Journal of Network Intelligence ©2025 ISSN 2414-8105
Taiwan Ubiquitous Information Volume **10**, Number **4**, November **2025**

Mobile Edge Computing Resource Optimization Method Based on Improved Firefly Optimization Algorithm

Xi-Hong Wang

University of Sanya
Sanya, Hainan 572000, P. R. China
wanghongxi197@163.com

Jin-Mei Zhan*

School of Information Science and Technology
Qiongtai Normal University
Haikou, Hainan 571127, P. R. China
zhanjinmei2019@126.com

She-Lei Li

Faculty of Engineering, Building Environment and Information Technology
SEGi University
Petaling Jaya 47810, Malaysia
3779570025@qq.com

*Corresponding author: Jin-Mei Zhan

Received March 29, 2024, revised January 24, 2025, accepted June 19, 2025.

Abstract. *At present, the application of mobile terminals is becoming increasingly intelligent and diversified. The implementation of many applications requires very high computational performance as the foundation. In edge deployment, multi-user concurrency will cause high load on edge computing, resulting in network delay and congestion. Therefore, a mobile edge computing resource optimization method based on improved firefly optimization (IFFO) algorithm is proposed. The proposed method first models the optimization of resource allocation and task scheduling in mobile edge networks, including mobile cloud computing delay, edge computing delay, etc. If the directly connected edge server has the ability to process tasks, there is no subsequent transmission delay; if it is necessary to use this as intermediary to send tasks to other edge servers in the edge domain for processing, it is necessary to increase the transmission delay of the backend. At the same time, an improved algorithm is proposed based on the original firefly optimization algorithm, which adds elite individuals to alleviate the problem of poor local optimal solutions. Use an improved firefly optimization algorithm to optimize and solve the constructed model. The experimental results show that when the number of mobile users increases from 10 to 80, the average delay of users only reaches a maximum of 2.3 seconds, which is significantly lower than the comparison method. When the request time slot is 6, the total cost of the proposed method is reduced by at least 25% compared to other methods. The proposed method has certain reference value in reducing latency and controlling total costs.*

Keywords: Mobile edge computing; Improved firefly optimization; Cloud computing; Delay; Mobile terminals; Edge servers

1. **Introduction.**

With the rapid development of Internet of Things technology, more and more terminal devices such as drones, implantable medical devices, and vehicle radars are being deployed on a large scale in various life scenarios, continuously collecting environmental information such as images and videos to support real-time data processing tasks such as power inspection [1], intelligent medical [2], and unmanned driving [3]. In this process, edge computing is gradually emerging which is a new ecosystem [4, 5], aiming at integrating telecommunications and IT services, and providing cloud computing platforms at the edge of the network. edge computing provides storage and computing resources at the edge to reduce the delay of end users, and can more effectively adopt the mobile network background and core network. Edge computing provides cloud computing and storage resources at the edge of the mobile network, with significant advantages such as ultra-low latency, intensive computing capabilities. It is quite necessary for reducing network congestion video stream analysis, augmented reality, and emerging applications.

At present, the application of terminals is becoming increasingly intelligent and diversified, such as artificial intelligence, augmented reality (AR), virtual reality (VR), autonomous driving and other applications gradually becoming a reality. The implementation of many applications requires high computational performance as support, such as VR/AR, cognitive assistance, etc. [5, 6]. If the user's device cannot provide sufficient computing power (such as due to outdated processors, limited capacity batteries, etc.), the application cannot run. In terms of portability, people usually hope to implement these applications on wearable devices, which means that the devices must lean towards miniaturization. At present, high performance and portability are almost contradictory, as smaller device sizes determine that devices can only be equipped with smaller processing chips and smaller batteries. The emergence of the computational offloading mode is to solve this contradiction. Computational offloading refers to the transfer of resource intensive computing tasks from resource constrained devices to external platforms.

On the basis of the edge cloud framework, there are also many research work on the deployment of edge computing application tasks for specific application scenarios. As literature [7] studied the deployment of social virtual reality (VR) applications in edge infrastructure, while meeting the economic operation of the system and the service quality of users. The article proposes an iterative algorithm called "ITEM" to determine the deployment location of multiple service entities, achieving efficient work in real scenarios. Literature [8] analyzed and studied the deployment of medical networks in edge computing, and comprehensively considered the deployment of tasks in local, edge nodes and data centers, which effectively reduced the system delay. Literature [9] proposed an input data stream scheduling method based on token bucket model for concurrent stream data systems, and shared edge computing resources to reduce application delay. Literature [10, 11] implements low latency deployment through reasonable modeling of applications in the edge computing environment, and literature [11] also proposes a programmable model for optimization. In [12, 13], the author proposed a collaborative caching framework, which can store data in advance by effectively using the network resources in edge computing, thus effectively improving the utilization of the network. Reference [14] analyzed task unloading and migration in mobile edge computing scenarios. Reference [15] also considered task scheduling based on edge data centers, but did not fully consider the task characteristics of multiple data streams, so its applicability is relatively limited.

Although the above studies consider the edge nodes for task deployment, most of them have not considered the task dependency in the edge computing environment. At the same time, in edge computing, because most tasks have the characteristics of low computation and small storage, they can also be used to improve efficiency. References [16, 17] enable tasks to be executed separately on smart devices and data centers through dynamic migration, and optimize the latency of streaming data processing through algorithms. But these methods mainly consider the two-level structure of data centers and intelligent devices, without considering the role of edge nodes. Although good results can be achieved in some scenarios, communication between smart devices and data centers often requires a backbone network, resulting in very high latency. If the edge nodes located between the two can be effectively utilized, it will provide more diverse and flexible solutions for improving application efficiency.

