Open Access Library Journal
2023, Volume 10, e10910
ISSN Online: 2333-9721
ISSN Print: 2333-9705

Task Offloading Scheduling with Time Constraint for Optimizing Energy Consumption in Edge Cloud Computing

Shufa Wen[1], Hongzhi Xu[2]

[1]School of Communication and Electronic Engineering, Jishou University, Jishou, China
[2]School of Computer Science and Engineering, Jishou University, Zhangjiajie, China
Email: wenshufa2021@163.com

How to cite this paper: Wen, S.F. and Xu, H.Z. (2023) Task Offloading Scheduling with Time Constraint for Optimizing Energy Consumption in Edge Cloud Computing. *Open Access Library Journal*, **10**: e10910.
https://doi.org/10.4236/oalib.1110910

Received: October 22, 2023
Accepted: November 26, 2023
Published: November 29, 2023

Abstract

In this paper, an improved genetic algorithm with delay constraint was designed. When initializing the population, a greedy strategy was adopted to ensure that there were enough excellent genes in the initial population, a normal distribution ordering selection strategy was adopted when selecting the next generation, so that high-quality chromosomes have a greater probability of being selected, and an adaptive cross-mutation strategy was proposed to achieve dynamic probability when cross-mutation was carried out to avoid the problem that the algorithm was easy to fall into the local optimal solution in the later stage, and at the same time, a maximum subsegment crossover strategy and a discardable mutation strategy were proposed to solve the problem of individual solution deterioration after crossover.

Subject Areas

Offloading Scheduling

Keywords

Edge Cloud Computing, Offloading Scheduling, Improving Genetic Algorithms, Limited Latency, Optimal Energy Consumption

1. Introduction

With the popularization of smart terminal devices and the large-scale development of emerging technologies such as VR and driverless driving, their applications have become more complex, and the requirements for network performance, latency, and computing power are increasing day by day. In order to

cope with the rapid development of technology and improve user service quality, edge cloud computing emerged as the times require. By moving tasks to edge servers closer to users, task delays and energy consumption can be effectively reduced. Due to the limited edge cloud resources, due to the diversity and complexity of edge cloud computing offloading scenarios, different offloading strategies will usually produce different effects. How to reasonably offload the different tasks generated by mobile local devices to ensure the quality of service? The problem needs to be solved urgently [1]. At present, a large number of research on edge computing ignores the resources of the central cloud. At the same time, task models and offloading strategies often only consider the optimal energy consumption when the delay is not limited, ignoring the optimal energy consumption when the delay is limited. In response to the above problems, this paper takes dependent applications composed of serial tasks as the research object, combines the system model of a single local device, multi-edge cloud servers and a single central cloud server, and finally builds an optimal solution under delay-limited conditions. Energy consumption task model, and an improved genetic algorithm suitable for delay constraints is designed.

At present, many scholars have studied the problem of edge computing task offloading. These studies can be roughly divided into three categories, namely minimizing delay as the goal, minimizing energy consumption as the goal, and the trade-off between delay and energy consumption.

1) Minimize time delay

In order to shorten the computing time of the application, Chen *et al.* [2] modeled the delay minimization problem as a convex optimization model and used convex optimization to solve it, which effectively shortened the computing time. Yu *et al.* [3] proposed a complete polynomial time approximation scheme for parallel multi-applications, which can greatly shorten the execution delay of tasks.

2) Minimize energy consumption

Zhang *et al.* [4] proposed a total cost algorithm based on enumeration search algorithm and improved Lagrangian relaxation for the stochastic shortest path problem. The simulation results show that compared with local execution and cloud execution, significant savings in mobile energy consumption on the device. Wen *et al.* [5] optimized the configuration of clock frequency and data transmission to meet the requirements of delay constraints, and minimized energy consumption by solving constraint optimization problems. This method can reduce the energy consumption of the equipment while ensuring performance.

3) Trade-off between delay and energy consumption

Considering energy consumption and execution delay at the same time, Mao *et al.* [6] proposed a low-complexity sub-optimal algorithm strategy based on alternating minimization, which comprehensively considers the comprehensive cost of delay and energy consumption, which can effectively reduce system costs and Its effectiveness is verified through simulation results. Mahmoodi *et al.* [7]

comprehensively considered delay and energy consumption in their research and modeled the problem as a linear programming model to obtain a solution that minimizes system cost. Through the linear programming method, the delay and energy consumption can be balanced and the optimal system cost can be obtained.

The above-mentioned classification analysis of the current status of past research is summarized below. First of all, the above-mentioned research on minimizing delay and minimizing energy consumption are all studies on a single goal, and do not combine delay and energy consumption. However, the applicable scenarios are limited. Secondly, the above-mentioned studies on the trade-off between delay and energy consumption usually simply use parameters to weight the relationship between delay and energy consumption, and have never considered the optimal situation when either delay or energy consumption is limited. The situation lacks certain constraints. Finally, research on edge cloud computing also ignores the resources of the central cloud, and often only considers both terminal devices and edge devices.

Therefore, in view of the above shortcomings, there is an urgent need to study energy-optimal offloading strategies under delay constraints in the device-edge-cloud tripartite system model scenario. Therefore, this paper designs an improved genetic algorithm with time constraint (IGAWTC) that is suitable for time delay constraints to obtain optimal energy consumption under time delay constraints. At the same time, simulation experiments are conducted to compare with other offloading strategies. The proposed IGAWTC The energy consumption is the lowest when the delay is limited.

2. System Model

This paper mainly considers studying the optimal energy consumption of serial tasks under the condition of limited delay under the system model of multi-terminal devices, multi-edge cloud servers and central cloud servers, in which each task is only allowed to be processed by one of the three parties of terminal, edge and cloud. Execution, **Figure 1** shows the edge cloud computing system model.

The key to edge cloud computing is to solve tasks generated by terminal devices while taking into account the characteristics of the tasks. Therefore, this paper considers the system model to be a terminal-edge-cloud tripartite system, making full use of the performance of edge cloud servers and cloud servers to meet the task offloading strategy. The process of designing a system is as follows. First, the task generated by the terminal device will determine whether the task is executed locally. If it is executed locally, it will be processed directly on the terminal device. If it is not executed locally, the task will be uploaded and assigned to the edge server or cloud server for execution. Finally, the edge cloud server or the cloud server can return the results to the terminal device after completing the task execution.

