electronics

MDPI

Article

An Improved Gravitational Search Algorithm for Task Offloading in a Mobile Edge Computing Network with Task Priority

Ling Xu [1], Yunpeng Liu [2], Bing Fan [3], Xiaorong Xu [1], Yiguo Mei [4,*] and Wei Feng [1,*]

[1] School of Communication Engineering, Hangzhou Dianzi University, Hangzhou 310018, China; 211080022@hdu.edu.cn (L.X.); xuxr@hdu.edu.cn (X.X.)
[2] Zhejiang Haikang Zhilian Technology Co., Ltd., Hangzhou 311113, China
[3] Frontier Technology Service Center, Hangzhou Dianzi University, Hangzhou 310018, China; fanbing@hdu.edu.cn
[4] Huaxin Consulting Co., Ltd., Hangzhou 310051, China
* Correspondence: meiyg.hx@chinaccs.cn (Y.M.); fengwei@hdu.edu.cn (W.F.)

Abstract: Mobile edge computing (MEC) distributes computing and storage resources to the edge of the network closer to the user and significantly reduces user task completion latency and system energy consumption. This paper investigates the problem of computation offloading in a three-tier mobile edge computing network composed of multiple users, multiple edge servers, and a cloud server. In this network, each user's task can be divided into multiple subtasks with serial and parallel priority relationships existing among these subtasks. An optimization model is established with the objective of minimizing the total user delay and processor cost under constraints such as the available resources of users and servers and the interrelationships among the subtasks. An improved gravitational search algorithm (IGSA) is proposed to solve this optimization model. In contrast with the other gravitational search algorithm, the convergence factor is introduced in the calculation of the resultant force and the crossover operation in a genetic algorithm is performed when generating the new particles during each iteration. The simulation results show that the proposed IGSA greatly improves the system performance compared with the existing algorithms.

Keywords: mobile edge computing; computing offloading; serial and parallel priority; improved graphgravitational search algorithm; genetic algorithm

Citation: Xu, L.; Liu, Y.; Fan, B.; Xu, X.; Mei, Y.; Feng, W. An Improved Gravitational Search Algorithm for Task Offloading in a Mobile Edge Computing Network with Task Priority. *Electronics* **2024**, *13*, 540. https://doi.org/10.3390/electronics13030540

Academic Editor: Martin Reisslein

Received: 26 November 2023
Revised: 21 January 2024
Accepted: 25 January 2024
Published: 29 January 2024

1. Introduction

With the development of the Internet of Things (IoT) in recent years, mobile user devices (UEs) such as smartphones and laptops have set off a new wave. The increased portability and computing power of mobile devices makes them an integral part of our lives. It has also led to the emergence of new applications such as augmented reality/virtual reality (AR/VR), online gaming and image processing on devices. However, the limited battery life of mobile devices and the low latency requirements of these applications have increased the need for new network models [1].

To address this issue, mobile edge computing offloading techniques have been deployed to handle the offloading tasks generated by user terminals to edge servers. Unlike traditional cloud computing, Mobile edge computing (MEC) [2] deploys computing and storage resources at the edge of a mobile network to provide an information technology (IT) service environment and cloud computing capabilities for mobile networks, thus providing users with ultra-low latency, low power consumption, and high broadband network service solutions [3–5]. As one of the key technologies in MEC, computing offloading [6–10] enables terminal devices to unload partial or all computational tasks to mobile edge servers for assistance, aiming to address the inherent issues of limited storage space, inadequate computing power, and energy constraints on terminal devices.

In practical applications of mobile edge computing (MEC), the process of offloading computation requires the collaborative consideration of both mobile terminals and MEC servers to address the following issues: what to offload, how to offload, how much to offload, and to whom and how much bandwidth and computational resources should be allocated [11–13]. Currently, research in this field typically involves two steps to facilitate this process: task offloading decision and resource allocation and offloading execution. the first step involves making decisions regarding which tasks or computations should be offloaded, and which terminal should be matched with which server or multiple servers. Various factors are considered, such as task characteristics (e.g., computation-intensive or data-intensive), network conditions, energy consumption, latency requirements, and the number and capacity of severs. Decision-making techniques, such as optimization algorithms [14–16] or machine learning models [17–19], are employed to determine the most suitable tasks offloading decision. Once the decision to offload certain tasks is made, the next step is to allocate the necessary resources to perform the offloading and computation of the tasks. This process is known as resource allocation and offloading execution. During this process, it is necessary to determine the required amount of bandwidth and computational resources, as well as the allocation strategy for these resources. Techniques like dynamic resource allocation, load balancing, and task scheduling are utilized to ensure the efficient utilization of resources and improved performance.

Inspired by the aforementioned facts, in order to minimize the total completion latency and processor cost of subtasks with priority constraints at the client-side, the tasks are offloaded to both edge servers and cloud servers. This paper proposes a new offloading algorithm called the improved GSA-based offloading (IGSA) algorithm. The IGSA algorithm incorporates a convergence factor to accelerate the search for optimal solutions in the search space. Additionally, it introduces crossover operations from genetic algorithms to improve the optimization results. The main contributions of this paper can be summarized as follows.

i This paper investigates the resource allocation problem in a multi-user mobile edge network based on the collaboration between cloud servers and edge servers. Specifically, the study takes into account the presence of both MCC servers, MEC servers, and subtasks with concurrent serial and parallel relationships.

ii To address the formulated user latency and processor cost minimization problem, an improved gravitational search algorithm (IGSA) is proposed. In contrast to the other gravitational search algorithm, the convergence factor is introduced in the calculation of the resultant force and the crossover operation in a genetic algorithm is performed when generating the new particles during each iteration. We conducted comprehensive experiments and evaluations to validate the performance and effectiveness of the proposed improved GSA (IGSA) algorithm.

The remainder of this paper is structured as follows. Section 2 introduces the previous work of other experts. Section 3 presents the system model and formulates the problem. Section 4 details the design of the proposed algorithms. Section 5 analyzes the proposed algorithm in this paper through numerical simulations. Finally, Section 6 concludes the paper.

2. Related Work

In recent years, with the development of intelligent terminals, there has been an emergence of more computation-intensive services and applications. Due to the limited storage and computational resources of end devices, mobile edge computing provides an alternative solution for data processing and storage. Among them, computation offloading, as a key technology, has attracted increasing attention. Through the in-depth exploration of this field, researchers have achieved fruitful results.

In [20], the authors studied a genetic and deep deterministic policy gradient (GADDPG)-based computation offloading scheme to achieve the optimal user experience quality. In [21], the author proposed an emerging method of deep reinforcement

learning for the dynamic clustering of IoT networks to solve the communication balance of IoT networks and the computational balance of edge servers. In [22], the authors proposed an improved multi-objective cuckoo search (IMOCS) algorithm to reduce the execution latency and energy consumption of user terminals. In [23], the author studied a deep reinforcement learning method that combines multiple neural networks to minimize the computational cost of the weighted sum of delay and energy consumption in dynamic environments with time-varying wireless fading channels. In [24], the authors investigated the policy gradient (PG) algorithm to achieve a low decision time and low task offloading time. In [25], the author studied a multi-agent power control algorithm to maximize the total transmission rate of all downlink transmissions. However, it is not difficult to notice that this category of research focuses on unordered, parallelizable, independent task models. Based on this, further investigation reveals that some research outcomes [26] have also considered the computation offloading of sequential tasks, i.e., taking into account the completion priority of tasks or subtasks. However, neither pure parallel nor pure sequential models are realistic as tasks in actual systems are a hybrid of both. Currently, only a small number of scholars have begun researching the complex task offloading problem in this hybrid model. For example, in [27], Anubhav Choudhary et al. proposed a problem of minimizing completion time and cost by considering the scenario of cloud computing. They presented a hybrid algorithm based on the meta-heuristic GSA and the heterogeneous earliest finish time (HEFT) heuristic to determine the monetary cost ratio (MCR) and the scheduling length ratio (SLR). However, their study only focuses on the scenario of cloud computing, which is relatively limited in scope. In [28], a heterogeneous computing system workflow scheduling based on the GSA is proposed. The design of this algorithm involves representing task dependencies using a new agent while preserving the constraints. The algorithm shows an improvement in terms of completion time, load balancing, and energy consumption. However, it does not consider the specific application of the algorithm in specific system scenarios.