The main work of this article is summarized as follows: 1) Modeling the resource allocation and task scheduling optimization problem of MEC; 2) Improve the traditional firefly optimization algorithm by incorporating elite individuals to solve the problem of poor local optimal solutions. 3) Using an improved firefly optimization algorithm to solve the optimization problem of MEC.

2. **Model construction.**

The following text mainly models resource allocation and task scheduling problems. The schematic diagram of the model is shown in Figure 1. The scenario is in a mobile edge network, and wireless transmission delay is not the focus of this article. It can be regarded as a relatively fixed constant t_w. After the task arrives at the base station, if the directly connected edge server has the ability to process the task, there is no subsequent transmission delay; If it is necessary to use this as an intermediary to send tasks to other edge servers in the edge domain for processing, it is necessary to increase the transmission delay of the backend. Assuming there are m types of services and n edge servers at the edge, the symbol $i \in m$ represents a certain service, and $j \in n$ represents a certain edge server.

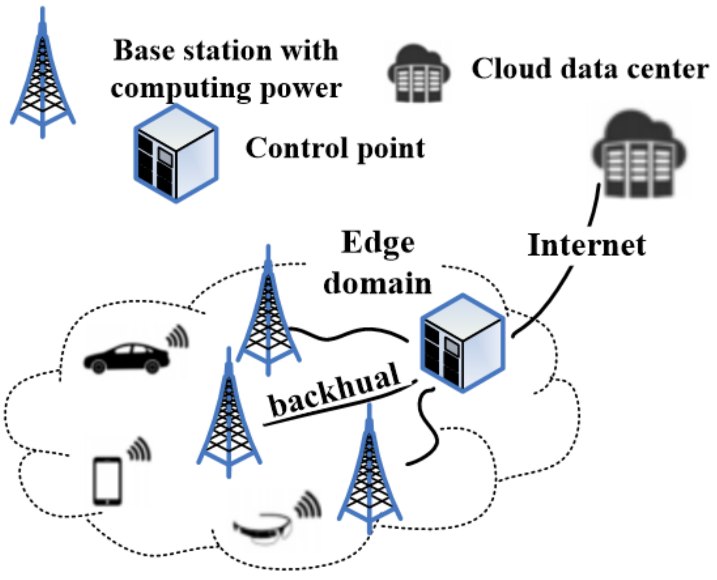

Figure 1. The diagram of model structure

2.1. **Edge cash.**

Firstly, consider the issue of edge caching [18]. In this article, it is believed that the successful operation of edge services requires resources to exceed a certain threshold Thr, below which services will not be able to run. And the resources at the edge are relatively limited. There is a decision-making problem here, which types of services are cached to minimize the average latency. Introduce decision variables $x_{i,j}$ to indicate whether service i is running on service j.

2.2. **Edge calculation delay.** Edge calculation delay refers to the time required for a task to be calculated in an edge server. The computation delay of a task is influenced by multiple dimensions of computing resources, and in this article, only the impact of CPU frequency on computation delay is considered. Assuming that the execution time of a task follows a distribution with parameter $1/\mu$, and μ denotes the service speed. Assuming that when the CPU frequency allocated to a service exceeds the aforementioned resource threshold, the service rate will increase linearly with the increase of CPU frequency. For example, if the coefficient of service i is k_i and the processor frequency allocated to edge server j is $f_{i,j}$, then the rate of service is $u_{i,j} = k_i \times f_{i,j}$. A simple M/M/1 queuing model can be used to model the computational delay of tasks.

2.3. **Cloud computing delay.** Due to the limitations of edge resources and the presence of sudden traffic, the edge is unable to handle user offloading tasks in a timely manner. In this case, the task may be sent to a remote data center for execution, and the latency of this process is referred to as cloud computing latency in this article. The communication delay experienced by cloud computing is much greater than the communication delay t_l within the edge network. It is worth noting that the latency of cloud computing varies over time, but in a relatively short period of time, it can be assumed that the communication latency from the edge to the cloud data center is a constant. Because cloud data centers have almost infinite computing resources and require less computing latency compared to edges, and are relatively fixed, overall, for each service i, a constant can be used to represent cloud computing latency within a cycle, represented by the symbol T_i.

2.4. **Modeling optimization.** Based on the above two kinds of delay, this article models the calculation of offloading latency. The purpose is to optimize the latency performance of the entire edge network by adjusting the resource allocation and task scheduling of the edge network. The latency of each user for each service is a random variable, and a weighted average (weight: θ) To characterize the performance of the entire edge.

As shown in Figure 2, when a certain edge server has more direct users using a certain service, the resource quantity of that service can be appropriately increased, that is, to increase $f_{i,j}$. When multiple services on the edge server have a high number of user requests, it is necessary to decide which services and what proportion of requests will be migrated, and where the migrated tasks will be executed. The arrival rate adjustment amount $d_{i,j}$, where j represents the load adjustment of service i on edge server j. When $d_{i,j}$ are positive numbers, it indicates the need to migrate the load of the same service from other base stations in the domain to this server. On the contrary, it means that a certain amount of user requests needs to be transferred to other nodes for computation, which may be other edge servers or cloud data centers.

After determining $d_{i,j}$, each node forwards the received user task requests to the controller in proportion $\rho_{i,j} = d_{i,j}/\lambda_{i,j}$, and the controller then determines the destination node for forwarding.

