American Journal of Engineering Research (AJER)
e-ISSN: 2320-0847 p-ISSN : 2320-0936
Volume-13, Issue-8, pp-62-71
www.ajer.org
Research Paper Open Access

# Research on Task Scheduling and Resource Allocation Algorithms in a Single-User Multi-Edge Node Environment

Yuchao Zhu[1] Jiang Wang[2]
[1]( China Mining Products Safety Approval and Certification Center, Beijing, China)
[2]( Tianjin University of Science and Technology, Tianjin, China)
Corresponding Author: Yuchao Zhu

***ABSTRACT :*** *Traditional centralized computing architectures typically send data to centralized data centers or the cloud for processing and analysis. This process often encounters issues such as significant data transmission latency, network congestion, low reliability, and data security concerns. Consequently, traditional centralized cloud computing architectures struggle to meet the demands for low latency, high bandwidth, and low energy consumption in certain scenarios. Edge computing is a distributed computing method that does not rely on centralized cloud systems. By processing data closer to the user devices at the network edge, it significantly reduces the system overhead required to complete tasks such as data analysis, decision-making, and real-time responses. This reduces latency, enhances resource utilization, and enables networks to better handle applications with strict delay or energy consumption requirements. To address these challenges, this paper investigates a task offloading algorithm within a cloud-edge-terminal collaborative architecture tailored for single users and multiple edge nodes. A single user device generates a set of tasks with dependency constraints. The total system overhead serves as the optimization objective for an improved genetic algorithm (ACGA) to determine the offloading strategy. First, this research presents a detailed parameterized modeling of the total overhead system model for the edge scenario, using its reciprocal as a fitness function for the algorithm. Next, it enhances the genetic algorithm's local search capability through improvements to the crossover operator based on the pheromone evaporation operation of ant colony optimization. Finally, simulation experiments on this edge network environmental model validate that, in a single user and multiple edge node network environment, the ACGA algorithm demonstrates good convergence after multiple iterations and significantly reduces system computation task latency, thereby decreasing user wait times.*
***KEYWORDS:*** *edge computing,Genetic Algorithm,Uninstall task,Resource allocation.*

---

Date of Submission: 03-08-2024 Date of acceptance: 14-08-2024

---

## I. INTRODUCTION

By concentrating computing tasks in remote large-scale data centers, cloud computing has shown great capabilities in processing large-scale data processing, featuring dynamic scalability, high reliability, on-demand deployment and other characteristics [1]. However, in some specific scenarios, cloud computing has been widely used in large-scale data processing. Such as driverless cars [2], smart home appliances, Virtual Reality [3] and other fields, this centralized computing mode shows limitations in meeting real-time requirements and processing large amounts of edge-generated data [4].

In view of these limitations, in the late 1990s and early 21st century, researchers began to explore the possibility of data processing and computing at the edge of the network, and mobile edge computing came into being as a complementary and expanded technology [5]. Mobile edge computing does not simply replace cloud computing. Cloud computing centers require edge servers to do preliminary processing of massive original data, while edge computing centers require powerful computing capabilities and massive storage as basic support [6]. The core idea of mobile edge computing is to transfer data processing tasks from the cloud to the edge of the network. The application scenarios of mobile edge computing cover many fields such as smart transportation [7], smart city [8], and Industry 4.0. Among these applications, mobile edge computing can provide faster service response, more efficient data processing capabilities, and a more personalized user experience.

With the popularization and deepening of mobile edge computing applications, how to effectively schedule tasks and allocate resources in edge environments has become an urgent problem to be solved. The

characteristics of mobile edge computing environment include the heterogeneous computing resources of edge nodes, the limited network bandwidth and the huge number of devices, which bring challenges to the scheduling of computing tasks and the effective allocation of resources. In this context, computing task offloading strategy is particularly important. It is related to how to efficiently allocate and execute computing tasks among edge nodes and between edge nodes and the cloud, so as to realize the optimal utilization of resources and ensure the efficiency and reliability of services.