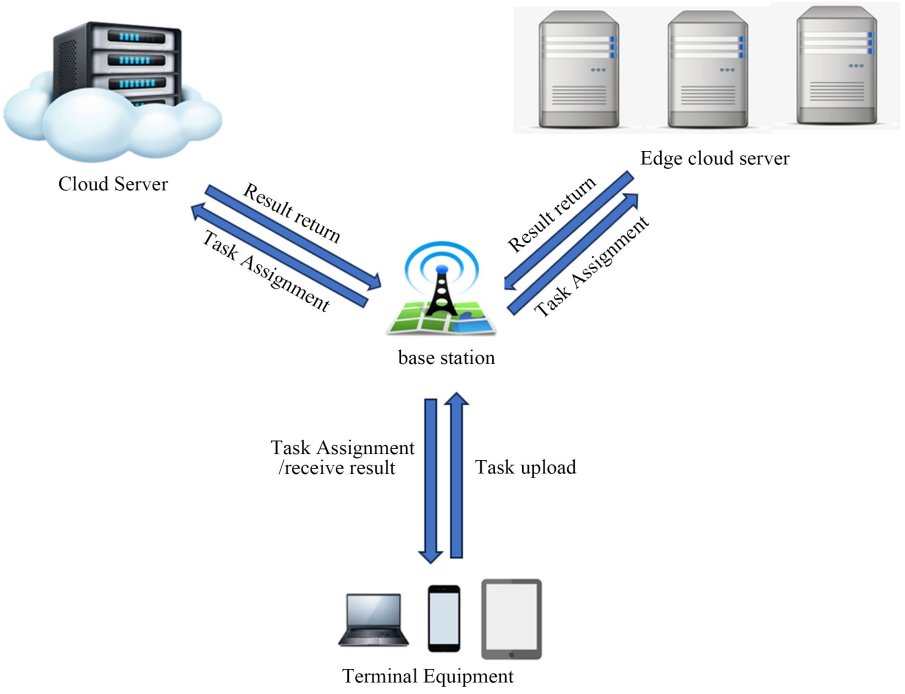

Figure 1. System model.

2.1. Delay Model and Energy Consumption Model

This article assumes that the system consists of a terminal device (the terminal device can be a mobile phone, a computer or an Internet of Things device, etc.), a central cloud server and m edge cloud servers. Among them, the application generated by the user can be disassembled into k sub-tasks, which can offload to the central cloud server or edge cloud server for processing, or it can be processed directly locally. Model it as $TASKS = \{task_1, task_2, \cdots, task_k\}$, and express the i-th task as a triplet as $task_i = \{D_i^{in}, Z_i, D_i^{out}\}$, where D_i^{in} represents the input data of the i-th task. Amount, Z_i represents the number of resources required to process the i-th task, and D_i^{out} represents the amount of data after the calculation of the i-th task is completed.

The terminal device is modeled as a five-tuple $U = \{f_l, p_l, p_{up}, p_{down}, p_{idle}\}$, where f_l represents the task computing capability of the local device, p_l represents the execution power of the local device, and p_{up} represents the upload of the local device. Power, p_{down} represents the local device receiving power, and p_{idle} represents the local standby power.

Use M to represent the set of edge cloud servers, that is $M = \{mec_1, mec_2, \cdots, mec_m\}$, and use a triple to represent the j-th edge cloud server, $mec_j = \{f_j^{mec}, r_j^{upmec}, r_j^{downmec}\}$, where, f_j^{mec} Indicates the task computing capability of the j-th edge cloud server, r_j^{upmec} indicates the upload rate of the j-th edge cloud server, and $r_j^{downmec}$ indicates the download rate of the j-th edge cloud server.

Similarly, the central cloud server is represented by a triplet, $cloud = \{f_{cloud}, r_{upcloud}, r_{downcloud}\}$, where f_{cloud} represents the task computing capability of the central

cloud server, $r_{upcloud}$ represents the upload rate of the central cloud server, and $r_{downcloud}$ represents the central cloud server download rates.

Use $O=\{o_1,o_2,\cdots,o_k\}$ to represent the offloading decision of k tasks, where, $o_i \in \{0,1,\cdots,m,m+1\}$. When $o_i = 0$, it means that the i-th task is a non-offloading task, and the task is executed on the local device; when $o_i \in \{1,\cdots,m\}$, it means that the i-th task is a task that is offloaded to the edge cloud for execution; when $o_i = m + 1$, it means that the i-th task is a task that is offloaded to the central cloud server for execution.

When the i-th task is a non-offloaded task, only the local execution delay and local execution energy consumption need to be considered, as shown in Equations (1) and (2):

$$t_i^l = \frac{Z_i}{f_l} \tag{1}$$

$$e_i^l = t_i^l \cdot p_l \tag{2}$$

When the i-th task is an offloaded task and executed on the edge cloud server, the task upload delay, execution delay and return delay need to be considered, as shown in Equation (3), and the energy consumption is as shown in Equation (4) Show:

$$t_i^{mec} = \frac{D_i^{in}}{r_i^{upmec}} + \frac{Z_i}{f_i^{mec}} + \frac{D_i^{out}}{r_i^{downmec}} \tag{3}$$

$$e_i^{mec} = \frac{D_i^{in}}{r_i^{upmec}} \times p_{up} + \frac{Z_i}{f_i^{mec}} \times p_{idle} + \frac{D_i^{out}}{r_i^{downmec}} \times p_{down} \tag{4}$$

When the i-th task is an offloaded task and executed on the central cloud server, similar to the edge cloud server, the task upload delay, execution delay and return delay need to be considered, as shown in Equation (5), the energy consumption is as shown in Equation (6):

$$t_i^{cloud} = \frac{D_i^{in}}{r_{upcloud}} + \frac{Z_i}{f_{cloud}} + \frac{D_i^{out}}{r_{downcloud}} \tag{5}$$

$$e_i^{cloud} = \frac{D_i^{in}}{r_{upcloud}} \times p_{up} + \frac{Z_i}{f_{cloud}} \times p_{idle} + \frac{D_i^{out}}{r_{downcloud}} \times p_{down} \tag{6}$$

According to the above calculation model, the $task_i$ delay calculation is as shown in Equation (7), and the energy consumption is as shown in Equation (8):

$$T_i = \begin{cases} t_i^l, & o_i = 0 \\ t_i^{mec}, & o_i \in \{1,\cdots,m\} \\ t_i^{cloud}, & o_i = m+1 \end{cases} \tag{7}$$

$$E_i = \begin{cases} e_i^l, & o_i = 0 \\ e_i^{mec}, & o_i \in \{1,\cdots,m\} \\ e_i^{cloud}, & o_i = m+1 \end{cases} \tag{8}$$