The offloading decision [29] mainly solves the problem of how the mobile terminal decides how to offload, how much to offload, and what to offload. There are mainly two-part offloading and partial offloading. In [27], the authors proposed a hybrid algorithm based on GSA and heterogeneous earliest finish time [30] (HEFT) to minimize the completion time and total computational cost. However, maintaining the dependency constraints based on the HEFT algorithm requires $O(T^2 \times P)$ complexity. In [28], the authors proposed a GSA-based algorithm that uses a lower complexity algorithm to maintain the priority and proposed a new proxy method.

3. System Model

3.1. Network Model

Figure 1 shows the system model, which consists of a mobile cloud computing (MCC) server, multiple mobile edge computing (MEC) servers and multiple users (UEs). In this scenario, the user terminal devices communicate with the edge servers through the base stations. The edge servers receive task requests from the user terminal devices. The cloud server, as the central node of the entire system, is responsible for receiving task requests from edge servers and handling the centralized task scheduling and allocation of computing resources. Each server, including MCC and MEC servers, can only handle one task at a time. The MEC servers perform collaborative computing and offloading to meet the users' delay requirements and are co-located with the users-owned base stations (BSs). All the MEC servers can communicate with each other through the resource pool and the communication time between different MEC servers can be ignored. Within the network, there are $|\Re|$ users, $\Re = \{1, 2, 3 \ldots, R\}$ and each user can generate a task that can be subdivided into multiple subtasks, which can be executed concurrently or sequentially, depending on their priority relationship. $S = \{s_l | 1 \leq l \leq L\}$ is the server set, where $s_l = l$, and s_1 is the MEC server to which the user belongs, $s_2, s_3, \ldots,$ and s_{L-1} are the other MEC servers in the same resource pool, and s_L is the MCC server. Similarly, $F = \{f_l | 1 \leq l \leq L\}$ represents

the available computing resource set, where f_1 is the computing resources of the hosting MEC server, $f_2, f_3, \ldots$, and f_{L-1} are the other MEC servers' computing resources in the same resource pool, and f_L is the MCC server' computing resources, that is, the CPU cycle number that can be provided per second.

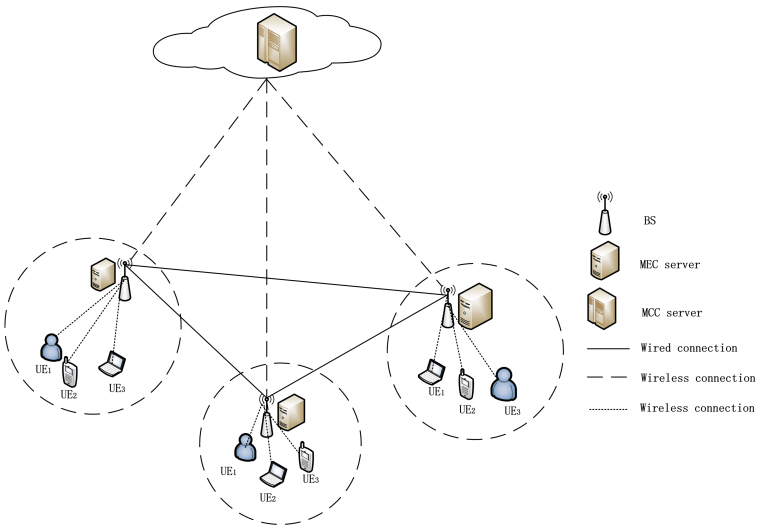


Figure 1. Network model.

3.2. Task Model

Figure 2 shows the task model, in which Figure 2(a), (b) represent the single-task and multi-task models, respectively. Each task can be divided into multiple sequential subtasks. As shown in the diagram, the number before the '-' symbol represents the task number, and the number after the '-' symbol represents the subtask number. In Figure 2(a), there is only one main task 1, while in Figure 2(b), there are three parallel main tasks 1, 8, and 15. The priority constraint relationship between the subtasks can be illustrated using a directed acyclic graph (DAG) $G = \langle V, E \rangle$. $V = \{v_i | 1 \leq i \leq I\}$ is the set of subtasks, which are also called nodes in the graph. E is the set of directed edges between subtasks, indicating the priority constraint relationship between subtasks, and $e_{ij} \in E$ means that task v_i must be executed before subtask v_j. Each subtask $v_i = \{d_i, g_i\}$, where d_i is the input data size of the ith subtask module, and g_i is the required CPU cycle to complete the task.

The subtasks without predecessor constraints are called entry nodes, and the subtasks without successor tasks become exit nodes. If there are multiple entry nodes at the same time, a new input node is added as the predecessor node of multiple entry nodes. For example, in Figure 2(b), the input node is added before the entry nodes 1, 12, and 23. Similarly, if there are multiple exit nodes, add a new exit node. For example, in Figure 2(b), the output node is added after the exit nodes 11, 22, and 33. The calculation and communication delays for both the input and output nodes are zero.

For the known DAG model, the predecessor node set of subtask v_i is $\text{pre}(v_i)$, and the successor node set is $\text{suc}(v_i)$. The input data size d_i of subtask v_i is derived from the output of the preceding task. For a subtask with $|\text{pre}(v_i)|$ preceding nodes, in order to facilitate an easier explanation, we define the data size provided by each preceding node as the average value $\frac{d_i}{|\text{pre}(v_i)|}$, where $|\cdot|$ denotes the cardinality of the set $\cdot$. Importantly, whether we choose the average value as the output of each preceding node does not affect the algorithm itself.

Furthermore, the layer value $h(i)$ of subtask v_i in the subtask set V can be determined by the following formula.

$$h(i) = \begin{cases} 1, \text{pre}(v_i) = \varnothing, \\ 1 + \max\{h(\text{pre}(v_i))\}, \text{pre}(v_i) \neq \varnothing \end{cases} \quad (1)$$

where $\max\{h(\mathrm{pre}(v_i))\}$ means the maximal layer value of the predecessor node set $\mathrm{pre}(v_i)$, and $\varnothing$ represents the empty set

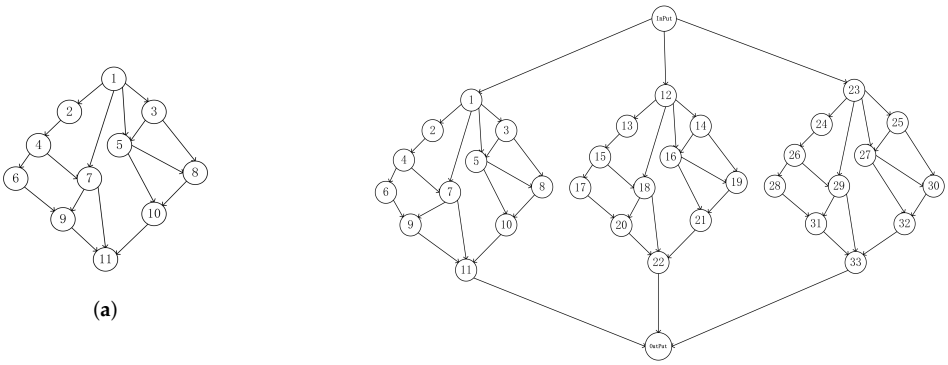


Figure 2. Task model.