In this context, this paper primarily investigates algorithms for solving computational task offloading strategies in edge environments. It explores the challenges associated with task offloading in mobile edge computing and proposes a series of optimization methods. Through a detailed examination of the key technologies and challenges involved in the task offloading process, it leverages the features of mobile edge computing, applying advanced algorithms and techniques to optimize task offloading, promote efficient utilization of computational resources, reduce overall system energy consumption, and enhance service responsiveness and reliability. Focusing on a single-user scenario with multiple edge nodes, an improved genetic algorithm is proposed for the computational task offloading problem, aiming to minimize the total system cost. A parameterized model of the network in this scenario is established, and simulation experiments demonstrate that the improved genetic algorithm significantly reduces the total system cost.

## II. RELATED CONCEPTS

This paper addresses the issues of latency and energy consumption in edge environments. A cooperative offloading model is established, involving multiple edge nodes and a cloud node for a single user device. An improved genetic algorithm is proposed to determine the offloading strategy. Initially, the latency and energy consumption within the offloading environment are analytically derived, leading to the construction of a parameter model that incorporates system total cost (the weighted sum of latency and local energy consumption) under various constraints. Subsequently, the reciprocal of the optimized objective function is used as the fitness function in the improved genetic algorithm. A coding scheme is presented to select superior individuals based on the fitness function values. An enhanced crossover operator is then developed, relying on the pheromone evaporation mechanism from ant colony optimization. After multiple iterations, the offloading strategy with minimal system total cost is obtained. Finally, a comparison with traditional genetic algorithms is performed. Experimental data indicates that the proposed algorithm significantly reduces the system's total cost, demonstrates stronger local search capabilities, and rapidly achieves superior offloading strategies.

### 2.1 MOBILE EDGE COMPUTING

Mobile Edge Computing, as a highly promising computing paradigm, has been widely applied and recognized across various fields for its distinct features and advantages. The edge computing architecture is shown in Fig. 1.

1) Low Latency: Mobile Edge Computing deploys computational resources closer to the user at network edge nodes. This configuration enables shorter data transmission paths and faster data processing, thereby reducing latency. Such low latency is crucial for applications that require real-time responses.

2) High Reliability: By pushing computing tasks to the source of data generation, Mobile Edge Computing facilitates data processing and decision-making near the user or device, reducing reliance on centralized data centers. This distributed computing model enhances the robustness and reliability of the system, ensuring that edge nodes continue to operate effectively even in cases of unstable network connections or failures at centralized data centers.

3) Bandwidth Savings: Mobile Edge Computing processes and analyzes data at the point of generation, transmitting only the results to centralized data centers or the cloud. This significantly reduces the volume of data transmitted over the network, conserving bandwidth resources. This bandwidth-saving feature is particularly important for large-scale Internet of Things (IoT) applications, as it helps lower data transmission costs and mitigate network congestion risks.

4) Flexibility and Scalability: The Mobile Edge Computing framework can flexibly deploy and scale computational resources based on actual needs. Edge nodes can be deployed in various geographical locations and network environments to adapt to different application scenarios and requirements. This flexibility and scalability enable Mobile Edge Computing to respond to evolving business demands and technological challenges.

5) Privacy Protection: Mobile Edge Computing localizes data processing and decision-making, allowing data to be processed and analyzed on-site, which reduces the risk of raw data leakage or loss during transmission. This localized data processing feature aids in protecting user privacy and data security, in compliance with privacy protection regulations and standards.

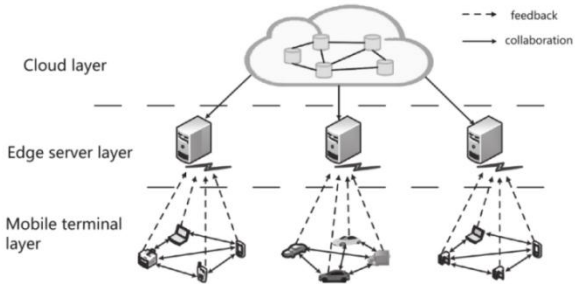



**Fig. 1 Edge computing architecture**

## 2.2 CALCULATING THE RELEVANT CONTENT OF OFFLOADING

Task offloading is a key concept in mobile edge computing, referring to the process of transferring computational tasks that would typically be executed by mobile devices (such as smartphones and tablets) to other devices that possess greater computational resources or are available to process these tasks. This practice alleviates the computational burden on mobile devices, enhances operational efficiency, extends device usage time, and ensures a better user experience when handling performance-intensive applications. Task offloading comprises three critical components: determining offloading, the volume of tasks to offload, and the specific tasks to be offloaded. The steps involved in computation offloading are illustrated in Fig. 2.