When all tasks are completed and executed, the total delay can be calculated as shown in Equation (9), and the total energy consumption is shown as Equa-

tion (10):

$$T_{total} = \sum_{i=1}^{k} T_i \tag{9}$$

$$E_{total} = \sum_{i=1}^{k} E_i \tag{10}$$

2.2. Problem Model

Given a set of serial tasks *TASKS*, terminal device *U*, edge cloud server *M* and central cloud server *cloud*, if all tasks are executed in the corresponding longest (or shortest) time, the system will generate the longest delay T_{max} (or the shortest delay T_{min}). The problem of this article is to minimize energy consumption under the given delay limit $T_{min} \le T_{limit} \le T_{max}$, right now:

$$\min E_{total} \tag{11}$$

$$\text{s.t.} \quad T_{total} \le T_{limit} \tag{12}$$

3. Improved Genetic Algorithm Suitable for Delay Constraints

3.1. Encoding and Decoding

When the genetic algorithm solves the offloading problem of edge cloud computing, each chromosome is defined as an offloading strategy. The value of the gene in the chromosome is an integer of [0, m + 1], where 0 means that the task is processed directly on the terminal device, and [1~m] indicates that the task is processed on different edge cloud servers, and m + 1 indicates that the task is processed on the central cloud server.

Assume that the number of subtasks generated by the terminal device is 10, then the task set is $TASKS = \{task_1, task_2, \cdots, task_{10}\}$, Table 1 Represents a computational offloading coding instance of 10 pending tasks.

Usually the algorithm initializes the offloading decision vector at the beginning, and then updates the offloading decision vector based on fitness. At the same time, this article assumes that the number of terminal devices is 1, the number of edge cloud servers is 5, and the number of cloud servers is 1. Therefore, this article can Assume that the i-th offloading decision vector of these 10 tasks is X_i = {0, 2, 4, 5, 6, 4, 3, 0, 5, 1}. This value is a randomly initialized value, and the minimum value is 0. The maximum value is 6, which demonstrates the encoding and decoding process. Among them, 0 means that the task is returned to the terminal device for processing, [1 - 5] means that the task is processed on different edge cloud servers, and 6 means that the task is processed on the central cloud server.

During the task offloading optimization process, it is necessary to know the task allocation on each node of the terminal, edge, and cloud. Therefore, it is necessary to decode the offloading decision vector. By observing the encoding method of the offloading decision vector, the decoding results are shown in Table 2.

Table 1. Offloading decision vector encoding example.

TASKS	$task_1$	$task_2$	$task_3$	$task_4$	$task_5$	$task_6$	$task_7$	$task_8$	$task_9$	$task_{10}$
Code	0	2	4	5	6	4	3	0	5	1

Table 2. Example of offloading decision vector decoding.

Node number	0	1	2	3	4	5	6
TASKS	$task_1$ $task_8$	$task_{10}$	$task_2$	$task_7$	$task_3$ $task_6$	$task_4$ $task_9$	$task_5$

It can be seen from the decoding results that $\{task_1, task_8\}$ is processed locally, $task_{10}$ is processed on the edge server mec_1, $task_2$ is processed on the edge server mec_2, $task_7$ is processed on the edge server mec_3, and $\{task_3, task_6\}$ is processed on the edge server. Processing is performed on mec_4, $\{task_4, task_9\}$ is processed on the edge server mec_5, and $task_5$ is processed on the central cloud server.

According to formulas (7) and (9), the delay and energy consumption after each task offloading is completed can be calculated. According to formulas (8) and (10), the delay and energy consumption of each offloading strategy can be calculated. Then the time delay is introduced. With extension constraints, a feasible solution to the problem model in this article can be obtained.

Finally, the IGAWTC algorithm is executed to optimize the offloading strategy. When the iteration is completed, the optimal solution to the problem model of this article can be obtained.

3.2. Algorithm Process

In IGAWTC, each individual is composed of a chromosome representing a set of genes. Each chromosome can be expressed as a solution to a problem, where each position represents a gene, and the length of each chromosome is k, which exactly fits the unloading decision vector of this article.

First, IGWTC initializes the operation to determine the value range of the fitness function, establish the accuracy and chromosome coding length, chromosome coding, and establish the population size, etc. Then the population is greedily initialized and the first generation population is randomly generated. Secondly, improved selection, crossover, mutation and other operations are performed to obtain the next generation population. Finally, after the iteration is completed, the optimal solution is output. The specific steps will be explained in detail in this section.

Figure 2 is the IGAWTC flow chart.

3.3. Greedy Initialization Population Strategy

The original distribution of the initial population has a great impact on the convergence of the algorithm. Unreasonable initial population distribution will

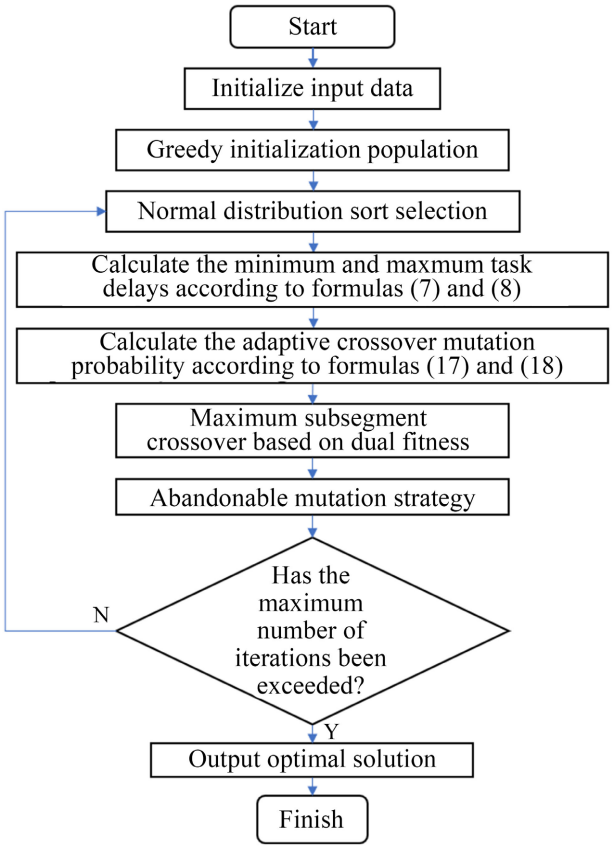

Figure 2. IGAWTC algorithm flowchart.

cause the algorithm to converge slowly, especially when the crossover and mutation operators of the genetic algorithm are poorly selected, or even premature convergence. The initial populations of standard genetic algorithms and most improved genetic algorithms are randomly generated in the search space. This method may result in not enough excellent genes when initializing the population, which is extremely detrimental to subsequent operations such as selection, crossover, and mutation.

