated at [0.1,100] KB.

### 5.1. Edge Server Placement

Keep the number of RAN at 3000, increase the number of ES at intervals of 5, and explore the change of user average delay, as shown in figure 2. The horizontal axis is the number of ES, and the vertical axis is the approximate average time delay of users accessing RAN. With the increase of the

number of ES, the average time delay of users of each algorithm decreases. The reason is that when the total number of RAN is unchanged, with the increase of the number of ES, RAN can be assigned to ES closer than the original, so the average time delay of users decreases. The average user delay of FA algorithm is the lowest, followed by Random and Top-K algorithms, and the delay effect of K-Means is the worst.

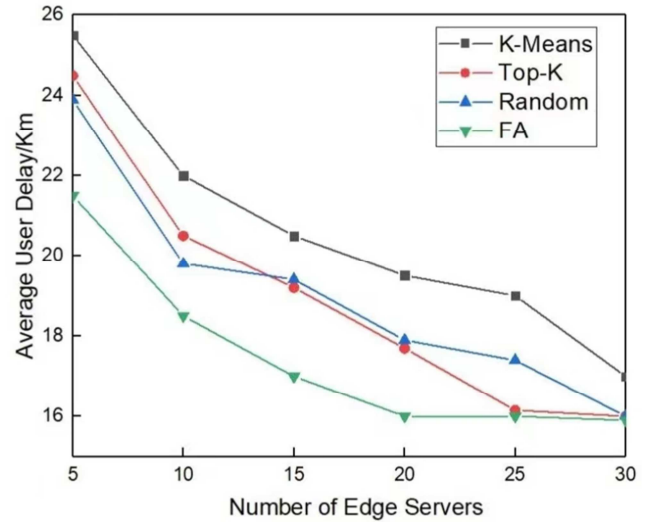


Figure 2. Average delay of users vs. number of ES.

Keep the number of base stations at 3000, and increase the number of ES at an interval of 5. Explore the system performance of the four algorithms, as shown in figure 3. In figure 3, the horizontal axis is the number of ES, and the vertical axis is the average user delay, ES load balancing amount, and ES resource utilization. Each weighted value is 0.33 to obtain the system performance value. In figure 3, the lower the system performance value under the edge server placement model, the better the system performance. The four algorithms rank FA, Random, Top-K, and K-Means from best to worst.

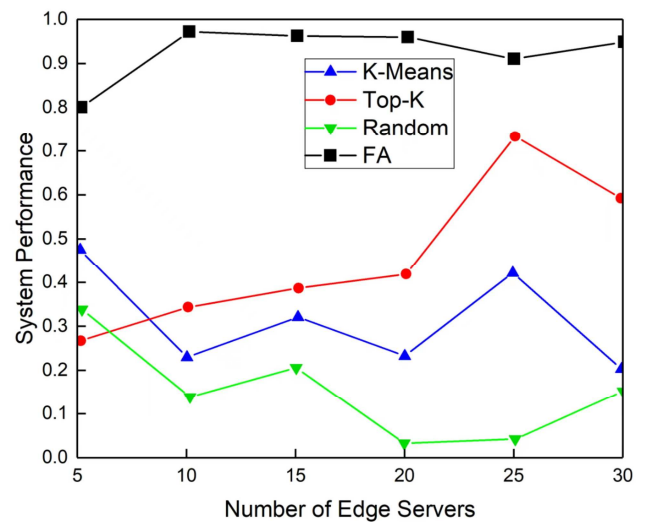


Figure 3. System performance of four algorithms vs. number of ES.

In this paper, the objective function of placing edge server is to iterate the average time delay of users, and the improved firefly algorithm proposed improves the system performance well, which is 9.4% and 22.6% higher than the Random algorithm and Top-K algorithm respectively. The number of edge servers placed in the optimal system performance obtained by the improved firefly algorithm is 20, and their coordinate positions are shown in table 2.