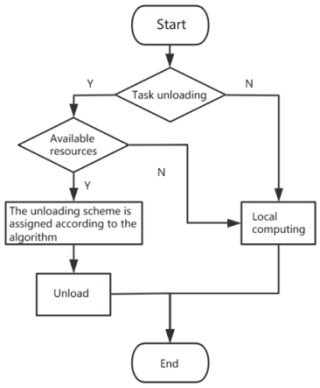



**Fig. 2-2 Task offloading steps**

During the task offloading process, there are several key optimization metrics that need to be considered to ensure optimal offloading results:

1) Latency: Latency refers to the time taken from task submission to task completion. Reducing processing latency during task offloading is crucial, particularly in scenarios like real-time video streaming or online gaming, where decreased latency can significantly enhance user satisfaction.

2) Energy Consumption: Energy consumption indicates the amount of energy expended during the task offloading process. Mobile devices often have limited battery capacity, making it essential to consider how to reduce energy consumption during offloading to extend battery life. Effective offloading decisions and optimized strategies can help minimize energy use, thereby increasing device uptime.

3) Bandwidth Consumption: Bandwidth consumption pertains to the network bandwidth used during task offloading. Since mobile devices typically connect to the internet or edge environments, it is important to minimize network bandwidth usage during offloading to avoid excessive resource utilization, ensuring a satisfactory experience for other users.

4) Resource Utilization: Resource utilization measures the efficiency of computing resources, storage resources, and others involved during the task offloading process. When selecting offloading targets, the resource utilization of the target server or node must be considered to ensure that tasks are fully utilized without wasting resources.

## 2.3 GENETIC ALGORITHM

Genetic Algorithm (GA) is a search algorithm that solves optimization problems by utilizing principles of evolution found in nature. In the 1960s, American professor Jhon Holland first proposed a genetic algorithm based on natural phenomena such as biological reproduction, hybridization, mutation, and selection. The specific genetic terminology used in genetic algorithms is illustrated in Table 1.

**Table 1 Specific terms in the genetic algorithm**

| a specific term for genetic algorithms | implication |
|---|---|
| chromosome (individual) | a feasible solution |
| gene | chromosomal element |
| population | the set of feasible solutions |
| crossover probability | the probability of individuals crossing |
| variation probability | the probability of individual variation |
| fitness | assess the value of the individual |

When using genetic algorithms to solve specific problems, it is necessary to map feasible solutions to chromosomes, with each solution corresponding to one chromosome[16]. The population consists of a collection of all chromosomes. The encoding of chromosomes can be determined based on specific circumstances, such as binary encoding or real-valued encoding. The genetic algorithm first randomly generates a population and then evaluates the fitness of individuals within this population using a fitness function. Ultimately, the next generation is selected based on these fitness values. Individuals with higher fitness values have a higher chance of being retained, while those with lower fitness values also have opportunities for retention. Following this, individuals are chosen for crossover and mutation. After several iterations, the overall fitness value is expected to improve until the algorithm converges.

## III. ALGORITHM FOR TASK SCHEDULING AND RESOURCE ALLOCATION IN A SINGLE-USER MULTI-EDGE NODE ENVIRONMENT

### 3.1 MODELING PROCESS

Assume the set of servers is $S=\{s_0,s_1,s_2,...,s_m,s_{m+1}\}$, where $s_0$ represents the local endpoint, $s_1$ denotes the edge node attached to the local endpoint, $s_{m+1}$ indicates the cloud endpoint, and $s_2,...,s_m$ represent other edge nodes. The set of computing capabilities for the servers is $F=\{f_0, f_1, f_2,..., f_m,f_{m+1}\}$ with subscripts corresponding one-to-one to the server set.