3.3. Delay Model

The computing power of UE is ignored compared with the servers in this paper, so all the subtasks are transmitted to it hosting MEC severs first, and then, the MCC server analyzes the gathered data to decide whether the subtasks should be offloaded to other MEC servers for computation, offloaded to the MCC server for computation, or locally computed on the hosting MEC server without any data transmission. Thus, the completion delay of each subtask v_i can be divided into two parts: processing delay p_i and transmission delay t_i.

The processing delay of subtask v_i on the edge server and cloud server are

$$p_i^{\mathrm{E}} = g_i / f_l, \quad 0 \le l \le L-1 \tag{2}$$

and

$$p_i^{\mathrm{C}} = g_i / f_L \tag{3}$$

where the symbols E and C represent the processing that takes place at edge and cloud server, respectively.

The processing delay of the subtask v_i on the server can be expressed as

$$p_i = \alpha p_i^E + \beta p_i^C \tag{4}$$

where α and β are all 0–1 variables, and

$$\alpha = \begin{cases} 1, \text{ task } v_i \text{ is processed on MEC server,} \\ 0, \text{ otherwise} \end{cases} \tag{5}$$

$$\beta = \begin{cases} 1, \text{ task } v_i \text{ is processed on MCC server,} \\ 0, \text{ otherwise} \end{cases} \tag{6}$$

and $\alpha + \beta = 1$.

Since the MCC server starts collecting information from all servers and deciding the offloading strategy only after the tasks are uploaded to their hosting servers, the latency from UEs to their hosting servers does not need to be considered in our model. In the process of data transmission, there are four possible scenarios. To clearly explain these scenarios, we define three binary variables y_{i,s_l}, $y_{i,s_l'}$, and y_{i,s_L} to denote whether the subtask v_i is executed by the MEC server l, l' or the MCC server, respectively. If the predecessor task v_j and the successor tasks v_i are all allocated on the same server, the input data $\frac{d_i}{|\mathrm{pre}(v_i)|}$ originates from within the local server and the transmission delay is zero. If the predecessor

task v_j is processed on the MEC server s_l, and the successor task v_i is executed on another MEC server $s_{l'}$, the transmission delay $t_{i,j,s_l \to s_{l'}}$ is equal to

$$t_{i,j,s_l \to s_{l'}} = y_{j,s_l} y_{i,s_{l'}} \frac{d_i}{|\mathrm{pre}(v_i)| \times r^{\mathrm{E}}} \tag{7}$$

where r^{E} is the transmission rate between BSs in the resource pool, and the arrow indicates the direction of the computing platforms for offloading, and

$$y_{j,s_l} = \begin{cases} 1, & \text{if predecessor task } v_j \text{ is processed on MEC server } s_l, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

$$y_{i,s_{l'}} = \begin{cases} 1, & \text{if successor task } v_i \text{ is processed on other MEC server } s_{l'}, \text{ and } l \neq l', \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

Similarly, it can be inferred that the transmission delay from BS to MCC is

$$t_{i,j,s_l \to s_L} = y_{j,s_l} y_{i,s_L} \frac{d_i}{|\mathrm{pre}(v_i)| \times r^{\mathrm{C}}} \tag{10}$$

and from MCC to BS is

$$t_{i,j,s_L \to s_l} = y_{j,s_L} y_{i,s_l} \frac{d_i}{|\mathrm{pre}(v_i)| \times r^{\mathrm{C}}} \tag{11}$$

where r^{C} is the transmission rate between the BS and MCC server, and y_{j,s_L}, y_{i,s_L} and y_{i,s_l} are all binary functions that have the same characteristics as y_{j,s_l} in Equation (8).

Taking into account the four aforementioned cases, the transmission delay from the predecessor task v_j to the successor task v_i can be described as

$$t_{i,j} = t_{i,j,s_l \to s_L} + t_{i,j,s_L \to s_l} + t_{i,j,s_l \to s_{l'}}, l \neq l', \forall l, l' \in \{1, 2, \ldots, L-1\} \tag{12}$$

Due to the fact that each server can only handle a single task at a time, tasks assigned to the same server must be executed sequentially. If task v_k is the preceding task of task v_i in the server's scheduling sequence, and m_k and p_k represent the start time and execution delay of task v_k, and then the start time m_i of task v_i must satisfy

$$m_i \geq m_k + p_k \tag{13}$$

Additionally, each task v_j in the preceding task set $\mathrm{pre}(v_i)$ of task v_i and data transmission between preceding task v_j and succeeding task v_i must be accomplished. Thus, we have

$$m_i \geq \max_{j \in \mathrm{pre}(v_i)} \{m_j + p_j + t_{i,j}\} \tag{14}$$

As a result, the start time m_i of task v_i is

$$m_i = \max \left\{ m_k + p_k, \max_{j \in \mathrm{pre}(v_i)} \{m_j + p_j + t_{i,j}\} \right\} \tag{15}$$

Finally, the completion time of all tasks is

$$t_{\mathrm{total}} = \max_{1 \leq i \leq I} \{m_i + p_i\}. \tag{16}$$

3.4. Server Cost Model

The cost of executing a task on a processor is closely related to the processor's own capability and execution time. We use the exponential pricing model [31] to define the cost in which a server with more processing power is associated with a higher cost, that is

$$O_i^{\mathrm{E}} = \sigma \times p_i^{\mathrm{E}} \times \delta \times \exp\left(\frac{f_{l'}}{\min\limits_{1 \le l \le L-1} f_l}\right) \tag{17}$$

and

$$O_i^{\mathrm{C}} = \sigma \times p_i^{\mathrm{C}} \times \delta \times \exp\left(\frac{f_L}{\min\limits_{1 \le l \le L} f_l}\right) \tag{18}$$

where O_i^{E} and O_i^{C} denote the cost of task v_i on MEC server and MCC server, respectively, and σ is a random variable used to generate different pricing models, $f_{l'}$ is the frequency of the MEC server that processes task v_i, $\min\limits_{1 \le l \le L} f_l$ is the minimum of server frequency, and δ is the base price charged to $\min\limits_{1 \le l \le L} f_l$.

Based upon this, the execution cost cost_i of task v_i is calculated as follows.

$$O_i = \alpha O_i^{\mathrm{E}} + \beta O_i^{\mathrm{C}} \tag{19}$$

where α and β are the Boolean indicators defined in Equations (5) and (6), respectively.

The total server cost is

$$O_{\mathrm{total}} = \sum_{i=1}^{I} O_i \tag{20}$$

3.5. Problem Formulation

This paper aims to investigate the resource allocation optimization problem in a mobile edge computing (MEC) network composed of multiple users, multiple edge servers, and one cloud server. The optimization objective is to minimize the task completion time and reduce server costs. User delay is represented by the total delay of all user tasks in the network, while the expression of the total processor cost is based on an exponential pricing model. Based on the above equations, the problem can be formulated as follows.

$$\min_{\alpha,\beta,\left\{y_{i,s_l'},1 \le i \le I,1 \le l \le L\right\}} \quad \lambda \times t_{\mathrm{total}} + (1-\lambda) \times \varphi \times O_{\mathrm{total}} \tag{21}$$

$$\text{s.t.} \quad C1: \sum_{l=1}^{L} \mathrm{y}_{i,s_l} = 1$$

$$C2: \sum_{i=1}^{I}\sum_{l=1}^{L} \mathrm{y}_{i,s_l} = I$$

$$C3: \alpha + \beta = 1$$

$$C4: 0 \le \lambda \le 1$$

In the set of constraints, C1 indicates that a task can only be assigned to one server for processing. C2 states that each task must be assigned a server for processing. C3 implies that tasks can either be processed in MEC or MCC, but not both. C4 makes sure that the weights of the optimization objectives are all positive values. Where λ is a weighting factor to balance the total delay and total cost in the optimization goal, φ is the normalization factor, which makes the total cost and total delay under one scale.