According to the M/M/1 model in queuing theory, the calculation delay of service i on edge server j can be obtained as:

$$t_{i,j}^c = \frac{x_{i,j}}{k_i \cdot f_{i,j} - \left(\lambda_{i,j} + d_{i,j}\right)}. \tag{1}$$

Even for the same service, each edge server has its own queuing queue, with different t^c, and needs to be weighted average based on the number of users it processes [19]. The proportion of load occupied by each edge server is:

$$\rho_{i,j}^c = \frac{\lambda_{i,j} + d_{i,j}}{\lambda_i^{\text{total}}}. \tag{2}$$

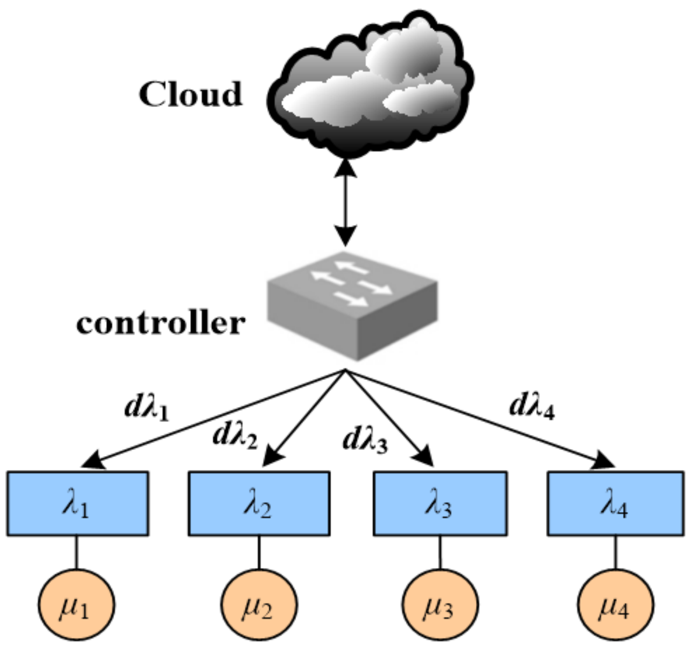

Figure 2. The diagram of model decision

The average computing delay of service i at the edge is:

$$\bar{t}_i^c = \sum_{j \in n} t_{i,j}^c \cdot \rho_{i,j}^c. \tag{3}$$

In addition to tasks executed at the edge, a portion of tasks are offloaded to the cloud for execution, and the proportion of tasks executed in the cloud is determined by the adjustment of the arrival rate $d_{i,j}$ for each edge server. The delay calculation introduced in backhaul transmission is also similar. Therefore, their transmission ratios can be expressed as follows [20]:

$$\rho_i^{\text{cloud}} = -\frac{\sum_{j \in n} d_{i,j}}{\lambda_i^{\text{total}}}, \tag{4}$$

$$\rho_i^l = -\frac{\sum_{j \in n} |d_{i,j}|}{\lambda_i^{\text{total}}}. \tag{5}$$

The average delay introduced is represented as:

$$t_i^{\text{cloud}} = \rho_i^{\text{cloud}} \cdot T_i \tag{6}$$

$$t_i^l = \rho_i^l \cdot t_i. \tag{7}$$

In summary, for a certain service i, its average delay can be expressed as:

$$\text{delay}_i = \sum_{j \in N} t_{i,j}^c \cdot \rho_{i,j}^c + \rho_i^{\text{cloud}} \cdot T_i + \rho_i^l \cdot t_i. \tag{8}$$

Constraints:
To ensure the convergence of the computation queue, the service rate must be greater than the request arrival rate, that is condition 1:

$$x_{i,j} \cdot k_i \cdot f_{i,j} > \lambda_{i,j} + d_{i,j}, \quad \forall i \in M,\ \forall j \in N. \tag{9}$$

At most, edge servers will forward directly connected tasks, and the adjusted service arrival rate cannot be less than 0, that is condition 2:

$$\lambda_{i,j} + d_{i,j} \geq 0, \quad \forall i \in M,\ \forall j \in N. \tag{10}$$

Only internally generated demands are processed within the edge domain, and the remaining demands are executed by the cloud. Therefore, the sum of service arrival rate adjustments should be less than or equal to 0, that is condition 3:

$$\sum_{j \in N} d_{i,j} \leq 0, \quad \forall i \in M. \tag{11}$$

On the same edge server, the total amount of resources allocated to different services cannot exceed the resource limit, that is condition 4:

$$\sum_{i \in M} x_{i,j} \cdot f_{i,j} \leq C, \quad \forall j \in N. \tag{12}$$

The final optimization problem is:

$$W = \min(\text{delay}_i) \quad \text{s.t. conditions 1 to 4.} \tag{13}$$

3. **Solve the optimization problem.**

3.1. **Firefly optimization algorithm.**

The Firefly Algorithm is a swarm intelligence algorithm that simulates the behavior of fireflies and is a naturally inspired algorithm. The Firefly Algorithm is based on the light attraction behavior of fireflies. The firefly individuals in the algorithm represent feasible solutions, while the brightness of fireflies represents the value of the objective function. Firefly individuals utilize their own brightness to attract other fireflies. Fireflies with lower brightness will move towards those with higher brightness, and the attraction between fireflies is inversely proportional to distance.

By using the firefly algorithm to optimize bandwidth resource allocation, the following advantages can be achieved.

1) *Global search capability:* Firefly algorithm has strong global search capability and can search for various possible solutions in bandwidth allocation space. This helps to find better bandwidth allocation solutions to improve bandwidth utilization efficiency and network performance.

2) *Adaptability and robustness:* The FFO algorithm has adaptability and robustness by simulating the behavior of fireflies. It can be adjusted and optimized based on the brightness value and distance between nodes to adapt to changes in network topology and bandwidth requirements.

3) *Parallelism:* The FFO algorithm can parallelly process the movement and interaction behavior of multiple nodes, thereby accelerating the optimization process. This makes it highly scalable for bandwidth resource allocation in large-scale networks.

