processes

MDPI

Article

M-E-AWA: A Novel Task Scheduling Approach Based on Weight Vector Adaptive Updating for Fog Computing

Zhiming Dai [1,2], Weichao Ding [1,*], Qi Min [1], Chunhua Gu [1,*], Baohua Yao [3] and Xiaohan Shen [1]

1 School of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China; daizm@gench.edu.cn (Z.D.); minqi@mail.ecust.edu.cn (Q.M.); 15615753615@163.com (X.S.)
2 School of Information Technology, Shanghai Jian Qiao University, Shanghai 201306, China
3 Shanghai Institute of Civil Defense Science, Shanghai 200020, China; 13764474005@139.com
* Correspondence: weich@ecust.edu.cn (W.D.); chgu@ecust.edu.cn (C.G.)

Abstract: Task offloading and real-time scheduling are hot topics in fog computing. This paper aims to address the challenges of complex modeling and solving multi-objective task scheduling in fog computing environments caused by widely distributed resources and strong load uncertainties. Firstly, a task unloading model based on dynamic priority adjustment is proposed. Secondly, a multi-objective optimization model is constructed for task scheduling based on the task unloading model, which optimizes time delay and energy consumption. The experimental results show that M-E-AWA (MOEA/D with adaptive weight adjustment based on external archives) can effectively handle multi-objective optimization problems with complex Pareto fronts and reduce the response time and energy consumption costs of task scheduling.

Keywords: fog computing; task scheduling; multi-objective evolutionary algorithm; MOEA/D

check for updates

Citation: Dai, Z.; Ding, W.; Min, Q.; Gu, C.; Yao, B.; Shen, X. M-E-AWA: A Novel Task Scheduling Approach Based on Weight Vector Adaptive Updating for Fog Computing. *Processes* **2023**, *11*, 1053. https://doi.org/10.3390/pr11041053

Academic Editors: Mengchu Zhou and Jie Zhang

Received: 10 January 2023
Revised: 28 March 2023
Accepted: 29 March 2023
Published: 31 March 2023

1. Introduction

As more and more devices are connected to the network in the IoT environment, a vast amount of data must be processed fast and on a large scale. In addition, 5G technology may provide the whole IoT system with increased transmission speed, decreased power usage, and decreased latency. The typical cloud-based centralized service architecture inhibits the correct operation of high-demand applications and services during data processing due to long transmission distances and unpredictable networks [1]. Fog computing is a distributed computing architecture that can process, store, and intelligently control data. Having high-density computing and storage devices deployed in the fog layer close to the user greatly reduces the latency of traditional cloud computing and also gives support for user mobility, which alleviates the huge burden of cloud computing for massive data processing [2,3].

As a distributed service architecture, the distribution of resources and load status between fog nodes is uneven when there are numerous service requests. This makes modeling difficult and finding an optimal way to schedule tasks challenging [4]. The key to improving the performance of fog computing services is to provide an efficient and reasonable task scheduling strategy based on the characteristics of the fog computing environment [5,6]. At present, fog computing task scheduling algorithms are generally divided into three categories: static scheduling algorithms [7], heuristic scheduling algorithms [8,9], and meta-heuristic scheduling algorithms [10]. The characteristic of a static scheduling algorithm is that it is independent of the state and accessibility of system resources. To start scheduling, detailed information about tasks must be obtained, and the state and accessibility of system resources cannot be changed once the task is submitted to the system [11]. The goal of a heuristic scheduling algorithm is to find the approximate optimal solution to the current problem in a reasonable amount of time, which has significant advantages in solving all kinds of complex single objective optimization problems. A metaheuristic algorithm can avoid the defects of a heuristic algorithm in solving multi-objective optimization problems by combining a random search strategy with a local optimal strategy [12]. In addition to the above three methods, neural network-related technology [13,14] has also been applied to task scheduling in fog computing in recent years. However, due to the problems of too many algorithm parameters and poor anti-interference, it is often difficult to extend it to the actual application environment [15].