Therefore, this paper proposes a greedy initialization population strategy for this problem. Two individuals of the population use a greedy algorithm. One individual selects the best delay for each task, one individual selects the best energy consumption for each task, and the other individuals are still initialized randomly, which ensures that there are enough excellent genes in the initial population and effectively improves the algorithm search efficiency.

3.4. Normal Distribution Ranking Selection Strategy

Since this article needs to consider the delay constraint, delay and energy consumption are usually mutually constrained. When the individual energy consumption is low but the delay does not meet the constrained delay condition, it is not a solution to the problem. When an individual meets the delay constraint but has high energy consumption, it means that the individual is not the optimal

solution to the problem. Obviously, the conventional use of delay sorting or energy consumption sorting selection is unreasonable. Therefore, this paper proposes a normal distribution sorting selection strategy to calculate the absolute value of individual delay and constraint delay and sort them in ascending order, as shown in Equation (13) as shown:

$$ABS_{species} = \left| T_{totol} - T_{limit} \right| \tag{13}$$

The normal distribution is a very widely used probability distribution, and its function expression is shown in Equation (14):

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \times \mathrm{e}^{\frac{(x-\mu)^2}{2\sigma^2}} \tag{14}$$

Since the mathematical expectation of the standard normal distribution is 0, and because this strategy calculates the absolute value of the solution, the generated solution values are all greater than 0. Therefore, after the population is sorted by absolute value, normalization is performed and the values are mapped to With $\mu = 0$ as the right half of the central symmetry axis, it is mapped to the interval (0, 2), as shown in Equation (15), and then the proportion probability of each individual is assigned through roulette, as shown in Equation (16) shown.

$$ABS'_{species} = 2 \times \frac{ABS_{species} - ABS^{min}_{species}}{ABS^{max}_{species} - ABS^{min}_{species}} \tag{15}$$

$$P(i) = \frac{f\left(ABS'^{i}_{species}\right)}{\sum_{i=1}^{N} f\left(ABS'^{i}_{species}\right)} \tag{16}$$

Among them, $ABS'_{species}$ represents the solution value after normalization of the absolute value, $ABS_{species}$ represents the absolute value without normalization, $ABS^{min}_{species}$ represents the individual with the smallest absolute value in the contemporary population without normalization, $ABS^{max}_{species}$ represents the individual with the largest absolute value in the contemporary population without normalization, and $ABS'^{i}_{species}$ represents the i-th value after normalization.

It can be seen that the normal distribution ranking selection strategy does not rely on individual fitness value information, but combines the delay constraints studied in this article and the standard normal distribution for ranking selection. It can inhibit the standard genetic algorithm based on fitness to a certain extent. Premature convergence and evolutionary stagnation caused by proportional selection strategies.

3.5. Adaptive Crossover Mutation Probability

In standard genetic algorithms, crossover and mutation probabilities are usually set to fixed values based on experience. If they are set too low, the standard genetic algorithm will not easily generate new individuals and will easily fall into local optimality. Setting it too high will turn the standard genetic algorithm into a purely random search algorithm. Standard genetic algorithms

usually require repeated experiments for each problem to determine the crossover and mutation probabilities, and it is often difficult to find the optimal probability.

Inspired by the literature [8], this paper proposes a nonlinear adaptive crossover mutation probability that combines fitness. It mainly based on the Sigmoid function to improve the crossover rate and mutation rate, a nonlinear adaptive crossover mutation probability adjustment formula combined with fitness is designed.

Equation (17) is the nonlinear adaptive crossover probability formula of IGAWTC. Equation (18) is the nonlinear adaptive mutation probability formula of IGAWTC.

Among them, T_{totol} is the independent variable, among which k_1, k_2 and k_3 are constants, which are set to 1.0, 0.1 and 0.2 respectively.

$$P(cr)=\begin{cases}\dfrac{k_1}{1+\mathrm{e}^{-20\times\left(\frac{T_{limit}-T_{totol}}{T_{limit}-T_{min}}\right)^6}}-k_2, & T_{totol}\le T_{limit}\\ P(cr)_{max}, & T_{totol}>T_{limit}\end{cases} \tag{17}$$

$$P(mu)=\begin{cases}\dfrac{k_3}{1+\mathrm{e}^{-20\times\left(\frac{T_{limit}-T_{totol}}{T_{limit}-T_{min}}\right)^6}}, & T_{totol}\le T_{limit}\\ P(mu)_{max}, & T_{totol}>T_{limit}\end{cases} \tag{18}$$

Nonlinear adaptive cross-mutation probability combined with fitness can mainly solve the following problems:

It can be seen from Equations (17) and (18) that when T_{totol} does not meet the delay constraints, it is given the highest crossover and mutation probability, which can quickly transform solutions that do not meet the constraints into solutions that meet the conditions.

When T_{totol} is close to T_{min}, it means that although the delay meets the constraints, it is obvious that better energy consumption can be found, so higher crossover and mutation probabilities are given to optimize it.

When T_{totol} is closer to T_{min}, it means that the time delay is close to the boundary, so the probability of crossover and mutation decreases non-linearly and smoothly, and the excellent body is better preserved.

3.6. Maximum Subsegment Crossover Strategy Based on Dual Fitness

In standard genetic algorithms, crossover operators usually adopt a single-point crossover or two-point crossover strategy to randomly determine crossover points on individual chromosomes. The crossover mixing speed of this strategy is slow, and the random selection of crossover points is usually easy to cause Excellent genes are exchanged, causing the individual's solution to become worse after crossing.

Therefore, this paper proposes a maximum sub-segment crossover strategy

based on dual fitness, considering the total delay and total energy consumption as delay fitness and energy consumption fitness. The specific operation is:

First, the individual that needs to be crossed is defined as the primary individual, and the individual that needs to be crossed with the primary individual is defined as the secondary individual. The selection of the secondary individual is determined by the primary individual.