In summary, optimizing bandwidth resource allocation using firefly algorithm can achieve advantages such as global search, adaptability, and parallelism, thereby improving bandwidth utilization efficiency and network performance. The brightness of individual fireflies at position r can be expressed as [21]:

$$I(r) = I_0\, e^{-\gamma r_{ij}^2}, \tag{14}$$

where I_0 represents the initial brightness of the firefly itself, γ denotes light absorption rate and r_{ij} represents Euclidean distance.

The attraction between two fireflies of different brightness is:

$$\beta_{ij} = \beta_0\, e^{-\lambda r^2}. \tag{15}$$

In the formula, β_0 represents the attraction between two fireflies with a distance of 0. Fireflies will move towards fireflies with higher brightness than themselves, so a formula for updating firefly position is constructed [21]:

$$u_i = u_i + \beta_{ij}(u_j - u_i) + \alpha(\text{rand} - 0.5), \tag{16}$$

where u_i, u_j represent the positions of individual fireflies, $\alpha(\text{rand} - 0.5)$ represents the disturbance term, and α represents a random parameter.

3.2. **Levy flight and its improved scheme.** Levy flight is a walking method that simulates the behavior of random phenomena in nature and has been widely used in improvement strategies for optimization algorithm research. For example, algorithm designs combined with PSO algorithm [22], grey wolf optimization algorithm [23], atomic search algorithm [24] have all improved algorithm performance to a certain extent. The characteristic of Levy flight is that its step size follows a Levy stable distribution and is a non-Gaussian stochastic process. Previous literature studies have shown that the search path of Levi's flight can enhance the algorithm's search ability in the global search space, expand the search range, and make it easier for the algorithm to jump out of local extremum points. However, the introduction of Levy flight will not only improve the algorithm's global search ability, but also weaken its local optimization ability to a certain extent in the later stage of algorithm iteration, that is, the algorithm may still be in a global search state when more small domain searches are needed, leading to a decrease in convergence accuracy.

To solve the problem of poor local optimal solution caused by the introduced Levy flight mechanism, a crossover operator with elite participation is introduced. For heuristic optimization algorithms, the elite individuals in the population play an important guiding role and have the potential to find the optimal solution. The more the population tends towards the elite individuals generated during iteration, the easier it is to approach the optimal solution. Considering that the advantage of the solution for elite individuals lies in the better performance of partial gene expression positions compared to non-elite individuals, using elite individuals to participate in crossover operations to scatter their advantageous gene fragments will significantly improve convergence speed compared to traditional methods that only retain elite individuals and directly apply them to crossover operators during the iteration process. By increasing the proportion of genes containing elite individuals in the population, it is more likely to obtain a population that maximizes the quality of solutions. In view of this, this article proposes a crossover operator with elite participation, which crosses non elite individuals in the population with the existing best elite individuals, retains the better individuals in the obtained offspring and parents, and replaces the fireflies of the original non elite individuals. This operator is easy to implement and has significant effects. It can quickly update the position of individual fireflies and easily obtain relatively better solutions. The problem being solved is a minimum value problem, and the algorithm is described as follows:

Step 1: Generate random intersections $r \in \{1, 2, \ldots, D\}$.

Step 2: Extract the genetic fragments of elite individual t_{best} located after the intersection point, i.e., $[b_r, b_{r+1}, \ldots, b_D]$; similarly, extract genetic fragments from non-elite individual t located after the intersection point, that is $[x_r, x_{r+1}, \ldots, x_D]$.

Step 3: Swap the extracted fragments to generate two new offspring individuals c_1 and c_2, that is $[b_1, b_2, \ldots, b_{r-1}, x_r, x_{r+1}, \ldots, x_D]$ and $[x_1, x_2, \ldots, x_{r-1}, b_r, b_{r+1}, \ldots, b_D]$.

Step 4: Solving $f(c_1)$ and $f(c_2)$ by substituting the genotype of offspring individuals into the objective function.

Step 5: Comparing the function values of offspring and parents, select the optimal individual to enter the population and replace the original non elite individual:

$$X_n' = \begin{cases} c_1, & \text{if } f(c_1) < \min\left(f(c_2),\, f(X_n^t)\right), \\ c_2, & \text{if } f(c_2) < \min\left(f(c_1),\, f(X_n^t)\right), \\ X_n^t, & \text{otherwise.} \end{cases} \tag{17}$$

3.3. **Optimization solution.** The process of finding the optimal solution of edge computing delay is shown in Figure 3. The search space of the optimized firefly algorithm is multidimensional. After randomly initializing the population, randomly generated firefly individuals are added to the population within each iteration cycle.