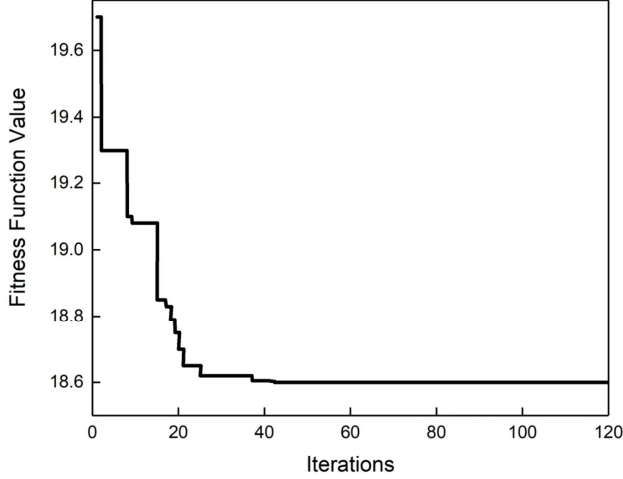


Figure 4. The curve of fitness vs. number of iterations.

Table 2. Position of ES.

Number	Abcissa/km	Ordinate/km
1	3.924	42.127
2	6.154	36.582
3	10.362	47.051
4	12.802	8.159
5	13.337	38.944
6	15.887	27.050
7	17.125	10.573
8	20.419	43.588
9	21.569	37.457
10	23.371	5.944
11	25.114	36.813
12	27.334	25.687
13	30.025	13.154
14	34.125	40.573
15	37.441	22.891
16	39.543	11.251
17	41.228	6.337
18	43.569	37.457
19	45.213	25.664
20	48.697	30.561

## 5.2. Multi-edge Offloading Based on Servers Placed

To verify the performance of the offloading policies proposed in this paper, simulation results of local offloading policies, immune algorithm (IA)-based policies and traditional PSO based policies are compared in this section. The basic parameter setting of the algorithm is based on the parameter example provided in literature [22-24], and certain adjustments are made in this paper according to the simulation environment. The parameters related to algorithm, task unloading and wireless communication link are listed in tables

3~5.

Table 3. Algorithm-related parameters.

Parameters	Values
Learning factor $c_1$	2
Learning factor $c_2$	2
Scale factor $\alpha$	0.4
Inertia factor $\omega$	0.4
Number of population $N$	60
Number of iterations $K$	100

Table 4. Task unloading related parameters.

Devices	Parameters	Values
Local devices	Computing efficiency $f_1/\text{GHz}$	1
	Bandwidth $B_1/\text{MHz}$	10
	Bandwidth $B_2/\text{MHz}$	75
	Calculated power $P1/\text{W}$	0.5
		60
Edge servers		100
	Computing efficiency $f_2/\text{GHz}$	10
Cloud server	Computing efficiency $f_3/\text{GHz}$	100

Table 5. Wireless communication links related parameters.

Parameters	Values
Noise density $\rho_0/\text{dBm}$	-100
Fading factor $\rho_1$	4
Loss factor $\rho_2$	2

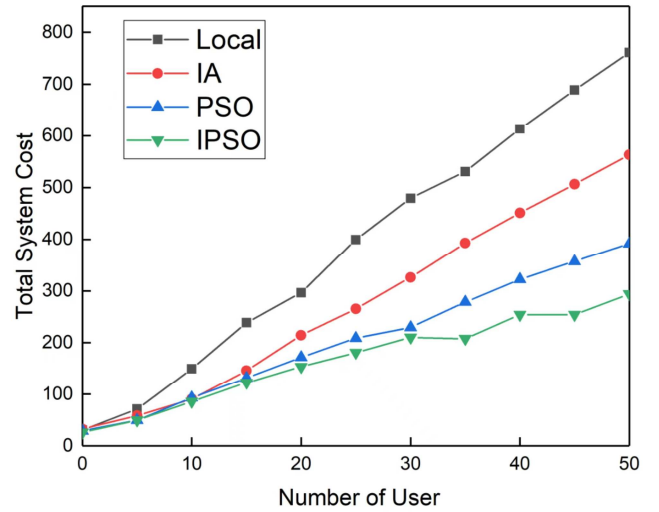


Figure 5. Total system cost vs. number of users.