Local user devices Ugenerate computing tasks subject to constraints, which are represented as a directed acyclic graph $G=(V,E)$ [17], where $V=\{v_1,v_2,...,v_m\}$ indicates the task set, with each task $v_i$ having a pair $\{d_i,c_i\}$ ,where $d_i$ represents the size of the task $v_i$ in kilobytes (kb), and $c_i$ represents the number of CPU cycles required for task $v_i$. The set Erepresents the dependencies among the tasks; a directed edge $(v_j,v_i)\in E$ indicates that task $v_i$ can only commence after completing task $v_j$. Task $v_j$ is termed the predecessor of task $v_i$, denoted as $\mathrm{pre}(v_i)$, while task $v_i$ is called the successor of task $v_j$, denoted as $\mathrm{suc}(v_j)$. When task $v_i$ has no predecessors, it is an entry node;conversely, if task $v_j$ has no successors, it is an exit node. The dependency relationship model is illustrated in Fig. 3.

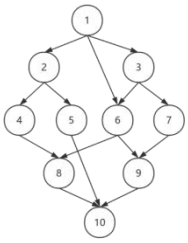



**Fig. 3 Task set with task dependencies**

Introducing decision variable $x_{i,j}$, if $x_{i,j}$ =1, it indicates that task $v_i$ is allocated to server $s_j$ for execution. However, it is important to note that if a task is offloaded to the cloud, it must first be offloaded to the edge before being transferred to the cloud. For instance, $x_{1,3}$ =1means that task $v_1$ is assigned to server $s_3$ for execution, requiring it to first be transmitted to $s_1$, and then from $s_1$ to $s_3$.

1）Latency Optimization Objectives

a. The execution time of task $v_i$ on server $s_j$ is given by:

$$t_{i,j}^{C} = \frac{c_i}{f_j}, \ v_i \in V, s_j \in S$$

Equation (1)

Where $c_i$ denotes the computational size of the task, and $f_j$ represents the computational capacity of server $s_j$.

b. The transmission time for offloading task $v_i$ to server $s_j$ is as follows:

$$t_{i,j}^{trans} = \begin{cases} 0 & s_j = s_0 \\ \frac{d_i}{r_{0,1}} & s_j = s_1 \\ \frac{d_i}{r_{0,1}} + \frac{d_i}{r_{edge}} & s_j \in \{s_2, \ldots, s_m\} \\ \frac{d_i}{r_{0,1}} + \frac{d_i}{r_{1,m+1}} & s_j = s_{m+1} \end{cases}$$

Equation (2)

In this context, r represents the transmission rate. To simplify the model, r depends solely on the network bandwidth B between the user' device and the target node, specifically the network bandwidth from local $s_0$ to $s_j$. Additionally, this paper does not consider the transmission time required to return the computed results to the local device after computation is completed.

c. The waiting time for task $v_i$ before it begins execution:

$$t_{i,j}^{wait} = \begin{cases} 0 & \text{pre}(v_i) = \varnothing \\ \max\left\{ \max\limits_{v_k \in \text{pre}(v_i)} (t_k^{wait} + t_{k,h}^{trans} + t_{k,h}^{C}), \ t_j^{last} \right\} & \text{pre}(v_i) \neq \varnothing, \ s_h \in S \end{cases}$$

Equation (3)

Before task $v_i$ begins execution, it must first wait for all its preceding tasks to be completely unloaded, transmitted, and executed. Additionally, each server can only handle one task at any given time; therefore, the start time of task $v_i$ must also be greater than the time spent waiting in queue to reach the designated unloading server.

Here, $t_{i,j}^{last}$ represents the queuing time for task $v_i$ upon arrival at server $s_j$, which can be calculated based on the queue model. For simplicity, we assume the use of the basic M/M/1 queue model for these calculations.

$$t_{i,j}^{last} \approx W_j = \frac{\lambda}{\mu_j(\mu_j - \lambda)}$$

Equation (4)

In this context, $W_j$ represents the average waiting time, $\lambda$ indicates the average arrival rate, and $\mu_j$ denotes the average service rate for $s_j$ assuming that all values are known in a static scenario.