In order to minimize the optimization problem (21) mentioned above, we seek the optimal values for the binary offloading decision variables α, β, and $y_{i,s_l'}$. Our objective is to reduce the total user delay and processor cost while adhering to the given constraints. However, as this is a nonlinear binary programming problem that falls under the NP-hard category, finding the optimal solution is not feasible. Therefore, we propose a heuristic algorithm called the improved GSA (IGSA) to approximate the optimal solution to the original problem as closely as possible. By applying the IGSA algorithm, we can obtain an approximate optimal solution that satisfies our optimization objectives.

4. Proposed Algorithm

As our algorithm is a hybrid of genetic algorithm (GA) and GSA, we first provide a brief description about both of these algorithms as follows.

4.1. Overview of GA

The genetic algorithm mimics the mechanisms of selection, crossover, and mutation observed in biological genetics and evolutionary processes to perform an adaptive search process for solving optimization problems. It starts by initializing the algorithm with any initial population (a set of feasible solutions). Then, an fitness function is designed to calculate the fitness value of each individual (a single solution) in the population. Next, individuals with higher fitness values are selected, and genetic crossover is performed among them to form a new population. This process is repeated iteratively to evolve and obtain one or several optimal solutions.

The genetic algorithm utilizes crossover operations to generate new feasible solutions, thereby maintaining population diversity and improving population quality. The proposed IGSA in this paper integrates the crossover and mutation operations of SA into the GSA algorithm. By performing these operations, a portion of the encoded positions of feasible solutions are replaced by a portion of the best solution in the current population with a certain probability q during the stage of generating a new population in GSA. This method further improves the convergence speed and optimization range of the algorithm.

4.2. Overview of GSA

Gravitational search algorithm (GSA) is a random heuristic search algorithm. Initially, the algorithm generates a set of particles (a set of feasible solutions) through random position encoding and forms an initial population by assembling these particles. Then, based on the law of universal gravitation, the algorithm calculates the force between particles using their masses and distances. Next, each particle calculates its own force based on the magnitude of total force and uses it to update its velocity and new position. Finally, the algorithm iterates continuously to obtain the final solution.

During the process of calculating the total force, IGSA improved the GSA and designed a convergence factor that only calculates the force between the top k particles with the optimal fitness values and maximum mass, filtering out some miscellaneous particles and highlighting the influence of superior individuals. This method further accelerates the convergence of the algorithm, achieving a good balance between exploration and exploitation.

4.3. Generation of Population

The population is composed of a group of particles. Each particle corresponds to a solution of the optimization problem. Therefore, the population is a set of feasible resource allocation methods. Initially, we generate a population $X = \{X_n, n = 1, 2, \ldots, N\}$, where the particle is denoted by X_n. Actually, a particle X_n means a complete solution of the problem. For the network with I subtasks and L severs, it can be expressed as follows.

$$X_n = \left\{ x_n^1, x_n^2, \ldots, x_n^i, \ldots, x_n^I \right\}, 1 \le i \le I, 1 \le n \le N \tag{22}$$

where x_n^i represents the sever assigned to the task v_i in the solution X_n.

Next, we proceed to encode the positions of the particles. Note that x_n^i implies the sever number, so we first define

$$0 < x_n^i \le 1. \tag{23}$$

and then have

$$s_l = \left\lceil x_n^i \times L \right\rceil \tag{24}$$

where the symbol $\lceil \rceil$ signifies the ceiling function.

The diagram in Figure 3 illustrates an example of particle encoding in a system with 3 base stations and 3 users, where each user has 4 subtasks.

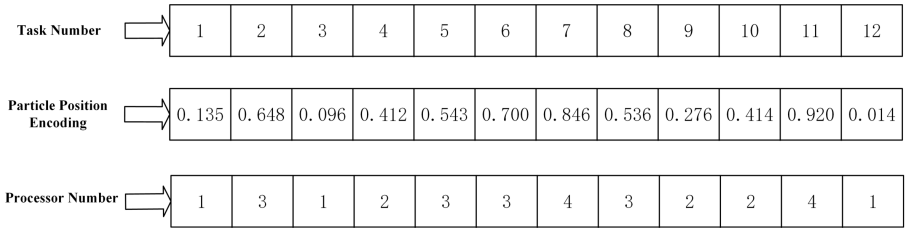


Figure 3. Particle position encoding.

Each particle has a fitness value which indicates the quality of the solution. For a given particle (allocation strategy), we can always calculate its total delay and total cost using Equations (16) and (20), respectively, so we can compute

$$q_n = \frac{1}{\lambda t_{\text{total}} + (1-\lambda) \times \varphi \times O_{\text{total}}} \tag{25}$$

where q_n denotes the fitness value of the nth particle, and λ and φ are defined in Equation (21).

The fitness value is inversely proportional to the total latency and total cost. Therefore, a particle with low latency and low cost will have a high fitness value. The higher the fitness value, the higher the likelihood of becoming the optimal solution.

The fitness can be obtained based on the particle's position encoding. Firstly, calculate the hierarchical values of subtasks based on Equation (1), and sort the subtasks in ascending order according to their hierarchical values. If there are subtasks with the same hierarchical value, sort them in descending order based on their corresponding position codes. This process yields the topological order vector, denoted by a. Next, in accordance with the order of subtasks in a, calculate the execution time p_i and start time m_i of each task using Formulas (4) and (15). During this process, the server number s_l corresponding to the subtasks are obtained from Equation (24). Then, evaluate the total user delay, total server cost, and fitness function value by applying Formulas (16), (20), and (25), respectively. A detailed procedure for computing the fitness is described in Algorithm 1.

Algorithm 1 Fitness-Calculation

Input: Particle position X_n
Output: Fitness q_n

Initialize the processor end delay matrix to 0 matrix, processor_time[l] = 0, $1 \leq l \leq L$
for each task $v_i \in V$ in the topological order of subtasks a **do**
  Compute the server number s_l corresponding to the task v_i according to Equation (24)
  Set the forward task end time to 0, that is, $m_i = 0$
  if the preceding task v_i exists **then**
    for traverse all tasks v_j in the forward node set of task v_i, $v_j \in \text{prev}(v_i)$ **do**
      if $m_i < m_j + p_j + t_{i,j}$ **then**
        $m_i = m_j + p_j + t_{i,j}$
      end if
    end for
  end if
  Compute processing delay p_i using Equation (4)
  The actual start time of task v_i, $v_i = \max\{\text{Processor\_time}[s_l], \text{m}_i\}$
  Update the processor end latency matrix, Processor_time[s_l] = $m_i + p_i$
end for
Calculate the total latency of user end tasks using Equation (16)
Calculate the total cost of the processor using Equation (20)
Calculate the fitness function value using Equation (25)

Typically, the computed fitness values vary in a wide range. Hence, we employ the method of max–min normalization to normalize the fitness to a range of [0, 1] to evaluate the mass of each particle. The mass of the nth particle is given by

$$Q_n(t) = \frac{q_n(t) - \min\limits_{n'=1,2,\ldots,N} q_{n'}(t)}{\max\limits_{n'=1,2,\ldots,N} q_{n'}(t) - \min\limits_{n'=1,2,\ldots,N} q_{n'}(t)} \tag{26}$$

where $q_n(t)$ is the fitness value of the nth particle, $\max\limits_{n'=1,2,\ldots,N} q_{n'}(t)$ is the highest particle fitness function value in the t-th iteration, and $\min\limits_{n'=1,2,\ldots,N} q_{n'}(t)$ is the lowest particle fitness function value in the t-th iteration.