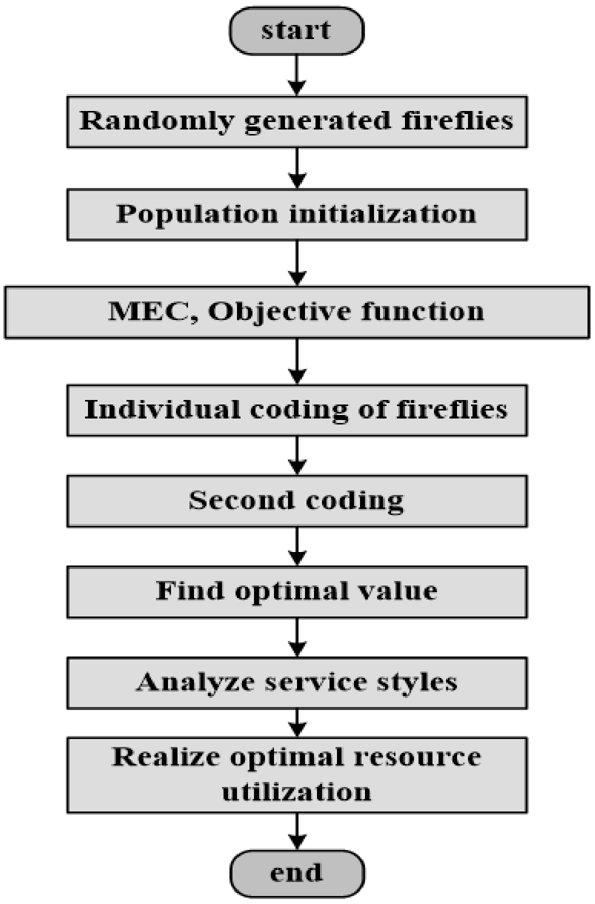
start
Randomly generated fireflies
Population initialization
MEC, Objective function
Individual coding of fireflies
Second coding
Find optimal value
Analyze service styles
Realize optimal resource utilization
end

Figure 3. Solution procedure for the optimization problem of mobile edge computing

Code the firefly individuals according to the distribution of users in the mobile edge computing network, and then select 2 different firefly individuals randomly from the firefly individuals for secondary coding. In each iteration process, find the optimal value through two steps: Firstly, based on the given objective function. Second, analyze the service types requested by users in mobile edge computing networks, so as to select appropriate service types to achieve optimal resource utilization. Set the maximization of benefits of mobile edge computing system as the core of network bandwidth resource optimization strategy, and determine the total revenue of the system after determining the service type of users. For a given mobile edge computing network, the delay with the largest user benefit is the smallest, and the optimization is the best.

In mobile terminal services, operators can target different types of services and require different bandwidth resources. For example, for live streaming services, due to the need for users to watch live content within a specific time frame, the service requires high bandwidth; For gaming services, as users need to play games within a specific time frame, this service has lower bandwidth requirements and higher latency requirements, and must maintain high real-time performance. By analyzing the different business types mentioned above and defining the relationship between business types and user benefits, we can better implement resource optimization strategies.

4. Experiments and Analysis.

4.1. Experimental environment and parameters settings.

This article simulates the experiment running on a server with an i7 processor and 16 GB of memory. The delay between edge computing nodes follows a uniform distribution of 0.1∼0.4 s [25]. The bandwidth capacity of each edge computing node to transmit data to the WAN is the same, which is 50 MB/s. In order to verify the effect of the algorithm under different edge computing node sizes, 50∼500 edge nodes are selected in the experiment. To simulate the effect of various types of streaming data processing applications competing for bandwidth resources, this article sets the bandwidth required for the data stream to follow a uniform distribution of 1–10 MB/s. Table 1 shows the main parameter settings of the algorithm in this article.

Table 1. The main parameter settings of IFFO in this paper

Parameters	Settings
The number of firefly groups	10∼20
The number of fireflies	[50, 100]
Attenuation coefficient	[0.5, 0.9]
Maximum attraction	[0.1, 1.0]
Step	[0.9, 1.5]
Light intensity attenuation coefficient	[0.05, 0.1]
Mars distance	[0.01, 0.1]
Maximum number of iterations	[500, 1000]

1) *Simulation software:* edge computing task unloading optimization simulation software includes NS-3 (network simulator 3), OMNeT++, MATLAB/Simulink, etc. These kinds of software provide a wealth of network models, communication protocols, and performance evaluation tools, which can simulate and analyze the edge computing task unloading scheme.
2) *Simulation hardware:* In the research of edge computing task unloading optimization, the simulation environment based on container virtualization technology, Docker and Kubernetes, can be used to build virtual edge computing nodes and devices to simulate the real edge computing scene. In addition, you can also use simulators or laboratory equipment to conduct simulation experiments, such as using Mininet to build a virtual network environment, and using OpenStack to build edge computing environment.

According to the application of mobile edge computing network at this stage, MATLAB software is used to simulate the mobile edge network to maintain its stability.

4.2. Comparative methods.

In order to analyze the effectiveness of the proposed method, a comparative analysis will be conducted on the following schemes.
(1) TCO (Task caching and offloading) [26]. This scheme first provides an offloading strategy, and then uses block coordinated descent method to provide a caching strategy.

(2) TPO (Task popular caching and offloading) [27]. This scheme is based on the number of task requests for caching, only caching tasks with more requests, without considering factors such as task freshness and data size.
(3) TCA (Task offloading and cache assist) [28]. This scheme designs a caching strategy to assist task offloading, which is based on the Zipf distribution probability model and is used to calculate the request frequency of tasks. Calculate the popularity of each task based on its request frequency and the proportion of task computation time, and update the cache in order of popularity until the cache space is filled.

At the same time, several commonly used heuristic algorithms were compared, such as the hybrid wolf pack optimization algorithm, the original firefly optimization algorithm, and the ant colony optimization algorithm.

4.3. **Analysis of delay.** Through the above experiments, it can be seen that resource optimization strategies have good effectiveness in practical applications. Considering that the computational performance of resource optimization strategies involves a lot of content, it is necessary to measure the practical application ability of resource optimization strategies from the perspective of computational delay. Before the experiment, each user was set to have four tasks to be calculated. Each task has relevance, and different network bandwidth resource optimization strategies were implemented. At the same time, two network edge computing capabilities were set, namely, 4GHz and 8GHz. The average computing delay of each user was counted using third-party software.