In real-world applications, where it is difficult to obtain accurate information about task complexity and node resource availability, a local plus random mutation metaheuristic algorithm is more effective in solving task scheduling problems with multiple conflicting goals. Additionally, task scheduling in fog computing is inherently NP-hard, and scalability, as well as resource utilization rate, are critical factors to consider. A metaheuristic algorithm is, therefore, suitable for solving such complex multi-objective optimization problems. This paper presents a study of fog computing task scheduling based on the MOEA/D optimization algorithm, which addresses the problems of existing task offloading, model construction, and algorithm research in three parts from the perspective of the trade-off between resource supply and demand. The specific contributions of this study are as follows:

1. Firstly, a task offloading model based on multi-decision information is proposed to determine the task offloading nodes based on multiple attributes; secondly, based on the task offloading model, the time delay and energy consumption in the task scheduling process of fog computing are analyzed to establish a multi-objective task scheduling model for fog computing, which provides a model based on multiple objectives.
2. To solve the problem of the uneven distribution of solutions acquired by the MOEA/D algorithm in the presence of a complex Pareto frontier, a multi-objective evolutionary algorithm M-E-AWA that dynamically adjusts the weight vector is presented. Second, we offer an external archive-based weight updating method, which employs an external archive to preserve the previous weights and this external archive to guide the weight vector update when a zero term appears in the new subproblem.
3. On the basis of the multi-objective scheduling model and the enhanced evolutionary algorithm, we propose a task scheduling approach based on M-E-AWA for solving the multi-objective scheduling model. A priority-based task scheduling strategy is designed to analyze multiple time stamps of tasks and establish the urgent priority of each task and use this priority to determine the order of task execution. Secondly, the tasks are converted into individuals conforming to evolutionary specifications through coding and decoding operations and are solved using genetic algorithms to obtain optimal solutions through a series of evolutionary processes such as population selection and mutation.

Section 2 of this paper proposes a multi-objective task scheduling model for fog computing. Section 3 presents the M-E-AWA algorithm proposed in this paper. Section 4 verifies the validity of the algorithm via experiments, and Section 5 provides the conclusion.

2. Model

2.1. Task Scheduling Model with Adaptive Priority Adjustment

This section proposes a task scheduling model based on adaptive priority adjustment. By considering the waiting time and generation time of tasks, the urgency degree is sorted out and grouped to ensure faster execution of high-priority tasks.

Here, j is defined to represent the priority level of task i; j = 1 means high priority, j = 2 means medium priority, and j = 3 means low priority; and three buffers, B_1, B_2 and B_3, are provided for the fog server to temporarily store the high, medium, and low priority tasks. For example, at i = 1 and j = 1, it means that the first task is stored in B_1. From a global perspective, tasks in B_1, B_2 and B_3 are scheduled sequentially. From a local point of view, tasks in the same buffer are scheduled by polling. However, some tasks cannot be completed in one slice and need to wait for the next round to execute the rest using other slices. In this case, because of the dynamics and uncertainty of the task, it is necessary to mark the urgency level of the task. Here, eme_{ij} is used to represent the urgency degree of $task_{ij}$, as defined in Equation (1):

$$eme_{ij} = \begin{cases} \lambda_1 \frac{\omega_1 time_{wai}+1}{time_{rem}} + \lambda_2 time_{com}, & time_{wai} < 0 \\ \frac{\xi}{\omega_2 time_{wai}+time_{rem}+time_{com}}, & time_{wai} \geq 0 \end{cases} \tag{1}$$

λ_1, λ_2, and ξ are experienced values. Several experiments have shown that the values of λ_1 = 1/3, λ_2 = 2/3, and ξ = 0.85 result in the most efficient task scheduling. $time_{wai}$ refers to the waiting time, when $time_{wai} < 0$, it means that the task has not been completed within the specified time, and $time_{wai}$ is the timeout time; When $time_{wai} > 0$, it indicates that the task has been completed within the specified time, and $time_{wai}$ represents the available time. $time_{rem}$ is the remaining time of the task and a higher value shows that more time is left, resulting in lower urgency for completing the task. $time_{com}$ is the task's generation time. To consider the generation time's weight of the expired task, it needs to be factored in ω_1 and ω_2 are two variable parameters whose definitions are presented in Equations (2) and (3):

$$\omega_1 = \begin{cases} 0.5 & |time_{wai}| < thr_1 \\ 1 & thr_1 \leq |time_{wai}| < 2thr_1 \\ 2 & 2thr_1 \leq |time_{wai}| \end{cases} \tag{2}$$

$$\omega_2 = \begin{cases} 0.4 & |time_{wai}| < thr_2 \\ 0.65 & thr_2 \leq |time_{wai}| < 2thr_2 \\ 1 & 2the_2 \leq |time_{wai}| \end{cases} \tag{3}$$

where thr_1 and thr_2 are two defined thresholds that satisfy $thr_1 < thr_2$.

A task scheduling strategy based on priority first sets buffers and distinguishes the priorities of buffers. Then the task is iterated over, and the priority with the highest urgency value is placed in the high priority buffer. Because the priorities of the three buffers are different, the time slices allocated in each buffer are different. The buffer with a higher priority is preferentially executed, and the execution time slices are proportional to the priority.