4.4. Force Computation

Let $Q_n(t)$ and $G(t)$ be the mass of n-th particle and the gravitational constant, respectively, in the t-th iteration. We can define the force acting on the n-th particle by n'-th particle for the t-th iteration as follows.

$$F^i_{n,n'}(t) = G(t) \times \frac{Q_n(t) \times Q_{n'}(t)}{R_{n,n'}(t) + \delta} \times (x^i_{n'}(t) - x^i_n(t)) \tag{27}$$

where $Q_n(t)$, $Q_{n'}(t)$ are the masses of particles n and n', respectively. $R_{n,n'}(t)$ is the Euclidian distance between two particles n and n', and $x^i_n(t)$ and $x^i_{n'}(t)$ are the encoded positions of the particle n and particle n' in the searching space with the dimension i, respectively; $G(t)$ is the gravitational factor, and δ is a small constant.

The gravitational constant G_0 is initialized in the beginning and reduces as the algorithm proceeds. In order to improve the search accuracy, we define $G(t)$ as a function of initial value G_0 and iteration number t.

$$G(t) = G_0 \times (\frac{t}{T})^{\gamma} \tag{28}$$

where T is the maximal iteration time and γ is a small constant.

To avoid trapping in a local optimum, kbest is introduced in the IGSA. The force acting on the particle is defined as follows:

$$F^i_n(t) = \sum_{n' \in \text{kbest}(t), n' \neq n} \text{rand}_n \times F^i_{n,n'}(t) \tag{29}$$

where rand_n is a random number between [0, 1] and Kbest is the set of k best particles with the biggest masses.

For each iteration, we compute the percent λ of particles that apply force to the others in the last iteration, and apply it to compute the value k in the current iteration. k is defined as

$$k = \left\langle \frac{N}{100} \times (\lambda + (1 - (\frac{t}{T}) \times (1 - \lambda))) \right\rangle \tag{30}$$

where the sign $\langle \cdot \rangle$ means to round $\cdot$.

From the above definition, it is obvious that the elements in set kbest(t) is a linear decreasing function of time. Only the particles in set Kbest attracts other particles, which filters out some idle particles and highlights the influence proportion of the better individual. Not only that, but this method also speeds up the convergence.

4.5. Position Update

The acceleration of the n particle at iteration t in the i dimension space can be defined as:

$$a^i_n(t) = \frac{F^i_n(t)}{Q_n(t)} \tag{31}$$

The velocity and position of particles are calculated as follows:

$$v_n^i(t+1) = \text{rand}_n \times v_n^i(t) + a_n^i(t) \tag{32}$$

$$x_n^i(t+1) = x_n^i(t) + v_n^i(t+1) \tag{33}$$

During the position update procedure, we introduce the idea of cross mutation in the genetic algorithm. In each iteration, for the particle whose position is updated, if the generated random number rank_r ($\text{rand}_r \in [0,1]$) is less than the predefined crossover probability p_c, the original position information of the agent is replaced with part of the position information in the global optimal solution. The change of the particle force before and after replacement is compared. Only when the force becomes larger, the replacement retains. The crossover procedure is shown in Algorithm 2.

Algorithm 2 IGSA algorithm

Initialize particle position
while (t < maxt) **do**
  calculate fitness s_i using Algorithm 1 and Formula (25)
  calculate $G(t)$ using Formula (28)
  calculate Q_n using Formula (26)
  for $i = 1 : N$ **do**
    Calculate the resultant force on particles using Formula (29)
    Calculate particle acceleration and velocity using Formulas (31) and (32)
    Calculate X_n using Formula (33)
  end for
  Calculate the optimal solution X_{best} for the current population
  for $i = 1 : N$ **do**
    $U_i = X_{\text{best}}$
    if $\text{rand}_r < p_c$ **then**
      Generate random integer N_l, $1 < N_l < N_d - l'''$, $[u_i^{N_l}, \ldots, u_i^{N_l+l}] = [x_{\text{best}}^{N_l}, \ldots, x_{\text{best}}^{N_l+l}]$
      if $\text{fitness}(U_i) > \text{fitness}(X_{\text{best}})$ **then**
        $X_n = U_i$
      else
        $X_n = X_n$
      end if
    end if
  end for
  t = t +1
end while

5. Simulations and Results

In this section, we evaluate the performance of the proposed IGSA. Firstly, the convergence of IGSA is simulated. Then, the proposed algorithm is compared with the HGSA [27] and GSAL [28] algorithms. The distinctions between the three algorithms are as follows:

(a) IGSA is a strategy designed to solve the joint task offloading and resource allocation problem in the context of cloud-edge collaborative dual-layer network structures. It adopts a series of optimization strategies, which enable it to outperform other algorithms in terms of performance. Firstly, the IGSA algorithm introduces the concept of a convergence factor during the computation of the resulting force. By incorporating a continuous iterative updating process, it aims to seek the optimal solution. This not only enhances the convergence speed of the algorithm but also allows for the discovery of improved solutions after multiple iterations. Secondly, the IGSA algorithm incorporates genetic algorithm crossover operations after each iteration, enhancing the algorithm's global search capability, and enabling it to better find the optimal solution.

(b) In HGSA, in [27], a meta-heuristic workflow scheduling algorithm is designed to address scheduling problems in cloud computing. The HGSA algorithm utilizes the output of the heterogeneous earliest finish time (HEFT) algorithm to seed the initial population, retains the best particle to enhance convergence speed, and optimizes the quality of the population through the threshold quality and update mechanisms. However, compared to the IGSA algorithm, the hybrid genetic/simulated annealing (HGSA) algorithm primarily adopts predefined heuristic strategies and lacks flexibility to adjust decision strategies according to real-time situations, resulting in poor adaptability.

(c) In the GSAL algorithm mentioned in [28], a recursive algorithm is used to generate effective task execution sequences to enhance the priority relationships between tasks. This approach allows for effective scheduling based on task correlations and dependencies. However, compared to the IGSA algorithm, the GSAL algorithm has a simpler search strategy, which may result in it getting stuck in local optimal solutions during the search process, leading to less desirable results.

In addition, since the objective function and fitness value are inversely related, and the system performance is directly proportional to fitness, we will analyze the system performance using the fitness value in further discussions.

This article conducts a simulation analysis on the proposed IGSA algorithm. Assuming that the total number of users in the system is R = 5 and the total number of processors is L = 4, there are three MEC servers and one MCC server in the system. The computing power of the MCC server is $f_L = 64$ GHz, while the computing power of MEC server is 16, 15, and 14 GHz, respectively. The required CPU cycle d_i of the subtask is taken as a random value at [0.2, 0.3] GHz, and the corresponding task size g_i is taken as a random value at [500, 1500] KB. The base price of the processor cost $\delta = 2$. Random variables $\sigma = 1$. Gravity factor $\lambda = 0.5$, normalization factor $\varphi = 0.0005$. In the IGSA algorithm, let the gravitational constant G_0, $\gamma = 0.3$, maximum iteration $\varsigma = 1000$, number of particles $N = 20$, crossover rate $p_c = 0.7$, and minimum gravitational coefficient 0.1. The simulation results are taken as the average of 10 experiments. The simulation parameters are summarized in Table 1.