Figure 5 shows the impact of the increasing number of user devices on the total cost of the system. In figure 5, the horizontal coordinate is the number of users. The ordinate represents the total system cost, and its value represents the weighted sum of time delay and energy consumption. As is the ordinate in figures 4~6. In this paper, there is a small amount of energy consumption in the standby state of the device server when setting the simulation environment, so the curve does not return to the origin. As the number of tasks generated by users increases, the system consumes more energy and delays during task offloading. However, it can be seen from the figure that compared with the LOCAL offloading strategy (i.e. all tasks generated by terminals are executed locally), the



IA algorithm and the traditional PSO algorithm, the total cost of the system is the smallest. Compared with the other three strategies, the total cost of the system is increased by 66.7%, 54% and 45.5%. It is proved that this strategy has higher effectiveness.

Figure 6 describes the impact of the increase in task computation on the total cost of the system. The number of tasks is 5000 GB. With the increase of task calculation, the total cost of the system also increases, but the total cost of IPSO algorithm is the smallest. When the task computation amount is less than 6000 GB, the total system cost of PSO algorithm is not much different from that of IPSO algorithm and IA algorithm. But when it is greater than 6000 GB, IPSO algorithm is obviously better than PSO algorithm and IA algorithm. This is because that the PSO algorithm is easy to converge to the local optimal, and in the future search process, the unloading effect is not ideal.

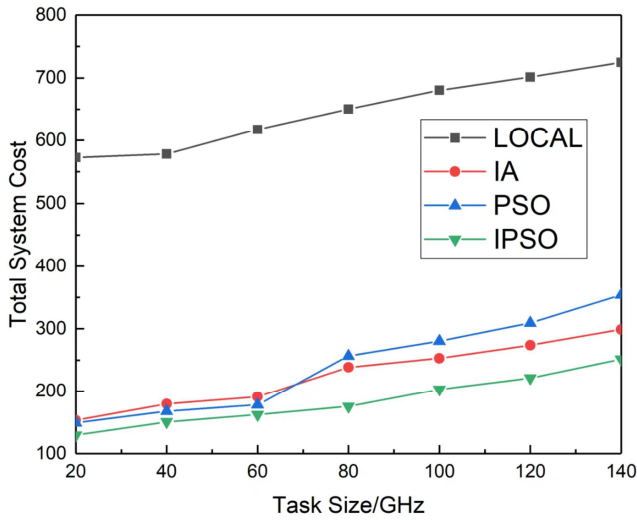


Figure 6. Total system cost vs. task size.

In order to explore the total cost of the number of remote edge servers on the system, the number of local servers was set as 1 in this experiment. As can be seen from figure 7, the total cost of the system is the lowest when there are 5 edge servers. At the same time, when there are more than 5 servers, the total cost of the system will increase slightly with the increase of the number of remote servers, because more remote edge nodes will make the computing task offloading more dispersed, and the system energy consumption and transmission delay cost will increase, so the curve will fluctuate a little. Local offloading is almost unaffected by the increasing number of servers.

Figure 8 shows the influence of the maximum user cooperation cost on the total system cost. The proportion coefficient  $K$  of this experiment task is 0.1. According to equation (16), the cost required by the user to request the edge server to cooperate to complete the task is related to the task data volume, which is randomly generated at [1,600] KB. Therefore, the maximum cost that the user can bear when setting the horizontal coordinate increases in the interval [10, 60].

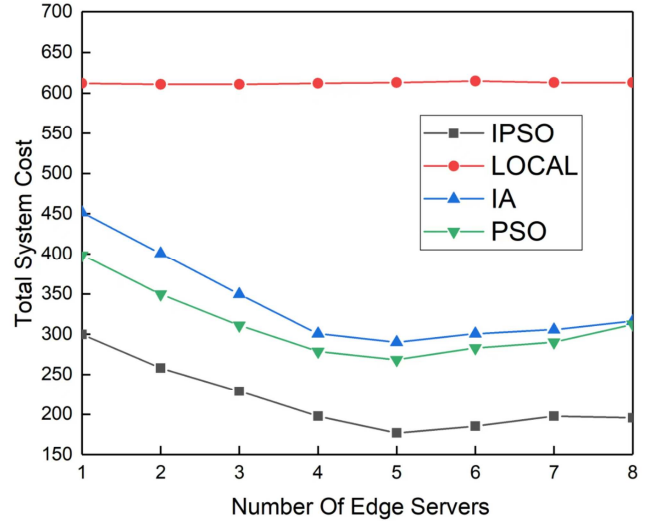


Figure 7. Total system cost vs. number of servers.