2.2. Multi-Objective Task Scheduling Model

This section provides an analysis of the task scheduling process in fog computing. The distribution of nodes in fog computing is highly scattered, resulting in unequal resource distribution owing to some regions being too sparse while others too dense. As a result, constructing a task scheduling model presents several challenges that require balancing multiple factors to avoid modeling bias while ensuring the desired effect [16–18]. In this paper, we propose a task offloading model based on multi-decision information, and select three computational objectives, namely transmission time, processing time, and energy consumption for the task scheduling model [19]. We adopt a fog computing architecture that prioritizes efficiency as the primary target. With the current hardware conditions, the transmission delay caused by the transmission medium is getting smaller, leading to more attention on the electrical energy problem during task processing. Efficient processing of tasks in fog computing is vital to ensure that the energy consumption of the data processing center does not exponentially increase with task volume [20,21]. We divide the time cost into two parts, transmission time and processing time. The transmission time calculation is directly proportional to the quality of medium transmission, with high-quality transmission resulting in a shorter transmission time. Processing time, on the other hand, relates directly to the computing resources available in fog computing. c allows us to highlight the critical points of our proposed task scheduling model. For the convenience of subsequent studies, it is assumed here that the data transmission during resource scheduling and computation result return is not limited by communication bandwidth and that the global communication consumption is relatively stable. This section uses the computational capacity of fog nodes as a measure to ensure that the allocated computational resources can satisfy the consumption of the service device before the deadline of the task is reached, which can reduce the task latency time and prevent task timeout. The relevant definitions of fog computing task scheduling discussed in this paper are as follows:

1. Set the fog computing node $F_{fog} = \{f_1, f_2, \ldots, f_j\}$, where f_j represents the j-th fog node, $1 \leq j \leq m$.
2. Set up a collection of tasks $T = \{t_1, t_2, \ldots, t_n\}$, n is the number of computational tasks, where t_i represents the i-th task, $1 \leq i \leq n$.
3. Set the allocation matrix $X_{ij} = (x_{ij})$, Task i is not assigned to fog node j if x_{ij} equals 0. A value of 1 indicates that fog node j is assigned to task i.
4. The time cost associated with a task is represented by the matrix ET of $m \times n$. ET_{ij} represents the time spent by task t_i on computation node f_j.
5. The service cost of a computational task is represented by the matrix C of $m \times n$. C_{ij} represents the cost of spending by task t_i on computation node f_j.

Table 1 explains the notation of the mathematical model.

Table 1. Mathematical notation.

Notation	Description
F	Fog node collection
U	Collection of terminal equipment
T	Mission collection
TS_i	Size of the t_i
TA_i	Arrival time of the t_i
$\gamma_{i,j}$	Transmission rate
cr_i	Size of task execution results
μ_j	Processing speed of the j-th fog node f_j
Φ_τ	Time cost
$\varphi_{i,j}$	Wireless interface parameters
O_i	The size of the i-th result
u_j	Compute the price once at compute node f_j

Transmission Time. T is used to represent the set of all tasks on the node, so the completion time of set T includes: the transmission time when the task is unloaded to the fog node, the transmission time when the processing result is returned to the user device, and the delay time generated when the task interacts with the user device.

Given a task t_i with a size of s_i, the calculation of the uplink transmission $T_{i,j} \uparrow$ of t_i unloaded to the j-th fog node is shown in Equation (4):

$$T_{i,j} \uparrow = \frac{s_i}{\gamma_{i,j}} \tag{4}$$

where $r_{i,j}$ is the transmission rate of the wireless network interface, which can be deduced from measurement [22,23]. Set cr_i as the size of the task execution result returned from the fog node, and the calculation of the transmission time $T_{i,j} \downarrow$ from the calculation result f_j to the task t_i is shown in Equation (5):

$$T_{i,j} \downarrow = \frac{cr_i}{\gamma_{i,j}} \tag{5}$$

The Equation (6) shows how to figure out the time delay $DelT_{i,j}$ caused by sending task i from the local transmission device to fog node f_j: where TS_i is the size of the current transmitted task:

$$DelT_{i,j} = \frac{TS_i}{\mu_j} \tag{6}$$

The time delay $DelT'_{i,j}$ of returning the task to the local device after the task is processed on the fog node is calculated as Equation (7), where O_i is the result size after the task is processed:

$$DelT'_{i,j} = \frac{O_i}{\mu_j} \tag{7}$$

To sum up, the calculation of the total transmission time needed to transmit task t_i between fog node f_j and the end user is shown in Equation (8):

$$ET^{trans}_{total} = \sum_{j=1}^{m} \sum_{i=1}^{n} \left(T_{i,j} \uparrow + T_{i,j} \downarrow + DelT_{i,j} + DelT'_{i,j} \right) \tag{8}$$