Table 1. Simulation parameters.

Name	Value
Number of processors	$L = 4$
Number of users	$R = 5$
The computing power of MCC server	$f_L = 64$ GHz
The computing power of MEC servers	16, 15, 14 GHz
The subtask required CPU cycle	$d_i = [0.2, 0.3]$ GHz
The subtask data size	$g_i = [500, 1500]$ KB
Processor base price	$\delta = 2$
Random variable	$\sigma = 1$
The weighting factor	$\lambda = 0.5$
The normalization factor	$\varphi = 0.0005$

Next, we first simulate the convergence of the proposed IGSA algorithm in the different task numbers using Figure 4 to illustrate that the algorithm can achieve fast convergence with a smaller number of iterations. Secondly, in the simulation presented in Figure 5, we compared the fitness values, delays, and costs of three algorithms under different task numbers to illustrate the advantages of the proposed algorithm in various performance metrics. Then, in Figure 6, we further simulated the convergence of different algorithms in terms of fitness values under the different numbers of tasks. Finally, in Figures 7 and 8, we compared the fitness values of the algorithms under different numbers of users and processors to demonstrate the performance of the proposed algorithm in this paper. These experimental results further validate the effectiveness and feasibility of the algorithm proposed in this study.

The following paragraph provides a detailed analysis of the simulation results.

According to Figure 4, in an MEC system with three MEC servers and one MCC server, for different task loads (8, 12, and 17), we observed two characteristics of the proposed IGSA algorithm in terms of fitness value. Firstly, as the number of iterations increases, this algorithm always converges rapidly to the optimal fitness value. This is because the gravitational effect attracts particles towards the optimal solution, causing them to cluster around it. As the number of iterations increases, the distance between particles decreases, bringing them closer to the optimal solution and achieving convergence towards the best solution. Secondly, when the task quantity decreases, especially for I = 8, the algorithm converges to a higher optimal fitness value (1.2). This is because, as the task quantity decreases, the search space of the problem becomes smaller and the solution space becomes more concentrated, making it easier to find the global optimal solution. Therefore, as the number of iterations increases, the algorithm is more likely to find a higher optimal fitness value.

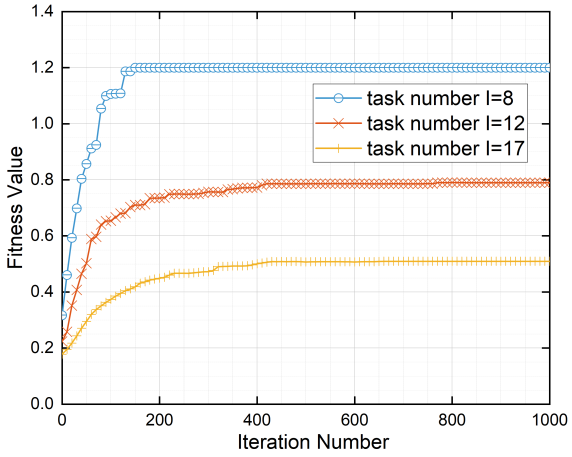
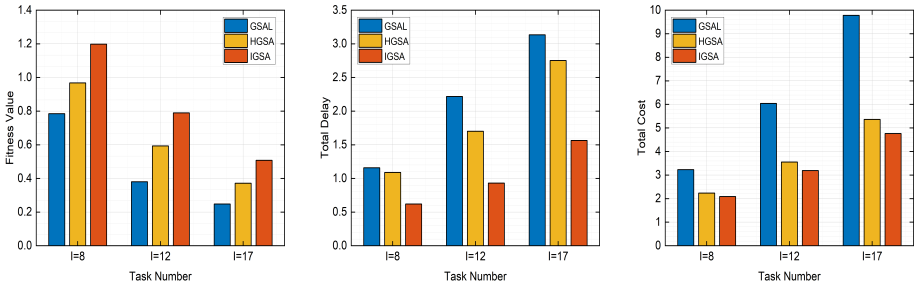


Figure 4. Algorithm convergence at different numbers of tasks.

In Figure 5, we analyzed the relationship between the task quantity and fitness value, the total user delay, and processor cost. We considered the task quantities of 8, 12, and 17. It was observed that, as the task quantity increased, the fitness values of all algorithms decreased, while the total delay and cost increased. In addition, in all scenarios, the algorithm proposed in this paper consistently outperforms other baseline algorithms, achieving higher fitness values and lower total delay and cost. This can be attributed to the IGSA algorithm's stronger global search capability as the task quantity increases. The convergence factor guides particles towards the global optimal solution, leading to rapid convergence. Additionally, the crossover operation in the genetic algorithm enhances the diversity of the search space, preventing the algorithm from getting stuck in local optima. This enables the algorithm to find solutions with higher fitness values, ultimately minimizing the total delay and cost.

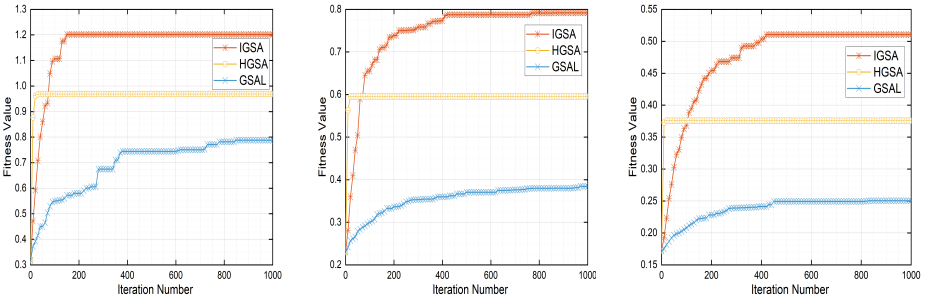
In Figure 6, we compared the IGSA algorithm proposed in this paper with the HGSA algorithm from the reference [27] and the GSAL algorithm from the reference [28]. We simulated their overall performance under the different values of N (8, 12, and 17). Analyzing the final optimization results, we found that the IGSA algorithm proposed in this paper exhibits a better optimization performance compared to the HGSA and GSAL algorithms. The HGSA algorithm proposed in [27] shows an average performance in terms of optimization results, despite its attempt to improve the convergence speed by replacing particles below a threshold with the current iteration's best particle. Similarly, the GSAL algorithm proposed in [28] demonstrates a poorer performance in convergence accuracy and optimization results, despite its use of the acceleration and force of the gravitational law to locate the next particle to be executed. In contrast, the IGSA algorithm proposed in this paper incorporates a convergence factor in the resulting force calculation and uti-

lizes the genetic algorithm's crossover operation in each iteration, leading to improved optimization results.



(**a**) Fitness comparison (**b**) Delay comparison (**c**) Cost comparison

Figure 5. Fitness, delay, and cost performance for different algorithms when $\lambda = 0.5$.



(**a**) I = 8 (**b**) I = 12 (**c**) I = 17

Figure 6. Performance comparison of different algorithms.

The following paragraph presents additional experimental results to demonstrate the effectiveness of the proposed IGSA algorithm when varying the number of processors and users. It also compares its performance with other leading algorithms in the field.