The horizontal axis in figure 8 represents the maximum cost to the user, which is related to the amount of data for the task, but it is a dimensionless physical quantity because its physical meaning represents the price paid by the user to purchase server resources. As we can be seen from figure 8, with the increase of the maximum cost that the user can afford, the total system cost decreases, because the user can offload more tasks to the edge server for execution and reduce the computing delay and energy consumption. However, the local offloading tasks remain basically unchanged because the tasks are processed locally and there is no need to request the edge server for cooperative calculation. The total cost of the system is almost not affected with a local task-executing.

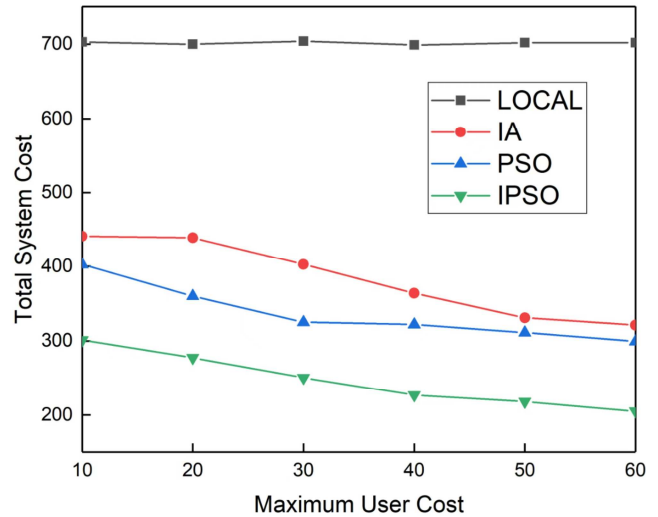


Figure 8. Total system cost vs. maximal user cost.

## 6. Conclusion

In this paper, we take the edge server placement as the premise to study the multi-edge task unloading strategy for the first time, and combine the three factors of user delay, ES load balancing and ES resource utilization in 5G network. Based on

the improved firefly algorithm, we take the average user delay in 5G network as the iterative target, the global optimal solution is searched, and the location and number of edge servers in the simulation environment are obtained. Then, based on the task computing and unloading mode of collaboration among user devices, local servers, remote servers and cloud servers, the delay and energy consumption modeling became the total cost of the system, and the IP SO algorithm was used for task unloading. Simulation results show that this strategy can improve the efficiency of task execution and reduce the total cost of the system effectively.

However, this paper does not consider the interference of communication links, nor does it consider the limited communication and computing resources of the edge server. In the subsequent research, the above deficiencies will be considered, and the algorithm of deep reinforcement learning will be adopted to continue the study of the task unloading strategy.

## Acknowledgements

This paper was supported by the National Natural Science Foundation of China (61661018), the Youth Foundation Project of Jiangsu Basic Research Program (BK20210064), the Doctoral Project of Entrepreneurship and Innovation of Jiangsu Province (JSSCBS20210863) and the Research Initiation Fund Project of Binjiang College of Nanjing University of Information Science & Technology (2021r006). H. Li is the corresponding author of this article.