When fog computing schedules tasks, different fog nodes have different processing abilities. In general, the fog nodes with more allocated resources will process tasks faster. The processing time (CPU execution time) of task t_i on fog node f_j is set to be composed of three parts: the time $Sub_{i,j}$ when the i-th task is submitted to the j-th fog node, the waiting time $Wai_{i,j}$ of the i-th task on the j-th fog node, and the execution time $Ope_{i,j}$ of the i-th task on the j-th fog node. The first two times can be obtained directly at the fog node. The calculation of the execution time $Ope_{i,j}$ is shown in Formula (9), where μ_j is the processing speed of f_j in MIPS:

$$Ope_{i,j} = \frac{l_i}{\mu_j} \tag{9}$$

The calculation of total processing time is shown in Equation (10):

$$ET^{exe}_{total} = \sum_{j=1}^{m} \sum_{i=1}^{n} \left(Sub_{i,j} + Wai_{i,j} + Ope_{i,j} \right) \tag{10}$$

The more fog computing resources required to process the same number of tasks on a node, the shorter the processing time, and the higher the computational power of the node, the greater the computational cost. Therefore, reducing time costs while minimizing energy consumption is crucial. A service level agreement (SLA) is a negotiated service quality agreement between a provider and subscriber that primarily concerns measurable service quality characteristics such as service, service time, and billing. The service level objective (SLO) is a measure of subscriber satisfaction that is more punitive than the SLA and serves as an internal objective to quantify service received from fog nodes, taking into ac-count frequency, response events, and availability. The resource cost includes costs associated with SLO violations and endpoint request processing by computing nodes. If V represents the total penalty cost for SLO violation, the penalty function for SLO violation is computed as shown in (11):

$$V = \sum_{i}^{q} \left[\gamma_i + \eta \times \left(t_i^{stop} - t_i^{deadline} \right) \right] \tag{11}$$

i is the serial number of SLO violation requests due to service timeout, q is the total number of SLO violation requests, γ_i is the basic penalty cost of the i-th SLO violation request, η is the penalty cost per unit time, which is generally caused by timeout, and t_i^{stop} is the actual response time of the i-th SLO violation request.

Equation (12) shows the calculation of average resource consumption processed on f_j with given task t_i [24]:

$$C_{i,j} = u_j \cdot ET^{exe}_{i,j} \tag{12}$$

$maxC_{i,j}$ represents the resource consumption of each CPU cycle. In this section, it is assumed that CPU resources are uniformly allocated to each current process on each server. In other words, if a CPU is composed of multiple cores, the processes will be uniformly allocated to each core.

The resources required to unload the task t_i with the size of TS_i to the fog node f_j (including the resource consumption caused by the time delay in the transmission process) are calculated as shown in Equation (13), and $\varphi_{i,j}$ is a constant related to the wireless interface:

$$EP_{i,j} \uparrow = \frac{\varphi_{i,j} \cdot TS_i}{\gamma_{i,j}} \tag{13}$$

If O_i is set as the result data size of task t_i, the resource consumption of the output data from fog node f_j to a user's equipment (including the resource consumption caused by the time delay in transmission) is calculated as shown in Equation (14):

$$EP_{i,j} \downarrow = \frac{\varphi_{i,j} \cdot O_i}{\gamma_{i,j}} \tag{14}$$

Therefore, the calculation of total resource consumption is expressed in Equation (15):

$$EP_{total} = \left(\sum_{j=1}^{m} \sum_{i=1}^{n} \left(C_{i,j} + EP_{i,j} \uparrow + EP_{i,j} \downarrow \right) \right) + V \tag{15}$$

3. Algorithm

3.1. Framework of M-E-AWA

In the fog computing task scheduling field, a multi-objective evolutionary algorithm (M-E-AWA) based on adaptive distribution of weight vectors is proposed to address the problem of balancing multi-objective conflict and optimizing the Pareto front, which is complex and discontinuous. M-E-AWA first calculates the individual sparsity to determine the population sparsity difference. Using this information, it dynamically adjusts the weight vector to guide the adaptive evolution of the population. The M-E-AWA algorithm framework is presented in Table 2. The algorithm initializes the variables in Step 1, calculates the ideal point and sparse difference of the current population in Step 2, and applies evolutionary operators to new individual generations in Steps 3–7, including paternal crossover and mutation. Step 8 calculates the sparse difference of the new population before and after evolution. Step 9 adaptively adjusts the weight vector according to the population sparsity; the final step describes the termination condition. The population sparsity judgment and weight vector adaptive updating will be further described below.