The populations are sorted by delay fitness and energy consumption fitness respectively, and the secondary individuals are selected by judging whether the delay of the main individual satisfies the delay constraint. When the main individual satisfies the delay constraint, the individual with the lowest energy consumption fitness is selected. As a secondary individual, the purpose is to reduce the energy consumption of the primary individual. When the delay constraint is not satisfied, the individual with the lowest delay fitness is selected as the secondary individual, with the purpose of reducing the delay of the primary individual.

Then search for the gene sub-segments of the main individual and the secondary individual, starting from the largest sub-segment. When the secondary individual in the same gene sub-segment is better than the main individual, the genes of this sub-segment will be crossed, otherwise the genes will continue to be narrowed down. Search for subsections. If information feedback is obtained that there is no gene subsection in the secondary individual that is better than the primary individual, the crossover will be terminated to ensure that excellent genes are not lost. The process is shown in **Figure 3**.

3.7. Abandonable Mutation Strategy

The mutation operation is an important operation to jump out of the local optimal solution when the algorithm falls into the local optimal solution. In the standard genetic algorithm, it is usually easy to mutate excellent genes into bad genes during the mutation operation. Therefore, in order to make the algorithm jump out of the local optimal solution quickly, it is optimal and does not mutate the solution that satisfies the delay constraint into a solution that does not satisfy the delay constraint. This paper proposes an abandonable mutation strategy. First, it is judged whether the delay of the individual to be mutated satisfies the delay constraint. If it is satisfied, the mutation operation is performed and then the judgment is made. Whether the individual solution after mutation still satisfies the time delay constraint, if so, accept the mutation, otherwise give up the mutation. If the delay of the mutated individual does not meet the delay constraint, the mutation operation is performed directly.

4. Experimental Simulation and Results

4.1. Experimental Environment and Parameter Configuration

In order to verify the performance of the IGAWTC algorithm, a program was written in JAVA language and a simulation comparison experiment was de-

signed. The experimental running environment is: Intel(R) Core(TM) i7-9750H CPU @ 2.60 GHz, Windows 10 Professional laptop with 16 GB memory. In the experiment, a central cloud server, terminal equipment and 5 edge cloud servers were set up. All parameters were set according to the literature [9] as shown in Table 3.

In order to simplify the calculation amount in this article, the parameter numerical design range is small, and the parameter design logic conforms to the edge cloud computing model. Generally speaking, the computing power of cloud servers is the strongest, followed by edge cloud servers and local servers are the weakest. At the same time, parameters such as upload, download and execution power are also determined according to the logic of the actual device. Generally speaking, the standby power is the lowest and the upload power is the highest. The bandwidth is usually that the download bandwidth is better than the upload bandwidth. Just make sure the data is logical.

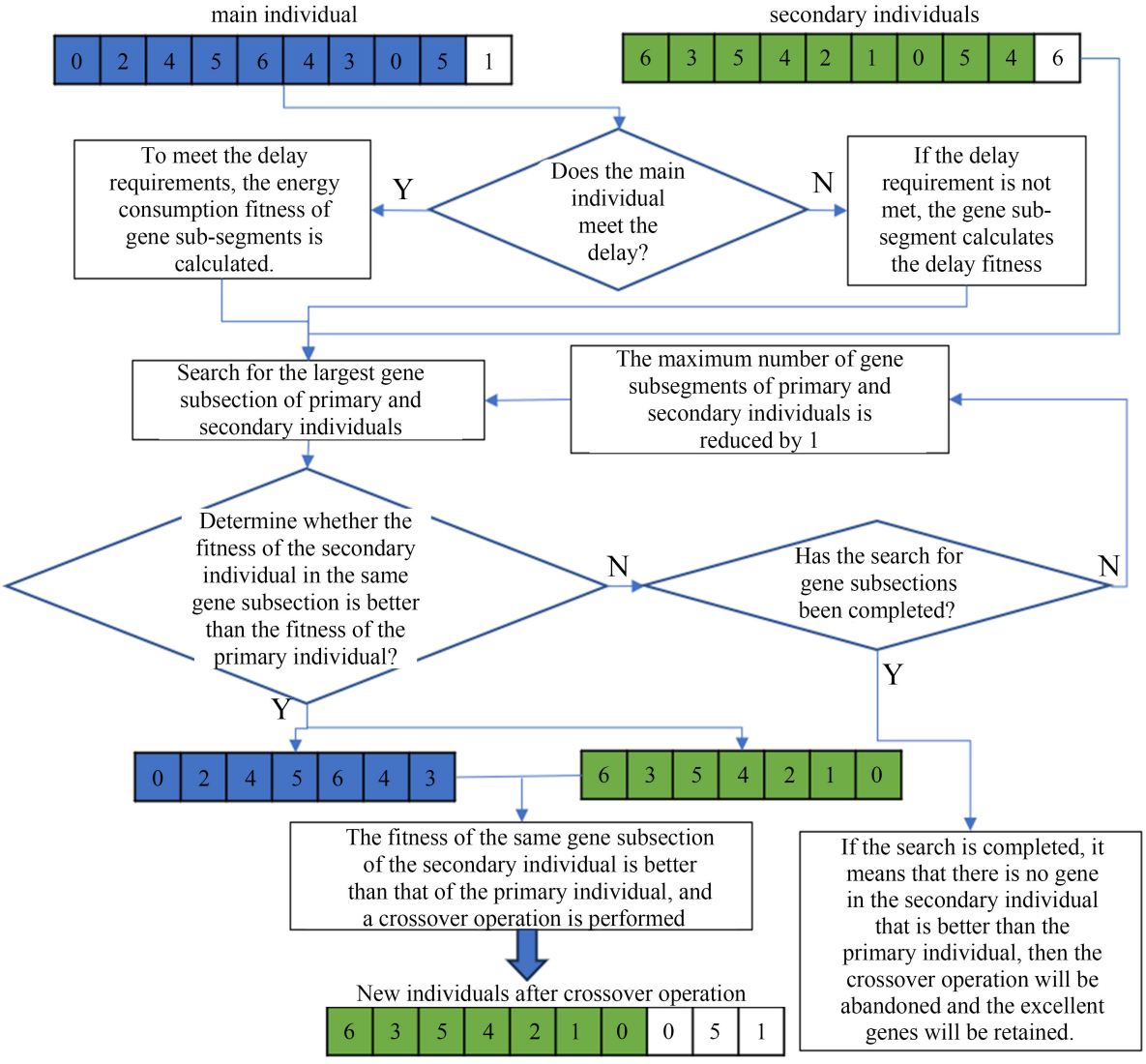

Figure 3. Maximum subsegment crossover strategy based on double fitness.