The experimental results are shown in Figure 4. Figure 4 shows the average latency of various resource optimization strategies under different computing conditions. The results of Figure 4 (a) show that as the number of users increases, the average latency of users continues to increase, reaching a maximum of 2.3 seconds. At 8GHz computing power, although the latency is reduced, it is still relatively high; The experimental results shown in Figures 4 (b) and 4 (c) are similar to those in Figure 4 (a), with higher average delay under both calculation conditions. In contrast, the experimental results shown in Figure 4 (d) have a relatively low average delay. Under both calculation conditions, the average delay is within 1 second, indicating that the improved firefly swarm method makes the calculation of network bandwidth resource optimization strategies faster and more stable. From the comprehensive results, it can be seen that the proposed mobile edge computing network bandwidth resource optimization strategy based on the improved firefly algorithm has low energy consumption and low latency, and its overall computing performance is better than other optimization strategies.

This strategy can dynamically adjust the allocation of network bandwidth resources based on network conditions and user needs, improve data transmission efficiency, and reduce energy consumption. Secondly, this strategy can avoid network congestion and resource waste, improve network utilization and performance, thereby reducing network latency and energy consumption. Finally, this strategy can fully utilize the computing and storage resources of edge devices, reduce the burden of cloud data processing and storage, and further reduce energy consumption.

Figure 5 shows the impact of the total number of task offloads on the average network latency. When the number of tasks allowed to be offloaded is small, the curve stabilizes. This is because there are fewer available free access points, and under a certain energy consumption, local devices cannot offload tasks to the CPU through a link with better channel quality. The changes in several algorithms in the early stage were relatively small, and the offloading effect between the access point and CPU was not significant. At this time, the delay mainly depends on the task matching relationship. As the total number of tasks increases to more than eight, the average network latency gradually decreases with

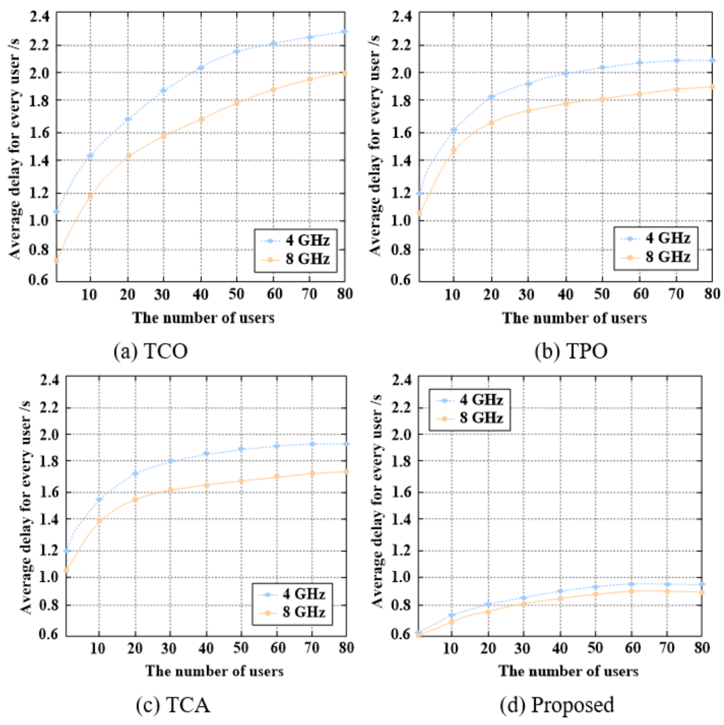

Figure 4. Average delay of different methods under the same calculation conditions

the number of allowed offloads. The proposed improved algorithm has a faster rate of latency reduction and performs better when the number of tasks is large, which is superior to other methods (ant colony optimization (ACO) algorithm, firefly optimization (FFO) algorithm and hybrid wolf optimization (HWO) algorithm).

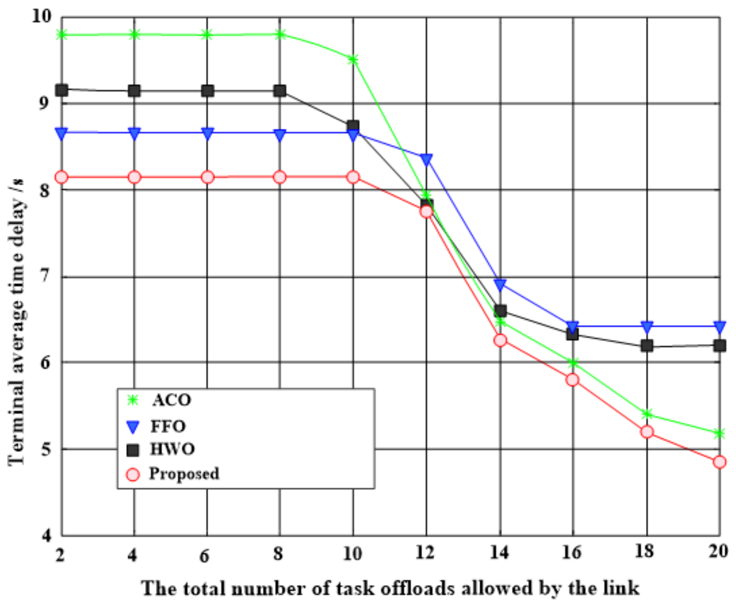

Figure 5. The relationship between different number of tasks and average network delay

4.4. **Analysis of user cost.** Figure 6 compares the relationship between total cost and number of users under various schemes. The experimental results indicate that when the number of users is 5, the total cost of the four schemes is similar. When the number of users is 30, the total cost of this solution is reduced by 41.6%, 61%, and 68% compared to other solutions, respectively. Therefore, this scheme consistently outperforms other schemes by achieving the lowest total cost, confirming the effectiveness of this scheme in reducing costs, especially when the number of users increases.