Table 2. Pseudo code of M-E-AWA algorithm.

```
Input: Max iterations
Output: Optimal solution individuals $x^1, \ldots, x^N$ and function values $FV^1, \ldots, FV^N$

Step 1. Initialize population *pop*, weight vector $\lambda^1, \lambda^2, \ldots, \lambda^N$ and neighborhood matrix
$B(i) = \{i_1, \ldots, i_T\}, P = 0, EW = [], Q$;
Step 2. Calculate the ideal point $z^* = (z_1{}^*, \ldots, z_m^*)$ and sparse difference degree $SL\_D$ of the current
population;
The individuals in the population are traversed, and the evolution process is as follows:
Step 3. For each individual, randomly choose whether to use neighborhood for individual evolution guidance;
Step 4. Use crossover operator SBX for individual crossover operation;
Step 5. Use polynomial mutation operator for individual mutation operation;
Step 6. Update the reference point z;
Step 7. Update the optimal solution, if $g(y|\lambda^i, z^*) \le g(x^j|\lambda^i, z^*)$, then set $x^j = y,\ FV^j = F(y)$;
Step 8. Calculate the difference of sparse diversity of populations before and after evolution:
$dRate = |SL\_D' - SL\_D|$;
Step 9. If $dRate$ is greater than the threshold T, the weight vector is updated adaptively; Otherwise, proceed to
step 10;
Step 10. Repeat steps 2–9 above until the maximum number of iterations Max is reached.
```

3.2. Judgment of Population Sparsity

This paper determines individual sparsity based on a fast and effective non-dominated solution pruning method proposed by kukkonen and DEB (2006) [25] and proposes a population sparsity difference estimation method based on individual sparsity. Firstly, Q individuals with the largest and smallest individual sparsities are selected according to Formula (16), and then the population sparsity difference *SL_D* is obtained according to Formula (17). Where *i* represents the *i*-th neighbor, and the calculation of sparsity requires computing the distance and product of the M neighbors of the *j*-th individual.

$$SL\left(ind^{j}, pop\right) = \prod_{i=1}^{m} L_{2}^{NN_{i}^{j}} \tag{16}$$

$$SL\_D = \frac{\sum_{i=1}^{Q} \hat{SL} - \sum_{i=1}^{Q} \overset{\vee}{SL}}{Q}, 2Q < N \tag{17}$$

In this paper, we utilize the population sparse difference estimation method to implement the population adjustment strategy. Specifically, the sparse population difference *SL_D* is used to adjust the weight vector. Prior to an individual's adjustment in the population, the current population difference is compared to that of the previous generation. If the difference exceeds a pre-determined value, then the population will be adjusted accordingly. By utilizing this approach, we can avoid unnecessary resource consumption and excessive parameter settings that may exist in the original algorithm.

3.3. Adaptive Updating of Weight Vector

The weight vector is calculated according to the generated individual and the optimal solution in MOEA/D-AWA [26]. When facing the zero term subproblem, MOEA/D-AWA obtains the weight vector by adding and adjusting parameters, but this method cannot express the generation significance of the weight vector. In order to make the generated weight vector more uniform and improve the quality of the solution, this paper proposes a method to generate weights based on the historical archive. The previously removed weights are stored in the external archive. When generating the weights corresponding to the zero-term subproblem, the generation process is guided according to the information in the external archive, so as to ensure that the newly generated weight vector is more consistent with the population distribution characteristics and promote the subsequent solution to be more uniform. The generation of the weight vector is shown in Formula (18). When $\prod_{j=1}^{m}\left(f_j^{sp} - z_j^*\right) = 0 = 0$, the weight vector generation method proposed in this paper discards a parameter ε in the MOEA/D-AWA algorithm and adds an item W_i^e, which is the previously removed weight vector stored in the external archive.

$$\lambda^{sp} = \left(\frac{1}{2}\left(\frac{\frac{1}{f_1^{sp}-z_1^*}}{\sum_{k=1}^{m}\frac{1}{f_1^{sp}-z_k^*}} + W_1^e\right), \ldots, \frac{1}{2}\left(\frac{\frac{1}{f_m^{sp}-z_m^*}}{\sum_{k=1}^{m}\frac{1}{f_1^{sp}-z_k^*}} + W_{nus}^e\right)\right) \\ \prod_{j=1}^{m}\left(f_j^{sp} - z_j^*\right) = 0 \tag{18}$$