Thus, the completion time for task $v_i$ is:

$$t_i^{fin} = t_{i,j}^{wait} + t_{i,j}^{C} + t_{i,j}^{trans}, \ s_j \in S$$

Equation (5)

In summary, the goal of delay optimization is:

$$\min T_{total} = \max_{v_i \in V} t_i^{fin}$$

Equation (6)

$$\text{S.t.} \ \sum_{i=1}^{n} x_{i,j} = 1$$

Equation (7)

$$x_{i,j} \in \{0,1\}$$ Equation (8)

$$t_{i,j}^{wait} \geq \max\left( t_k^{fin}, t_{i,j}^{last} \right), v_k \in \mathrm{pre}(v_i), s_j \in S$$ Equation (9)

Equation (7) ensures that each task $v_i$ can only be executed on one device; Equation (8) is a binary variable constraint, meaning that a task can either be offloaded or not offloaded, guaranteeing that each task $v_i$ can be processed; Equation (9) addresses the dependency constraints between tasks.

2）Energy Consumption Optimization Objective

The energy consumption model only considers the energy costs borne by the local user, namely the energy consumed for local computation and the energy required for offloading transmission, without accounting for the energy costs incurred during computation and transmission at the edge and cloud layers.

a. Energy consumption of task $v_i$ during local computation:

$$E^{local} = x_{i,0}(P^C \times T^C), \quad v_i \in V$$ Equation (10)

Among them, PC represents the average computational power consumption of the CPU, while TC indicates the time required for task execution, calculated according to equation (1).

b. When considering only the transmission consumption for offloading tasks to s1, the following applies:

$$E^{trans} = P^{trans} \cdot \sum_{i=1}^{n} \sum_{k=1}^{m+1} x_{i,k} d_i, \quad s_k \in \{s_1, s_2, \ldots, s_{m+1}\}$$ Equation (11)

Among them, Ptrans represents the average transmission power consumption of the local device. In summary, the total energy consumption is:

$$E_{total} = E^{local} + E^{trans}$$ Equation (12)

3）The overall system overhead target is

$$\min Z = \alpha T_{total} + (1-\alpha) E_{total}$$ Equation (13)

Ttotal represents the total delay incurred by the task, and $\alpha$ denotes the weight factor for delay. When $\alpha$ =1, it indicates that the system is delay-sensitive and only considers delay costs. When $\alpha$ =0 , it signifies that the system is energy-sensitive and focuses solely on energy consumption costs.

### 3.2 RESEARCH ON OFFLOADING ALGORITHM BASED ON IMPROVED GENETIC ALGORITHM

#### 3.2.1 POPULATION INITIALIZATION

This section assumes a population size of s, a number of user tasks equal to n, and m+1 edge nodes (including the cloud).

A chromosome is initialized with a length of n, where gene values are randomly generated within the range of 0 to m+1, using integer encoding. The chromosome is represented as $C_i = \{c_1, c_2, \ldots, c_n\}$.

For example, if Ci={0,1,2,3,4,1,0}, it indicates that task v1is executed by server s0 (x1,0=1), task v2 by server s1(x2,1=1), task v3by server s3 (x3,2=1), task v4 by server s3 (x4,3=1), and task v6 by server s1 (x6,1=1), and so forth. The remaining unmentioned values are xi,j=0.

#### 3.2.2 SELECTION

To select suitable individuals for genetic processing from the population, this section combines elitism and roulette wheel selection methods. Based on the fitness function, individuals with higher fitness from the parent population are preferentially selected to advance to the next generation, followed by using the roulette wheel method to select the remaining individuals, thereby forming a new generation. Chromosomes with higher fitness have a greater probability of being selected, while individuals with lower fitness still maintain a chance to pass on their genes.

The fitness of each individual is calculated using the fitness function, as detailed in Table 2, followed by the calculation of the selection probability for each individual.