The comparison results in Figure 7 demonstrate the clear advantage of our algorithm over others. As the number of users increases from 5 to 7, our algorithm only experiences a 31% decrease in fitness value, while the other two algorithms show decreases of 43% and 44%, respectively. Moreover, as the number of users increases, the fitness value gradually decreases. This is because the increase in the number of users leads to an increase in total delay and cost. As the fitness value is a weighted sum of delay and cost, it naturally decreases. However, despite the increase in the number of users from 5 to 13, our algorithm still demonstrates a significant advantage.

By observing Figure 8, it can be noticed that, as the number of processors increases, the three algorithms show different performances in terms of fitness value. In the graph, the IGSA algorithm consistently achieves the optimal fitness value. This is because the algorithm introduces a crossover operation, facilitating effective gene pairing and selecting individuals with higher fitness as parents for the next generation or directly including them in the next generation. This mechanism leads to a superior genetic composition, resulting in an improvement in the overall fitness value of the system. Additionally, we observe that the fitness value decreases most rapidly when the number of processors increases from 3 to 6. However, when the number of processors further increases to 15, the fitness value stabilizes. This can be explained by the fact that, with fewer processors, there is more room for optimization, resulting in a significant decrease in fitness value. However, as the number of processors continues to increase, the system reaches a point of optimal utilization. Further increasing the number of processors may not lead to a significant

performance improvement since the system is already operating at a relatively stable level of resource utilization.

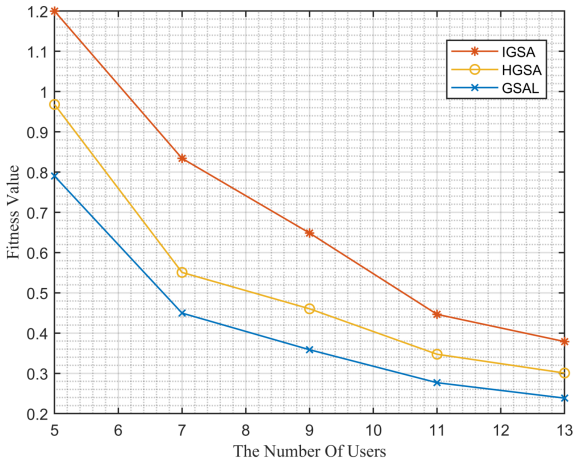


Figure 7. Performance comparison of different algorithms as the number of users changes.

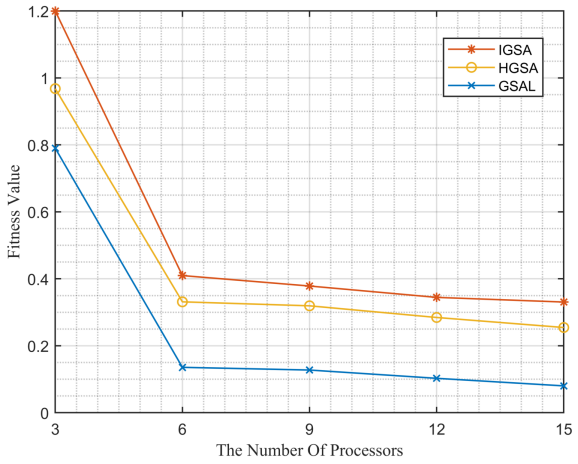


Figure 8. Performance comparison of different algorithms as the number of processors changes.

From the above analysis, we can see that our algorithm outperforms existing HGSA and GSAL algorithms in terms of fitness value, total delay, and total cost. A key reason for this is that our proposed IGSA algorithm introduces the concept of a convergence factor in the calculation of a resulting force during the iteration process to search for the optimal solution. This allows for continuous iterative updates and a more efficient convergence towards the optimal solution. The introduction of this convergence factor not only accelerates the convergence speed of the algorithm but also helps find better solutions after multiple iterations. Furthermore, after each iteration, we also apply the crossover operation of the genetic algorithm to enhance the global search capability of the algorithm. Through this approach, we can better explore the search space and find better solutions. Therefore, our IGSA algorithm can provide both a fast convergence speed and a high global search capability, enabling it to find high-quality solutions.

6. Conclusions

In this paper, we consider the problem of a computing offload in a three-tier mobile-edge computing network consisting of a multi-user, multi-edge server, and a cloud server. By calculating the offloading strategy, we will optimize the user task execution latency and server computing costs. In order to achieve better uninstallation benefits for all users, multiple subtasks need to be handled for multiple users. We designed a suitable gravity search algorithm (IGSA) to address this issue, modified the strategy of gravity search

algorithm in calculating resultant force and each iteration, and proposed an improved gravity search algorithm (IGSA) to solve this problem. We designed a comprehensive simulation experiment to verify the performance of IGSA and the feasibility of the model. The experimental results show that this strategy is significantly superior to the baseline method in terms of latency, cost, and other aspects.

In future work, we will continue to follow the work of this article and focus on some issues worthy of research.

i Assuming that offloading follows a centralized decision-making approach in this paper, in the follow-up research, we will explore the dynamic offloading decision-making.
ii In this paper, we focused on the overall delay for users and the total cost for processors. In future research, we will discuss additional aspects such as security, privacy, and user experience.
iii We plan to investigate more complex and diverse models and tasks in future work, potentially integrating them within the field of deep learning, which holds significant interest and value.

Author Contributions: Conceptualization: L.X. and Y.L.; methodology: L.X., Y.L., X.X. and Y.M.; software: L.X. and B.F.; formal analysis: W.F.; investigation: L.X., Y.L., X.X., B.F., Y.M. and W.F.; resources: L.X., Y.L., X.X. and B.F.; data curation: Y.L., X.X., and B.F.; writing—original draft preparation: L.X., Y.L. and B.F.; writing—review and editing: Y.M. and W.F.; visualization: L.X., Y.M. and W.F.; supervision: Y.M. and W.F.; project administration: Y.M. and W.F.; funding acquisition: Y.M. and W.F. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partly supported by the Natural Science Foundation of China under Grant 62101169, Grant 62301204, and Grant 62371174.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: Author Yunpeng Liu was employed by Zhejiang Haikang Zhilian Technology Co., Ltd. and Yiguo Mei was employed by Huaxin Consulting Co., Ltd. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Nguyen, D.C.; Ding, M.; Pathirana, P.N.; Seneviratne, A.; Li, J.; Niyato, D.; Dobre, O.; Poor, H.V. 6G Internet of Things: A comprehensive survey. *IEEE Internet Things J.* **2022**, *9*, 359–383. [CrossRef]
2. Liang, B.; Gregory, M.A.; Li, S. Multi-access Edge Computing fundamentals, services, enablers and challenges: A complete survey. *J. Netw. Comput. Appl.* **2022**, *199*, 103308. [CrossRef]
3. Dong, S.; Xia, Y.; Kamruzzaman, J. Quantum Particle Swarm Optimization for Task Offloading in Mobile Edge Computing. *IEEE Trans. Ind. Inform.* **2023**, *19*, 9113–9122. [CrossRef]
4. Mei, J.; Tong, Z.; Li, K.; Zhang, L.; Li, K. Energy-Efficient Heuristic Computation Offloading With Delay Constraints in Mobile Edge Computing. *IEEE Trans. Serv. Comput.* **2023**, *16*, 4404–4417. [CrossRef]
5. Wang, P.; Li, K.; Xiao, B.; Li, K. Multiobjective Optimization for Joint Task Offloading, Power Assignment, and Resource Allocation in Mobile Edge Computing. *IEEE Internet Things J.* **2022**, *9*, 11737–11748. [CrossRef]
6. Hu, Z.; Niu, J.; Ren, T.; Dai, B.; Li, Q.; Xu, M.; Das, S.K. An Efficient Online Computation Offloading Approach for Large-Scale Mobile Edge Computing via Deep Reinforcement Learning. *IEEE Trans. Serv. Comput.* **2022**, *15*, 669–683. [CrossRef]
7. Liu, L.; Yuan, X.; Chen, D.; Zhang, N.; Sun, H.; Taherkordi, A. Multi-User Dynamic Computation Offloading and Resource Allocation in 5G MEC Heterogeneous Networks With Static and Dynamic Subchannels. *IEEE Trans. Veh. Technol.* **2023**, *72*, 14924–14938. [CrossRef]
8. Wang, F.; Cai, S.; Lau, V.K.N. Sequential Offloading for Distributed DNN Computation in Multiuser MEC Systems. *IEEE Internet Things J.* **2023**, *10*, 18315–18329. [CrossRef]
9. Li, X.; Chen, T.; Yuan, D.; Xu, J.; Liu, X. A Novel Graph-Based Computation Offloading Strategy for Workflow Applications in Mobile Edge Computing. *IEEE Trans. Serv. Comput.* **2023**, *16*, 845–857. [CrossRef]
10. Li, K.; Wang, X.; He, Q.; Ni, Q.; Yang, M.; Dustdar, S. Computation Offloading for Tasks With Bound Constraints in Multiaccess Edge Computing. *IEEE Internet Things J.* **2023**, *10*, 15526–15536. [CrossRef]
11. Fang, T.; Yuan, F.; Ao, L.; Chen, J. Joint Task Offloading, D2D Pairing, and Resource Allocation in Device-Enhanced MEC: A Potential Game Approach. *IEEE Internet Things J.* **2022**, *9*, 3226–3237. [CrossRef]