## References

- [1] Bilal K., Khalid O., Erbad A., et al. Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers [J]. *Computer Networks*, 2018, 130: 94-120.
- [2] Zhang Y., Liang Y., Yin M., et al. A review of computational unloading schemes in multi-access edge computing [J]. *Chinese Journal of Computers*, 2021, 44 (12): 2406-2430.
- [3] Xie R., Lian X., Jia Q., et al. Overview of Mobile Edge Computing Unloading Technology [J]. *Journal of Communication*, 2018, 39 (11): 138-155. (in Chinese).
- [4] Li Y., Yang C., Deng M., et al. A dynamic resource optimization scheme for MEC task offloading based on policy gradient [C]. 2022 IEEE 6th Information Technology and Mechatronics Engineering Conference, Chongqing, 2022: 342-345.
- [5] Liu Z. and Zhao J. Optimized task offloading policy in given sequence in mobile edge computing [C]. IEEE 6th International Conference on Computer and Communications, Chengdu, 2020: 1656-1660.
- [6] Ketyko I., Kecskes L., Nemes C., et al. Multi-user computation offloading as multiple Knapsack problem for 5G mobile edge computing [C]. European Conference on Networks and Communications, Athens, 2016: 225-229.
- [7] Wang C., Yu F. R., Liang C., et al. Joint computation offloading and Interference management in wireless cellular networks with mobile edge computing [J]. *IEEE Transactions on Vehicular Technology*, 2017, 66 (8): 7432-7445.
- [8] Ndikumana A., Ullah S., Leanh T., et al. Collaborative cache allocation and computation offloading in mobile edge computing [C]. 19th Asia-Pacific Network Operations and Management Symposium, Seoul, 2017: 366-369.
- [9] Xu J., Chen L., Ren S., Online learning for offloading and autoscaling in energy harvesting mobile edge computing [J]. *IEEE Transactions on Cognitive Communications & Networking*, 2017, 3 (3): 361-373.
- [10] Mehrabi M., Shen S. W., Latzko V., et al. Energy-aware cooperative offloading frame-work for inter-dependent and delay-sensitive tasks [C]. IEEE Global Communications Conference, Taipei, 2020: 1-6.
- [11] Guo X. Y., Singh R., Zhao T. C., et al. An index-based task assignment policy for achieving optimal power-delay trade off in edge cloud systems [C]. IEEE International Conference on Communications, Kuala Lumpur, 2016: 1-7.
- [12] Guo J. F., Song Z. Z., Cui Y., et al. Energy-efficient resource allocation for multi-user mobile edge computing [C]. IEEE Global Communications Conference, Singapore, 2017: 1-7.
- [13] Zhang H., Guo J., Yang L., et al. Computation offloading considering fronthaul and backhaul in small-cell networks integrated with MEC [C]. IEEE Conference on Computer Communications Workshops, Atlanta, 2017: 115-120.
- [14] Xian Y. J., Song Q. Y., Guo C. R., et al. A master-slave MEC server Collaborative Uninstallation and Resource allocation Scheme [J]. *Telecommunications Technology*, 2022, 62 (4): 407-415. (in Chinese).
- [15] Huang L., Bi S. Z., Zhang Y., et al. Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks [J]. *IEEE Transactions on Mobile Computing*, 2020, 19 (11): 2581-2593.
- [16] Fister L., Fister L., Yang X. S., et al. A comprehensive review of firefly algorithms [J]. *Swarm and Evolutionary Computation*, 2013, 13: 34-46.
- [17] Zhao J. M. Effectiveness of normalization pre-processing of big data to the machine learning performance [J]. *Journal of the Korea Institute of Electronic Communication Sciences*, 2019, 14 (3): 547-552.
- [18] Deng T. Research on Mobile Edge Computing Unloading Method Based on Collaborative Technology [D]. Nanjing: Nanjing University of Posts and Telecommunications, 2021.
- [19] Li B., Wang K. Y., Xue D., et al. K-Means based edge server deployment algorithm for edge computing environments [C]. IEEE Smart World, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation, 2018: 1169-1174.
- [20] Zehlike M., Bonchi F., Castillo C., et al. Fair: A fair top-k ranking algorithm [C]. Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. 2017: 1569-1578.
- [21] Li Y. Z., Wang S. G. An energy-aware edge server placement algorithm in mobile edge computing [C]. 2018 IEEE International Conference on Edge Computing, 2018: 66-73.

- [22] Liu Y. Research on Improvement and Application of Particle Swarm Optimization Algorithm [D]. Xi'an: Xidian University, 2013. (in Chinese).
- [23] Liu Y. B., Liang C. Application of immune particle swarm optimization algorithm in optimal allocation of agricultural water resources [J]. Mathematics in Practice and Cognition, 2011, 41 (20): 163-171. (in Chinese).
- [24] Miao Y. Q., Xu Y., Zhang W. Z., et al. Task Unloading Strategy Based on Improved Particle Swarm Optimization Algorithm in Networking of Vehicles [J]. Research in Computer Applications, 21, 38 (7): 2050-2055. (in Chinese).