Table 3. Simulation parameter setting.

Symbol	Definition	Value
k	Number of subtasks	100 - 700
f_l	local computing power	2.5 GHz
p_l	local execution power	5.5 W
p_{up}	Upload power	3 W
p_{down}	Download power	2 W
p_{idle}	Local standby power	0.3 W
D_i^{in}	The amount of input data for the i-th task	100 - 350 Mb
Z_i	The number of resources required to process the i-th task	1 - 30 GHz
D_i^{out}	The amount of data after the i-th task calculation is completed	100 - 350 Mb
f_j^{mec}	The computing power of the jth edge cloud server	3.5 - 7.5 GHz
r_j^{upmec}	The jth edge cloud server upload bandwidth	35 - 39 Mbps
$r_j^{downmec}$	The download bandwidth of the jth edge cloud server	40 - 44 Mbps
f_{cloud}	Central cloud server computing power	15 GHz
$r_{upcloud}$	Central cloud server upload bandwidth	30 Mbps
$r_{downcloud}$	Central cloud server download bandwidth	35 Mbps
N	population size	30
$maxite$	The maximum number of iterations	100

4.2. Simulation

4.2.1. Task Environment Construction

The IGAWTC algorithm proposed in this article is mainly related to the Improve genetic algorithm (IGA) in the literature [10], the adaptive genetic algorithm (AGA) in the literature [11], the Standard genetic algorithm (SGA), And the random allocation method (Random) is compared, in which each algorithm adopts the optimal retention strategy.

This paper generates interval correspondence between the input data volume of the task, the number of resources required by the task, and the data volume after the task calculation is completed. That is, when $D_i^{in} \in [0,200)$, Z_i corresponds to the interval [0, 16), and D_i^{out} corresponds to the interval. [100, 200) generated. Correspond in sequence as shown in **Table 4**. At the same time, control is performed when tasks are generated to ensure that tasks are evenly distributed among various intervals.

In order to represent different delay constraints, the delay constraint calculation formula is designed: $T_{limit} = T_{min} + (T_{max} - T_{min}) \times \alpha$, where $\alpha \in [0,1]$, the smaller α means the tighter the delay constraint, and vice versa, the looser the delay.

The energy consumption that cannot meet the delay requirements is represented by the maximum value on the y-axis.

Table 4. Data volume interval corresponding table.

Symbol	Interval			
D_i^{in}	[100, 200)	[200, 250)	[250, 300)	300, 350)
Z_i	[1, 16)	[16, 20)	[20, 25)	[25, 30)
D_i^{out}	[100, 200)	[200, 250)	[250, 300)	[300, 350)

4.2.2. Experimental Design

Experiment 1: This experiment tests the impact of changes in delay constraints on each algorithm. The number of iterations is 100, and the number of subtasks generated by the terminal device is limited to 100, 400 and 700 respectively. When the delay constraint α value is reduced from 4/5 to 1/5 (that is, the delay is getting tighter), the energy consumption generated by each algorithm is shown in **Figures 4-6** respectively.

From the simulation results in the above figure, the following conclusions can be drawn:

1) Judging from the columnar data, under different delay constraints, IGAWTC's energy consumption to meet delay constraints is better than IGA, AGA, SGA and Random, indicating that IGAWTC has great advantages in reducing energy consumption under the delay constraint.

2) As the delay constraint continues to tighten, the energy consumption of the algorithm's satisfying delay is increasing. This is because when the delay constraint is tightened, the algorithm will look for an offloading location with lower delay, but it often takes lower latency means energy consumption will typically be higher. Then judging from the upward trend line in the figure, IGAWTC has the smallest growth potential compared to IGA, AGA and SGA. However, its Random cannot find the energy consumption that satisfies the delay constraint when the delay constraint is tightened to 3/5, and SGA cannot find the energy consumption that satisfies the delay constraint when it is tightened to 1/5. This shows that IGAWTC has better optimization performance than other algorithms when the delay is tightened.

3) When the number of tasks is 100 and the delay constraint is 4/5, it can be seen that the energy consumption that satisfies the delay found by each algorithm is basically the same. As the delay constraint continues to tighten to 2/5, it can be seen that by the time the optimization performance of SGA has developed a certain gap with the other three types of algorithms, and when it is tightened to 1/5, the optimization performance of Random and SGA has a significant gap compared with IGAWTC.

At the same time, as the number of tasks increases, it can be seen that the gap between IGAWTC and other algorithms is gradually widening. When the number of tasks is 700, it can be calculated that IGAWTC is better than IGA and AGA by more than 6%, and is better than other algorithms by more than 30%.

The above conclusion is generated because IGAWTC sorts the population in a better way and sets a more reasonable adaptive crossover mutation probability, and avoids the deterioration of the solution set to the greatest extent. Therefore,

IGAWTC satisfies the delay constraint in each case. The energy consumption is the smallest, indicating that IGAWTC has the best performance in searching for optimal energy consumption under different constraints compared with existing algorithms.

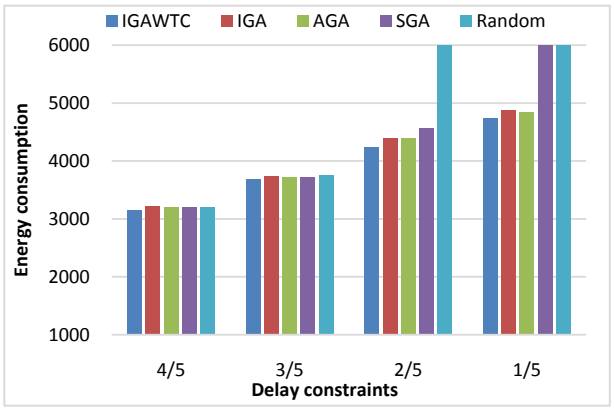

Figure 4. Optimal energy consumption under different delay constraints when the number of subtasks is 100.

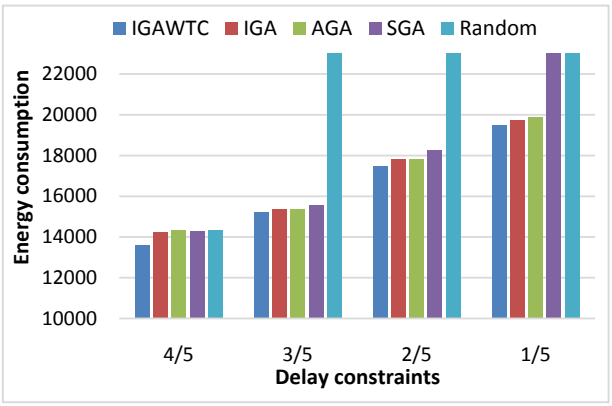

Figure 5. Optimal energy consumption under different delay constraints when the number of subtasks is 400.