12. Wang, X.; Han, Y.; Shi, H.; Qian, Z. JOAGT: Latency-Oriented Joint Optimization of Computation Offloading and Resource Allocation in D2D-Assisted MEC System. *IEEE Wirel. Commun. Lett.* **2022**, *11*, 1780–1784. [CrossRef]
13. Chen, G.; Chen, Y.; Mai, Z.; Hao, C.; Yang, M.; Du, L. Incentive-Based Distributed Resource Allocation for Task Offloading and Collaborative Computing in MEC-Enabled Networks. *IEEE Internet Things J.* **2023**, *10*, 9077–9091. [CrossRef]
14. Pan, L.; Liu, X.; Jia, Z.; Xu, J.; Li, X. A Multi-Objective Clustering Evolutionary Algorithm for Multi-Workflow Computation Offloading in Mobile Edge Computing. *IEEE Trans. Cloud Comput.* **2023**, *11*, 1334–1351. [CrossRef]
15. Laboni, N.M.; Safa, S.J.; Sharmin, S.; Razzaque, M.A.; Rahman, M.M.; Hassan, M.M. A Hyper Heuristic Algorithm for Efficient Resource Allocation in 5G Mobile Edge Clouds. *IEEE Trans. Mob. Comput.* **2024**, *23*, 29–41. [CrossRef]
16. Vieira, R.F.; Souza, D.D.S.; Silva, M.S.D.; Cardoso, D.L. A Heuristic for Load Distribution on Data Center Hierarchy: A MEC Approach. *IEEE Access* **2022**, *10*, 69462–69471. [CrossRef]
17. Zheng, K.; Jiang, G.; Liu, X.; Chi, K.; Yao, X.; Liu, J. DRL-Based Offloading for Computation Delay Minimization in Wireless-Powered Multi-Access Edge Computing. *IEEE Trans. Commun.* **2023**, *71*, 1755–1770. [CrossRef]
18. Sun, M.; Xu, X.; Han, S.; Zheng, H.; Tao, X.; Zhang, P. Secure Computation Offloading for Device-Collaborative MEC Networks: A DRL-Based Approach. *IEEE Trans. Veh. Technol.* **2023**, *72*, 4887–4903. [CrossRef]
19. Jiao, X.; Ou, H.; Chen, S.; Guo, S.; Qu, Y.; Xiang, C.; Shang, J. Deep Reinforcement Learning for Time-Energy Tradeoff Online Offloading in MEC-Enabled Industrial Internet of Things. *IEEE Trans. Netw. Sci. Eng.* **2023**, *10*, 3465–3479. [CrossRef]
20. Zhang, H.; Yang, Y.; Shang, B.; Zhang, P. Joint resource allocation and multi-part collaborative task offloading in MEC systems. *IEEE Trans. Veh. Technol.* **2022**, *71*, 8877–8890. [CrossRef]
21. Liu, Q.; Xia, T.; Cheng, L.; van Eijk, M.; Ozcelebi, T.; Mao, Y. Deep Reinforcement Learning for Load-Balancing Aware Network Control in IoT Edge Systems. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 1491–1502. [CrossRef]
22. Li, J.; Shang, Y.; Qin, M.; Yang, Q.; Cheng, N.; Gao, W.; Kwak, K.S. Multiobjective oriented task scheduling in heterogeneous mobile edge computing networks. *IEEE Trans. Veh. Technol.* **2022**, *71*, 8955–8966. [CrossRef]
23. Shang, C.; Sun, Y.; Luo, H.; Guizani, M. Computation Offloading and Resource Allocation in NOMA–MEC: A Deep Reinforcement Learning Approach. *IEEE Internet Things J.* **2023**, *10*, 15464–15476. [CrossRef]
24. Li, Y.; Yang, C.; Deng, M.; Tang, X.; Li, W. A dynamic resource optimization scheme for MEC task offloading based on policy gradient. In Proceedings of the 2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC), Chongqing, China, 4–6 March 2022; Volume 6, pp. 342–345.
25. Zhang, L.; Liang, Y.C. Deep Reinforcement Learning for Multi-Agent Power Control in Heterogeneous Networks. *IEEE Trans. Wirel. Commun.* **2021**, *20*, 2551–2564. [CrossRef]
26. Ning, Z.; Dong, P.; Kong, X.; Xia, F. A cooperative partial computation offloading scheme for mobile edge computing enabled Internet of Things. *IEEE Internet Things J.* **2018**, *6*, 4804–4814. [CrossRef]
27. Choudhary, A.; Gupta, I.; Singh, V.; Jana, P.K. A GSA based hybrid algorithm for bi-objective workflow scheduling in cloud computing. *Future Gener. Comput. Syst.* **2018**, *83*, 14–26. [CrossRef]
28. Biswas, T.; Kuila, P.; Ray, A.K.; Sarkar, M. Gravitational search algorithm based novel workflow scheduling for heterogeneous computing systems. *Simul. Model. Pract. Theory* **2019**, *96*, 101932. [CrossRef]
29. Abu-Taleb, N.A.; Abdulrazzak, F.H.; Zahary, A.T.; Al-Mqdashi, A.M. Offloading decision making in mobile edge computing: A survey. In Proceedings of the 2022 2nd International Conference on Emerging Smart Technologies and Applications (eSmarTA), Ibb, Yemen, 25–26 October 2022; pp. 1–8.
30. Sun, F.; Cao, J.; Lu, Z. HEFT-dynamic scheduling algorithm in workflow scheduling. In Proceedings of the 2022 34th Chinese Control and Decision Conference (CCDC), Hefei, China, 15–17 August 2022; pp. 4885–4890.
31. Su, S.; Li, J.; Huang, Q.; Huang, X.; Shuang, K.; Wang, J. Cost-efficient task scheduling for executing large programs in the cloud. *Parallel Comput.* **2013**, *39*, 177–188. [CrossRef]