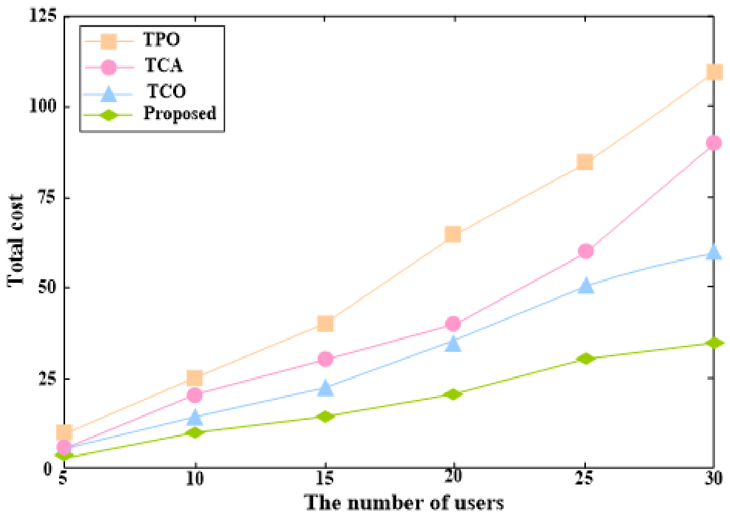

Figure 6. The relationship between the total cost of different solutions and the number of users

Figure 7 shows the relationship between the total cost and request time slots for four strategies, with the number of users set to 20. The results indicate that the total cost of all four schemes decreases with increasing request time slots. The cache-oriented task offloading strategy proposed in this article consistently outperforms the other three solutions. This is because the proposed solution in this article considers multiple factors for caching and updating tasks, thereby more effectively utilizing cache resources. The TCA scheme only considers task request frequency as the basis for task caching, while ignoring other key factors. And TCO randomly selects cached tasks, and when random cache replacement occurs, popular tasks may be replaced by less popular tasks, leading to significant fluctuations in total costs. When the request time slot is 6, the total cost of our proposed solution is reduced by 75.4%, 34.8%, and 26.9% compared to other solutions. Therefore, as the request time slots increase, the proposed solution can more effectively utilize cache resources and reduce system overhead costs.

5. **Conclusion and Discussion.** This article proposes a bandwidth resource optimization strategy based on the firefly algorithm, which allocates network bandwidth to users and determines the maximum delay time of users, calculates network energy consumption, and then uses the firefly algorithm to optimize bandwidth. This method can not only solve the resource allocation problem in multi-user systems, but also effectively reduce network latency and improve user satisfaction. In addition, by comparing with other optimization strategies, it can be found that the algorithm proposed in this paper has advantages in convergence speed and computational performance. The research work of this paper has certain guiding significance for the performance analysis and optimization of mobile edge computing systems.

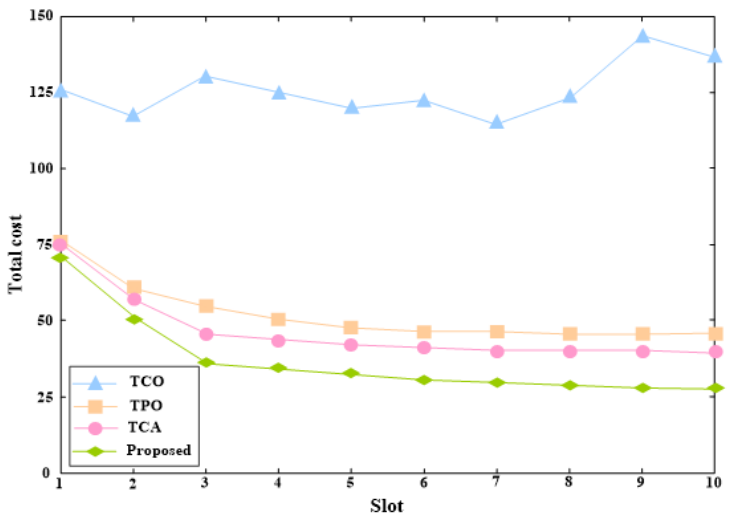

Figure 7. The relationship between the total cost of different strategies and request time slots

With the continuous change of the network environment, the strategies proposed in this paper still have room for progress. The following two points can be considered for future research work: 1) In future research, we will consider more factors, such as Internet of Things services based on mobile edge computing, telemedicine based on edge computing, intelligent transportation, industrial Internet of Things and other application scenarios. At the same time, we can also study the edge cloud computing service model and quality of service assurance mechanism based on mobile edge computing system. 2) How to analyze the impact of devices on user response time in edge networks during the optimization process; How to improve the computational performance of firefly algorithm in multi-user systems by improving it; And how to improve system performance to adapt to the dynamic changes in user demand for bandwidth resources.

REFERENCES

[1] J. Colistra, "The evolving architecture of smart cities," in *Proceedings of the 2018 4th IEEE International Smart Cities Conference*. Piscataway, pp. 1–8, 2018.

[2] A. Solanas, C. Patsakis, M. Conti, S. V. Ioannis, R. Victoria, F. Francisco, P. Octavian, A. P. Pablo, N. P. Despina, and M. B. Anotni, "Smart health: A context-aware health paradigm within smart cities," *IEEE Communications Magazine*, vol. 52, pp. 74–81, 2014.

[3] J. Wang, D.-G. Yang, and X.-M. Lian, "Research on electrical electronic architecture for connected vehicle," in *Proceedings of the 2016 IET International Conference on Intelligent and Connected Vehicles*. London, IET, pp. 1–6, 2016.

[4] H.-Z. Lv, D. Chen, B. Fan, Y.-X. Wu, and Y.-X. Wu, "Standardization progress and case analysis of edge computing," *Journal of Computer Research and Development*, vol. 55, pp. 487–511, 2018.