The weight vector update method based on the external archive is employed to adjust the population, ensuring that the generated weight vector complies with the distribution law of the population and thereby ensuring uniformity in the resulting solutions. The weight vector updating process during population evolution occurs as follows: First, the deletion operation is performed by inputting the population and the required number of adjustments. Next, the sparsity of individual population is calculated and the corresponding individuals are deleted in the appropriate area while simultaneously adding the deleted individual's corresponding weight vector to the EW for use as reference information for future weight vector generation. Second, the addition operation is performed by inputting the population to be adjusted, the required adjustment quantity, the optimal solution, and the weight external archive EW. Next, the area to be added is identified based on the population's sparsity, new individuals are added to the area, and a different weight generation scheme is used depending on the characteristics of the generated individuals when adding the corresponding weight vector.

4. Experiments and Results

4.1. Algorithm Performance Verification Experimental Setup

This section uses the PlatEMO framework for experiments. PlatEMO is an open-source and free MATLAB-based platform for evolutionary multi-objective optimization that can be run in any operating system that supports MATLAB.

To demonstrate the performance of the proposed algorithm, three sets of comparison experiments are set up in this section.

In the first set, the MOEA/D, NSGA-II, NSGA-III, and MOEAPSL algorithms are selected as the comparison algorithms and compared with the M-E-AWA algorithm on the DTLZ1 to DTLZ6 datasets.

The second set chooses the MOEA/D-AWA algorithm as the comparison algorithm and conducts experiments on the complex Pareto datasets IMOP1 and IMOP3 to test the performance of the M-E-AWA algorithm on the complex datasets.

The third set chose the MOEA/D-AWA algorithm as the comparison algorithm and designed an experiment to test the efficiency of population adjustment by comparing the number of adjustments of the M-E-AWA algorithm with the MOEA/D-AWA algorithm in order at 10,000, 15,000, and 20,000 evolutionary generations, respectively.

The evolutionary parameters of the algorithms in the comparison experiments were set the same except for the change in the evolutionary algebra of the algorithm used to perform the population sparse differences: the population size N was 300, the evolutionary algebra T was 10,000, and the genetic operators were random selection, simulated binary crossover, and polynomial variation, respectively. To show the rigor of the experiments, each experiment was run 30 times, and the results obtained were averaged.

The experiment uses IGD (inverted generational distance) and HV as multi-objective evaluation indexes.

4.2. Experimental Results and Analysis

In this section, comparative experiments and result analysis are performed. Tables 3–5 show the performance of M-E-AWA and the comparison algorithm on IGD evaluation metrics for different numbers of targets, and Tables 6–8 show the performance of M-E-AWA and the comparison algorithm on HV evaluation metrics for different numbers of targets.

Table 3. Comparison of IGD values between M-E-AWA and contrast algorithms on DTLZ1–DTLZ6(The target number is 8).

Problem	M	M-E-AWA	MOEAD	NSGAII
DTLZ1	8	4.4291×10^{-1} (4.03×10^{-1})	8.2955×10^{-1} (6.17×10^{-1})	3.1165×10^{1} (9.37×10^{0})
DTLZ2	8	5.8866×10^{-1} (5.91×10^{-2})	4.1397×10^{-1} (4.10×10^{-2})	1.5629×10^{0} (2.77×10^{-1})
DTLZ3	8	1.5775×10^{1} (9.12×10^{0})	2.1418×10^{1} (9.86×10^{0})	9.6284×10^{2} (1.84×10^{2})
DTLZ4	8	4.3454×10^{-1} (1.06×10^{-1})	8.6529×10^{-1} (9.90×10^{-2})	1.4744×10^{0} (1.63×10^{-1})
DTLZ5	8	5.9679×10^{-2} (1.54×10^{-2})	2.7225×10^{-1} (8.30×10^{-5})	4.5730×10^{-1} (1.70×10^{-1})
DTLZ6	8	2.7903×10^{-1} (1.54×10^{-1})	4.1497×10^{-1} (7.48×10^{-1})	7.7771×10^{0} (7.35×10^{-1})