**Table 2 The calculation process of the fitness**

| | |
|---|---|
| Input: chromosome $c_i$ | |
| 1 | For $task,s_j$ in chromosome Ci |
| 2 | Calculate $t_{task,s_j}^{wait}$ by (3-3) |
| 3 | Calculate $t_{task}^{fin}$ by $x_{task,s_j}$ with (3-5) |
| 4 | If tmax < ttask : tmax = ttask |
| 5 | Calculate Etotal by $x_{task,s_j}$ with (3-12) |
| 6 | Calculate Z by (3-13) |
| 7 | Output $Z_{Ci}$ |

1) Calculate the fitness function value based on the fitness assessment:

$$Fitness_{C_i} = \frac{1}{Z_{Ci}} \quad \text{Equation (14)}$$

Here, Ci represents a chromosome, and ZCidenotes the total system overhead corresponding to the unloading scheme associated with that chromosome, as calculated according to Equation (13).

2) Calculate the probability of Ci being selected, q(Ci)

$$q(C_i) = \frac{Fitness_{C_i}}{\sum_{i=k}^{s} Fitness_{C_k}} \quad \text{Equation (15)}$$

### 3.3 IMPROVEMENT STRATEGY FOR CROSSOVER OPERATION BASED ON ANT COLONY PHEROMONE CONCENTRATION

Crossover operation is a primary search operator in genetic algorithms, and an effective crossover approach allows offspring to inherit superior genes from the previous generation. However, traditional genetic algorithm crossover involves randomly selecting two individuals to exchange a portion of their genes. This method often converges quickly in the early stages, making it susceptible to "premature convergence", and exhibits poor local optimization capabilities, leading to suboptimal iterative results[18]. In this section, we focus on improving the crossover operation by introducing a pheromone concentration-based crossover mechanism, which can effectively prevent the generation of duplicate or low-quality offspring and enhance the algorithm's search efficiency and solution quality. The specific steps are as follows:

1) Randomly select a chromosome Ci from the population, initializing $C_{best}$=Ci. Assume the initial pheromone concentration is R0, the current concentration begins at RX=R0, and the termination concentration is Rf. The pheromone will evaporate by $\Delta R$ for N cycles.

2) Randomly select two points for exchange, forming a new individual Cx and calculate the fitness difference $\Delta\, fitness = fitness_{C_x} - fitness_{C_i}$ ;

3) If $\Delta\, fitness > 0$, then $C_{best}$=Cx ;

4) If $\Delta\, fitness < 0$, check if $e^{-\frac{\Delta fitness}{R_x}} \le \xi$, where $\xi = \text{rand}(0,1)$. If true, retain Ci=Ci; otherwise, update it to Ci=Cx ;

5) Check if the number of cycles has reached N. If not, return to step 2; if it has, proceed to step 6.

6) Calculate the updated pheromone concentration $R_X = R_X - \Delta R$. If $R_X < R_f$, output $C_{best}$; otherwise, return to step 2.

### 3.4 MUTATION

Mutation in genetic algorithms is a method for introducing new genetic diversity into the population by randomly altering the values of certain genes in individuals. The mutation operation aids the algorithm in exploring regions of the solution space that are not covered by the current population, thereby avoiding premature convergence to local optimal solutions. In the context of network task allocation and resource

optimization problems, the mutation operation may involve randomly selecting a task and changing the server it is assigned to, thereby exploring various potential task allocation schemes.

## IV. EXPERIMENTAL SIMULATION RESULTS AND ANALYSIS

### 4.1 EXPERIMENTAL ENVIRONMENT

The parameters of the simulation environment and algorithm hyper parameters used in this chapter are shown in Tables 3 and 4

**Table 3 Simulation parameters in the experiment**

| simulation parameter | value |
|---|---|
| task data size | 500kb~1500kb |
| period required for task processing | 50~150 |
| average server queuing time | 1s |
| local computing power | 1.2GHz |
| edge computing capability | 16GHz |
| cloud computing capability | 64GHz |
| local - edge bandwidth | 10M |
| edge - edge bandwidth | 30M |
| edge - cloud bandwidth | 50M |
| locally calculated power PC | 1W |
| local transmission power Ptrans | 0.5W |

**Table 4 Parameters used by the algorithm**

| Hyper parameter | value |
|---|---|
| number of servers | 7 |
| number of tasks | 20 |
| population size | 20 |
| number of iterations | 100 |
| delay weight | 0.5 |
| crossover probability | 0.8 |
| variation probability | 0.1 |

### 4.2 COMPARATIVE EXPERIMENTS

This section designs two sets of experiments to simulate task offloading under the marginal scenarios discussed in this chapter, verifying the superiority of the improved genetic algorithm ACGA for offloading optimization. By comparing it with the traditional genetic algorithm GA, the ACGA algorithm demonstrates significant advantages in reducing task execution time and conserving energy.