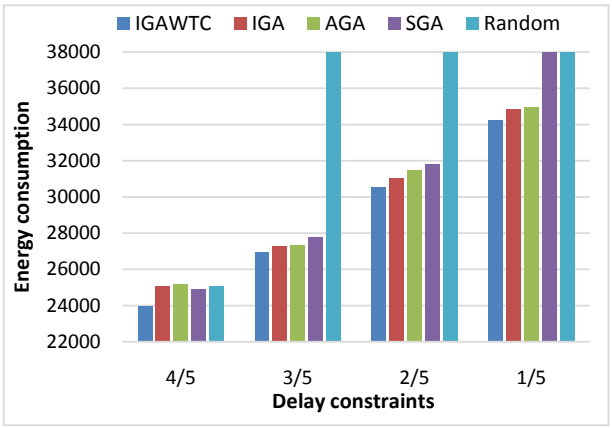

Figure 6. Optimal energy consumption under different delay constraints when the number of subtasks is 700.

Experiment 2: This experiment tests the impact of changes in the number of tasks on the performance of each algorithm. The number of iterations is 100. When the number of tasks generated by the terminal device increases from 100 to 600, set the delay constraint α to 4/5 and 2/5. The energy consumption generated by each algorithm is shown in Figure 7, Figure 8.

From the simulation results in the above figure, the following conclusions can be drawn:

1) When the delay constraint is 4/5, IGAWTC has the greatest advantage over other algorithms. As the delay constraint tightens to 2/5, it will increase the difficulty of algorithm optimization, so the advantages of IGAWTC are shrinking. But it can still be found to be better than other algorithms.

2) As the number of tasks increases, the energy consumption of each algorithm to meet the delay constraint gradually increases. However, it can be seen that since IGAWTC has the optimal energy consumption under different number of tasks, the time delay of IGAWTC to meet is the growth rate of constrained energy consumption is also the lowest. It is also found that as the number of tasks increases, the advantages of IGAWTC gradually increase.

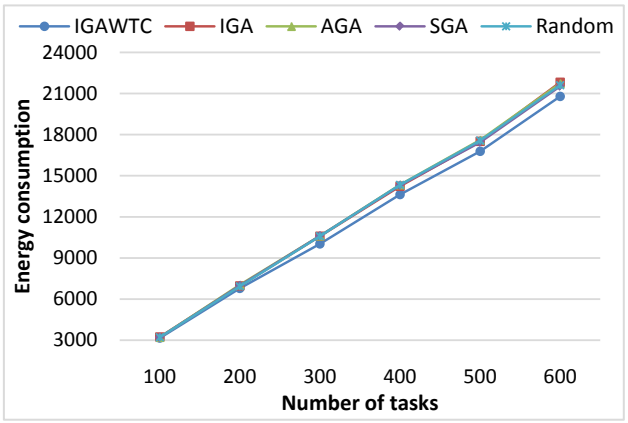

Figure 7. The optimal energy consumption of different task numbers when the delay constraint is 4/5.

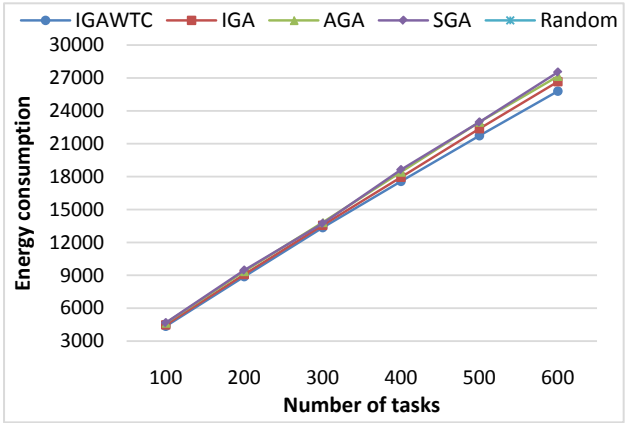

Figure 8. The optimal energy consumption of different task numbers when the delay constraint is 2/5.

Experiment 3: This experiment evaluates the convergence of each algorithm under delay constraints. The delay constraint is fixed at 4/5, and the number of tasks generated by the terminal device is limited to 100, 400, and 700 respectively. When the number of iterations increases from 1 to 100, the energy consumption generated by each algorithm is shown in Figures 9-11.

From the simulation results in the above figure, the following conclusions can be drawn:

1) From the perspective of the number of tasks, when the number of tasks is small, the gap between IGAWTC and other algorithms is small. As the number of tasks increases, it can be seen that the advantages of IGAWTC compared with other algorithms gradually increase. This is because other algorithms can also find better solutions when the number of tasks is small, but as the number of tasks increases, the difficulty of optimization by other algorithms increases. It also shows that IGAWTC has more stable optimization performance.

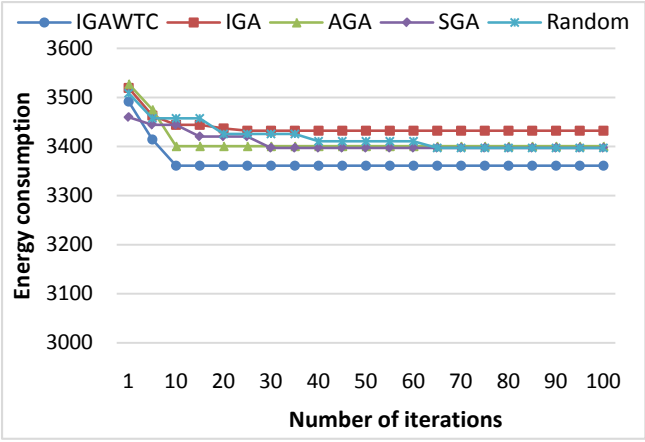

Figure 9. Algorithm convergence analysis when the number of tasks is 100.