Problem	M	NSGAIII	MOEAPSL
DTLZ1	8	1.6624×10^{0} (8.74×10^{-1})	2.2977×10^{1} (4.52×10^{0})
DTLZ2	8	4.2607×10^{-1} (3.88×10^{-2})	1.3682×10^{0} (1.38×10^{-1})
DTLZ3	8	6.0302×10^{1} (2.11×10^{1})	1.9829×10^{2} (1.40×101)
DTLZ4	8	4.7731×10^{-1} (8.26×10^{-2})	9.6292×10^{-1} (1.31×10^{-1})
DTLZ5	8	2.2922×10^{-1} (4.40×10^{-2})	8.6811×10^{-1} (1.69×10^{-1})
DTLZ6	8	4.7466×10^{0} (9.73×10^{-1})	7.4268×10^{-1} (6.02×10^{-1})

Table 4. Comparison of IGD values between M-E-AWA and contrast algorithms on DTLZ1–DTLZ6(The target number is 10).

Problem	M	M-E-AWA	MOEAD	NSGAII
DTLZ1	10	6.0764×10^{-1} (5.83×10^{-1})	7.1777×10^{-1} (3.78×10^{-1})	3.5617×10^{1} (1.71×10^{1})
DTLZ2	10	6.5076×10^{-1} (5.47×10^{-2})	5.1433×10^{-1} (5.83×10^{-2})	1.4143×10^{0} (1.66×10^{-1})
DTLZ3	10	1.2979×10^{1} (5.79×10^{0})	2.1555×10^{1} (9.69×10^{0})	9.3831×10^{2} (2.49×10^{2})
DTLZ4	10	6.1660×10^{-1} (7.93×10^{-2})	8.9900×10^{-1} (6.11×10^{-2})	1.4922×10^{0} (1.19×10^{-1})
DTLZ5	10	7.0864×10^{-2} (7.02×10^{-3})	2.7223×10^{-1} (7.79×10^{-5})	5.3538×10^{-1} (1.64×10^{-1})
DTLZ6	10	2.4692×10^{-1} (2.43×10^{-1})	3.2992×10^{-1} (2.35×10^{-1})	7.7899×10^{0} (5.50×10^{-1})

Problem	M	NSGAIII	MOEAPSL
DTLZ1	10	2.4867×10^{0} (7.74×10^{-1})	2.1557×10^{1} (1.11×10^{1})
DTLZ2	10	4.9281×10^{-1} (3.92×10^{-2})	1.7235×10^{0} (1.69×10^{-1})
DTLZ3	10	8.7370×10^{1} (3.24×10^{1})	2.1605×10^{2} (1.11×10^{1})
DTLZ4	10	5.5091×10^{-1} (5.86×10^{-2})	1.5496×10^{0} (1.51×10^{-1})
DTLZ5	10	2.3942×10^{-1} (3.83×10^{-2})	8.5802×10^{-1} (1.77×10^{-1})
DTLZ6	10	6.0815×10^{0} (6.61×10^{-1})	1.0980×10^{0} (8.12×10^{-1})

Table 5. Comparison of IGD values between M-E-AWA and contrast algorithms on DTLZ1–DTLZ6(The target number is 15).

Problem	M	M-E-AWA	MOEAD	NSGAII
DTLZ1	15	4.0954×10^{-1} (2.46×10^{-1})	1.1793×10^{0} (7.28×10^{-1})	4.4001×10^{1} (1.91×10^{1})
DTLZ2	15	9.9356×10^{-1} (3.96×10^{-2})	9.4064×10^{-1} (4.50×10^{-2})	1.3856×10^{0} (1.14×10^{-1})
DTLZ3	15	1.1870×10^{1} (6.93×10^{0})	2.6347×10^{1} (1.43×10^{1})	8.0812×10^{2} (1.83×10^{2})
DTLZ4	15	9.9511×10^{-1} (6.54×10^{-2})	9.9052×10^{-1} (2.96×10^{-2})	1.5103×10^{0} (9.09×10^{-2})
DTLZ5	15	2.0677×10^{-1} (1.94×10^{-2})	2.5736×10^{-1} (1.68×10^{-1})	7.3628×10^{-1} (3.16×10^{-1})
DTLZ6	15	2.2747×10^{-1} (4.28×10^{-2})	4.9940×10^{-1} (3.57×10^{-1})	8.1212×10^{0} (6.06×10^{-1})

Problem	M	NSGAIII	MOEAPSL
DTLZ1	15	1.0025×10^{0} (6.69×10^{-1})	5.6511×10^{0} (1.07×10^{1})
DTLZ2	15	7.5887×10^{-1} (2.19×10^{-2})	1.9864×10^{0} (2.14×10^{-1})
DTLZ3	15	2.9381×10^{1} (1.22×10^{1})	2.3105×10^{2} (1.75×10^{1})
DTLZ4	15	7.7430×10^{-1} (1.32×10^{-2})	1.9075×10^{0} (1.59×10^{-1})
DTLZ5	15	3.0013×10^{-1} (5.73×10^{-2})	1.0706×10^{0} (2.64×10^{-1})
DTLZ6	15	3.2375×10^{0} (7.60×10^{-1})	2.0772×10^{0} (1.36×10^{0})

Table 6. Comparison of HV values between M-E-AWA and contrast algorithms on DTLZ1–DTLZ6(The target number is 8).