1) The first set of simulation experiments proposed in this chapter compares the latency overhead under different numbers of tasks, specifically between local execution and offloading execution using the ACGA algorithm, with a weight coefficient $\alpha$ =1:

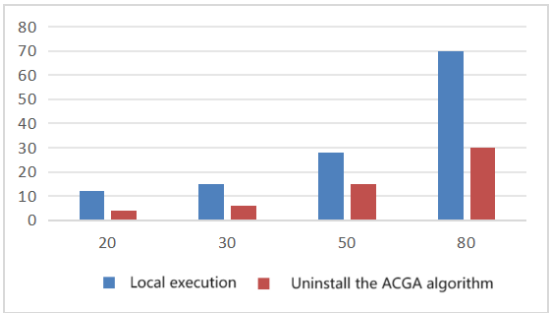
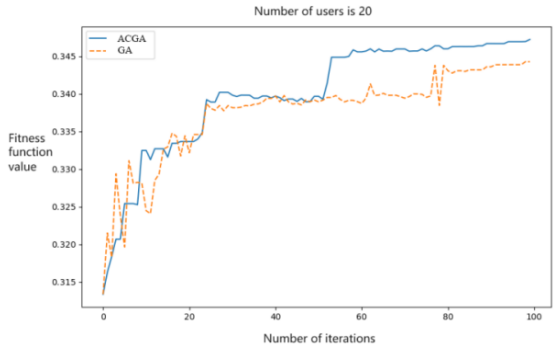


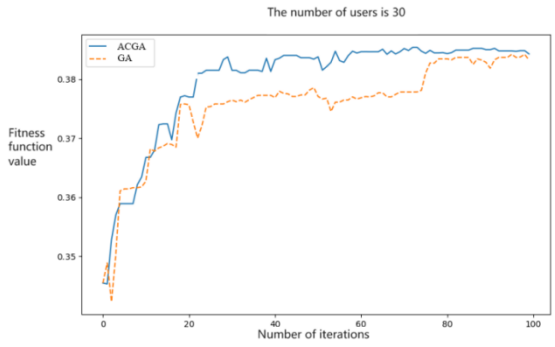

**Fig. 4 Local execution and uninstall execution system delay overhead comparison**

In Figure 4, it can be observed that under varying numbers of tasks, the utilization of the ACGA algorithm for offloading execution significantly reduces system latency compared to executing tasks entirely locally. This reduction decreases user waiting time and can enhance service quality.

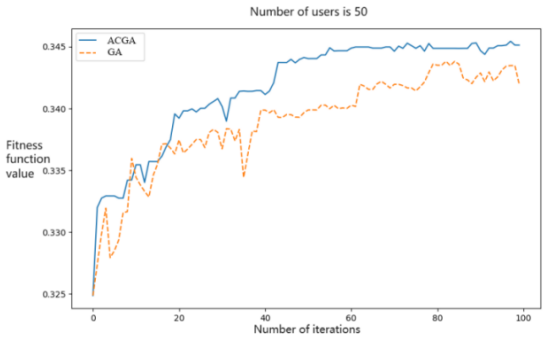
2) In this chapter, a comparison is made between the total overhead of the ACGA algorithm and traditional GA under different task quantities:



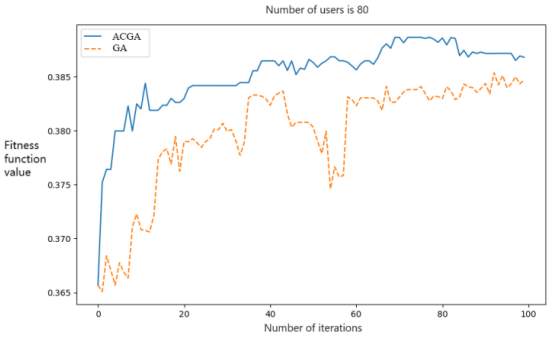

**(a)**




**(b)**




**(c)**




**(d)**

**Fig. 5 The relationship between the ACGA algorithm and GA and the total overhead under different task sets**

Figures 5 (a), (b), (c), and (d) depict the process of the algorithm iterating 100 times with task quantities of 20, 30, 50, and 80, respectively. It is evident that the Ant Colony Genetic Algorithm (ACGA) consistently outperforms the Genetic Algorithm (GA) as the complexity of the task set increases, particularly in terms of search capability in later stages. This superiority arises from the pheromone evaporation operation in the ant colony algorithm, which allows it to escape local optima, while the global search capability of the genetic algorithm ensures a greater number of high-quality starting points, ultimately leading to improved solutions in global optimization problems.

## V. CONCLUSION

This paper presents research on offloading strategies in a scenario with a single user and multiple edge nodes. The task set generated by the single user exhibits certain dependencies, and the total cost of the system model is defined as the weighted sum of local energy consumption and latency. A parametric modeling derivation process is specifically conducted, and an improved genetic algorithm, ACGA, is introduced as a solution method. Finally, simulation experiments are conducted, and the experimental results indicate that the ACGA algorithm introduced in this chapter significantly reduces system latency costs. Moreover, in terms of total cost, its convergence performance is superior to that of traditional genetic algorithms (GA), particularly demonstrated by its enhanced search capability in the later stages, enabling it to find better solutions for global optimization problems, resulting in lower system costs.

## REFERENCES

[1]. Kim W J J O T. Cloud computing: Today and tomorrow [J]. 2009, 8(1): 65-72.
[2]. Dai P, Hu K, Wu X, et al. A probabilistic approach for cooperative computation offloading in MEC-assisted vehicular networks [J]. 2020, 23(2): 899-911.
[3]. Shin D-H J T, Informatics. The role of affordance in the experience of virtual reality learning: Technological and affective affordances in virtual reality [J]. 2017, 34(8): 1826-36.
[4]. Hu Y C, Patel M, Sabella D, et al. Mobile edge computing—A key technology towards 5G [J]. 2015, 11(11): 1-16.
[5]. Jaisimha A, Khan S, Anisha B, et al. Smart transportation: an edge-cloud hybrid computing perspective; proceedings of the Inventive Communication and Computational Technologies: Proceedings of ICICCT 2019, F, 2020 [C]. Springer.
[6]. Khan L U, Yaqoob I, Tran N H, et al. Edge-computing-enabled smart cities: A comprehensive survey [J]. 2020, 7(10): 10200-32.
[7]. Xie R, Tang Q, Qiao S, et al. When serverless computing meets edge computing: Architecture, challenges, and open issues [J]. 2021, 28(5): 126-33.
[8]. Zhang J, Chen B, Zhao Y, et al. Data security and privacy-preserving in edge computing paradigm: Survey and open issues [J]. 2018, 6: 18209-37.
[9]. Zhang P, Zhou M, Fortino G J F G C S. Security and trust issues in fog computing: A survey [J]. 2018, 88: 16-27.
[10]. Zhang Y, Liu H, Jiao L, et al. To offload or not to offload: An efficient code partition algorithm for mobile cloud computing; proceedings of the 2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET), F, 2012 [C]. IEEE.
[11]. Shan X, Zhi H, Li P, et al. A survey on computation offloading for mobile edge computing information; proceedings of the 2018 IEEE 4th international conference on big data security on cloud (BigDataSecurity), IEEE international conference on high performance and smart computing,(HPSC) and IEEE international conference on intelligent data and security (IDS), F, 2018 [C]. IEEE.
[12]. Holland J H. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence [M]. MIT press, 1992.
[13]. Li K, Tang X, Veeravalli B, et al. Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems [J]. 2013, 64(1): 191-204.