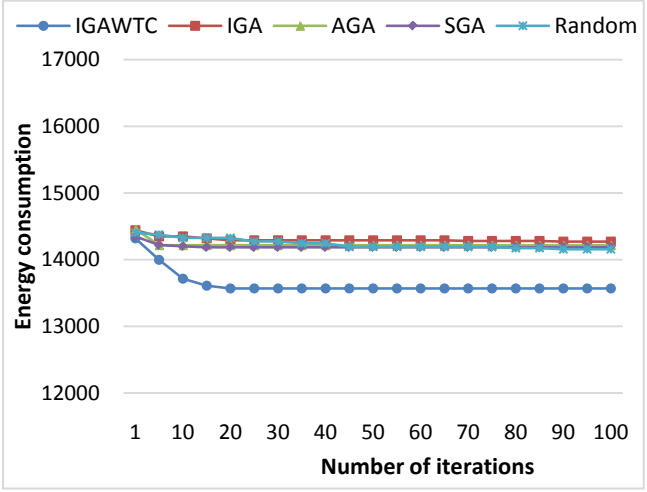

Figure 10. Algorithm convergence analysis when the number of tasks is 400.

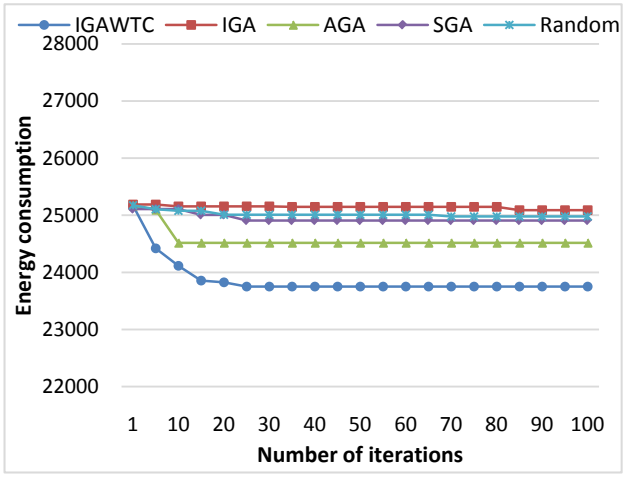

Figure 11. Algorithm convergence analysis when the number of tasks is 700.

2) From the perspective of convergence performance, compared with other algorithms, IGAWTC has little difference in the early stage of iteration, and begins to have greater advantages after stabilizing in the later stage. At the same time, it can also be seen that the convergence performance of IGAWTC is also better than other algorithms, the convergence speed is faster and the effect is better, which verifies that IGAWTC has faster convergence performance and better optimization performance than other algorithms.

5. Conclusion and Suggestion

This article considers that a large number of studies on edge cloud computing ignore the resources of the central cloud. At the same time, task models and offloading strategies often only consider the optimal energy consumption when the delay is not limited, ignoring the situation when the delay is limited. In order to achieve optimal energy consumption, an improved genetic algorithm suitable for delay constraints was designed, and a comparative experiment was conducted between IGAWTC and existing algorithms.

The conclusions obtained through three sets of experiments verify that IGAWTC has faster convergence speed and optimization performance. At the same time, it still has great advantages under different delay constraints and different number of tasks, indicating that the IGAWTC proposed in this article can be more effective. Effectively reduce energy consumption while meeting delay constraints.

In the next step of work, it is planned to consider expanding the serial task model into a parallel task model, which requires focusing on task dependencies and parallelism. At the same time, further in-depth research on offloading strategies is required.

Fund Project

This work was supported by the National Science Fund subsidized project

(62062036).

Conflicts of Interest

The authors declare no conflicts of interest.

References

[1] Li, Y., Wang, X., Gan, X., Jin, H., Fu, L. and Wang, X. (2020) Learning-Aided Computation Offloading for Trusted Collaborative Mobile Edge Computing. *IEEE Transactions on Mobile Computing*, **19**, 2833-2849. https://doi.org/10.1109/TMC.2019.2934103

[2] Chen, M. and Hao, Y. (2018) Task Offloading for Mobile Edge Computing in Software Defined Ultra-Dense Network. *IEEE Journal on Selected Areas in Communications*, **36**, 587-597. https://doi.org/10.1109/JSAC.2018.2815360

[3] Yu, R., Xue, G. and Zhang, X. (2018) Application Provisioning in FOG Computing-Enabled Internet-of-Things: A Network Perspective. *IEEE INFOCOM* 2018—*IEEE Conference on Computer Communications*, Honolulu, 16-19 April 2018, 783-791. https://doi.org/10.1109/INFOCOM.2018.8486269

[4] Zhang, W., Wen, Y. and Wu, D. (2015) Collaborative Task Execution in Mobile Cloud Computing under a Stochastic Wireless Channel. *IEEE Trans on Wireless Communications*, **14**, 81-93. https://doi.org/10.1109/TWC.2014.2331051

[5] Wen, Y., Zhang, W. and Luo, H. (2012) Energy-Optimal Mobile Application Execution: Taming Resource-Poor Mobile Devices with Cloud Clones. 2012 *Proceedings IEEE INFOCOM*, Orlando, 25-30 March 2012, 2716-2720. https://doi.org/10.1109/INFCOM.2012.6195685

[6] Mao, Y., Zhang, J. and Letaief, K. (2017) Joint Task Offloading Scheduling and Transmit Power Allocation for Mobile-Edge Computing Systems. 2017 *IEEE Wireless Communications and Networking Conference* (*WCNC*), San Francisco, 19-22 March 2017, 1-6. https://doi.org/10.1109/WCNC.2017.7925615

[7] Mahmoodi, E., Uma, R. and Subbalakshmi, K. (2019) Optimal Joint Scheduling and Cloud Offloading for Mobile Applications. *IEEE Transactions on Cloud Computing*, **7**, 301-313. https://doi.org/10.1109/TCC.2016.2560808

[8] Liu, W. and Zhou, Y. (2009) Modifed Inertia Weight Particle Swarm Optimizer. *Computer Engineering and Application*, **45**, 46-48.

[9] Qin, Z., Su, J., Liu, X. and Zhu, M. (2022) Energy-Aware Workflow Real-Time Scheduling Strategy for Device-Edge-Cloud Collaborative Computing. *Computer Integrated Manufacturing Systems*, **28**, 3122-3130.

[10] Chen, L., Wang, Z. and Mo, Y. (2022) Improved Genetic Algorithms for Adaptive Replication Crossover and Mutation. *Computer Simulation*, **39**, 323-326+362.

[11] Wu, Z., Shao, H. and Wu, X. (1999) A New Adaptive Genetic Algorithm and Its Application in Multimodal Function. *Contorl Theory and Applications*, 127-129.