[5] T. Vo, P. Dave, G. Bajpai, and R. Kashef, "Edge, Fog, and Cloud Computing: An Overview on Challenges and Applications," *arXiv*, 01863, 2022.

[6] R. Meneguette, R. D. Grande, J. Ueyama, GPR Filho, and E. Madeira, "Vehicular Edge Computing: Architecture, Resource Management, Security, and Challenges," *ACM Computing Surveys*, pp. 244550843, 2023.

[7] L. Wang, L. Jiao, T. He, J. Li, and M. Muhlhauser, "Service entity placement for social virtual reality applications in edge computing," in *Proceedings of the 2018 IEEE Conference on Computer Communications*. Piscataway: IEEE, pp. 468–476, 2018.

[8] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, "Cost efficient resource management in fog computing supported medical cyber-physical system," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, pp. 108–119, 2017.

[9] R. Tolosana-Calasanz, J. Á. Banares, C. Pham, and O. Rana, "End-to-end QoS on shared clouds for highly dynamic, large-scale sensing data streams," in *Proceedings of the International Symposium on Cluster, Cloud and Grid Computing*. Piscataway: IEEE, pp. 904–911, 2012.

[10] N. Govindarajan, Y. Simmhan, N. Jamadagni, and P. Misra, "Event processing across edge and the cloud for internet of things applications," in *Proceedings of the International Conference on Management of Data*. New York: ACM, pp. 101–104, 2014.

[11] A. Brogi and S. Forti, "QoS-aware deployment of IoT applications through the fog," *IEEE Internet of Things Journal*, vol. 4, pp. 1185–1192, 2017.

[12] H. Tan, Z. Han, X.-Y. Li, et al., "Online job dispatching and scheduling in edge-clouds," in *Proceedings of the Conference on Computer Communications*. Piscataway: IEEE, pp. 1–9, 2017.

[13] S. Guo, B. Xiao, Y. Yang, et al., "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *Proceedings of the Annual IEEE International Conference on Computer Communications*. Piscataway: IEEE, pp. 1–9, 2016.

[14] W. Liu, Y.-C. Huang, W. Du, and W. Wang, "Resource-constrained serial task offload strategy in mobile edge computing," *Journal of Software*, vol. 31, pp. 1889–1908, 2020.

[15] X. Ju, S. Su, C. Xu, et al., "Computation offloading and tasks scheduling for the internet of vehicles in edge computing: A deep reinforcement learning-based pointer network approach," *Computer Networks*, pp. 109572, 2023.

[16] K.-J. Wang, C.-J. Lin, D. T. B., et al., "Bilayer stochastic optimization model for smart energy conservation systems," *Energy*, pp. 247, 2022.

[17] Z. Ali, Z. H. Abbas, G. Abbas, et al., "Smart computational offloading for mobile edge computing in next-generation Internet of Things networks," *Computer Networks*, vol. 24, pp. 198, 2021.

[18] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE International Conference on Computer Communications*. IEEE, pp. 109–118, 2018.

[19] S. R. Sabuj, D. K. P. Asiedu, K. J. Lee, et al., "Delay optimization in mobile edge computing: cognitive UAV-assisted eMBB and mMTC services," *IEEE Transactions on Cognitive Communications and Networking*, vol. 2, pp. 8, 2022.

[20] N. Yu, Q. Xie, Q. Wang, et al., "Collaborative Service Placement for Mobile Edge Computing Applications," in *IEEE Global Communications Conference*. IEEE, pp. 102–111, 2019.

[21] X.-S. Yang, "Firefly algorithm, Lévy flights and global optimization," *Research and Development in Intelligent Systems XXVI: Incorporating Applications and Innovations in Intelligent Systems XVII*. London: Springer, pp. 209–218, 2010.

[22] B.-L. Yan, Z. Zhao, Y.-C. Zhou, et al., "A particle swarm optimization algorithm with random learning mechanism and Levy flight for optimization of atomic clusters," *Computer Physics Communications*, vol. 219, pp. 79–86, 2017.

[23] Y. Pathak, K.-V. Arya, and S. Tiwari, "Feature selection for image steganalysis using Levy flight-based grey wolf optimization," *Multimedia Tools and Applications*, vol. 78, pp. 1473–1494, 2019.

[24] S. Barshandeh and M. Haghzadeh, "A new hybrid chaotic atom search optimization based on tree-seed algorithm and Levy flight for solving optimization problems," *Engineering with Computers*, vol. 37, pp. 3079–3122, 2021.

[25] Y. Pang, J.-G. Wu, L. C. Long, and M.-Y. Yao, "Energy balancing for multiple devices with multiple tasks in mobile edge computing," *Journal of Frontiers of Computer Science and Technology*, vol. 16, pp. 480–488, 2022.

[26] Y.-X. Hao, M. Chen, L. Hu, M.-S. Hossain, and A. Ghoneim, "Energy efficient task caching and offloading for mobile edge computing," *IEEE Access*, vol. 6, pp. 11365–11373, 2018.

[27] X.-L. Yang, Z.-S. Fei, J.-C. Zheng, N. Zhang, and A. Anpalagan, "Joint multi-user computation offloading and data caching for hybrid mobile cloud/edge computing," *IEEE Transactions on Vehicular Technology*, vol. 68, pp. 11018–11030, 2019.

[28] C.-C. Zhao, X.-M. Guo, and X.-W. Hai, "Joint optimization strategy of task offloading and resource allocation for cache-assisted mobile edge computing," *Science Technology & Engineering*, vol. 23, pp. 3812–3819, 2023.