Problem	M	M-E-AWA	MOEAD	NSGAII
DTLZ1	8	3.5458×10^{-1} (3.62×10^{-1})	1.2144×10^{-1} (1.99×10^{-1})	0.0000×10^{0} (0.00×10^{0})
DTLZ2	8	7.2299×10^{-1} (5.04×10^{-2})	8.1531×10^{-1} (6.83×10^{-2})	1.5631×10^{-3} (4.00×10^{-3})
DTLZ3	8	0.0000×10^{0} (0.00×10^{0})	0.0000×10^{0} (0.00×10^{0})	0.0000×10^{0} (0.00×10^{0})
DTLZ4	8	8.8075×10^{-1} (9.12×10^{-2})	4.1038×10^{-1} (1.26×10^{-1})	8.8923×10^{-5} (3.19×10^{-4})
DTLZ5	8	1.0017×10^{-1} (9.66×10^{-3})	9.0912×10^{-2} (1.75×10^{-4})	8.4771×10^{-3} (1.78×10^{-2})
DTLZ6	8	8.7927×10^{-2} (1.66×10^{-2})	4.7607×10^{-2} (4.32×10^{-2})	0.0000×10^{0} (0.00×10^{0})

Problem	M	NSGAIII	MOEAPSL
DTLZ1	8	5.6497×10^{-3} (3.07×10^{-2})	0.0000×10^{0} (0.00×10^{0})
DTLZ2	8	8.3894×10^{-1} (3.19×10^{-2})	4.2415×10^{-4} (1.54×10^{-3})
DTLZ3	8	0.0000×10^{0} (0.00×10^{0})	0.0000×10^{0} (0.00×10^{0})
DTLZ4	8	8.1086×10^{-1} (8.12×10^{-2})	3.6243×10^{-2} (7.58×10^{-2})
DTLZ5	8	3.8384×10^{-2} (2.14×10^{-2})	0.0000×10^{0} (0.00×10^{0})
DTLZ6	8	0.0000×10^{0} (0.00×10^{0})	6.6868×10^{-2} (4.10×10^{-2})

Table 7. Comparison of HV values between M-E-AWA and contrast algorithms on DTLZ1–DTLZ6(The target number is 10).

Problem	M	M-E-AWA	MOEAD	NSGAII
DTLZ1	10	3.1212×10^{-1} (3.58×10^{-1})	8.1261×10^{-2} (1.38×10^{-1})	0.0000×10^{0} (0.00×10^{0})
DTLZ2	10	7.9919×10^{-1} (4.16×10^{-2})	7.8331×10^{-1} (8.26×10^{-2})	2.7457×10^{-3} (6.85×10^{-3})
DTLZ3	10	0.0000×10^{0} (0.00×10^{0})	0.0000×10^{0} (0.00×10^{0})	0.0000×10^{0} (0.00×10^{0})
DTLZ4	10	8.8557×10^{-1} (5.14×10^{-2})	4.5787×10^{-1} (9.18×10^{-2})	4.4690×10^{-4} (2.07×10^{-3})
DTLZ5	10	9.0859×10^{-2} (2.25×10^{-4})	9.9513×10^{-2} (1.91×10^{-3})	2.3753×10^{-3} (8.64×10^{-3})
DTLZ6	10	7.5849×10^{-2} (3.45×10^{-2})	4.5833×10^{-2} (4.12×10^{-2})	0.0000×10^{0} (0.00×10^{0})
Problem	**M**	**NSGAIII**	**MOEAPSL**	
DTLZ1	10	0.0000×10^{0} (0.00×10^{0})	9.2380×10^{-3} (5.05×10^{-2})	
DTLZ2	10	7.9111×10^{-1} (4.99×10^{-2})	1.2855×10^{-5} (5.07×10^{-5})	
DTLZ3	10	0.0000×10^{0} (0.00×10^{0})	0.0000×10^{0} (0.00×10^{0})	
DTLZ4	10	7.4365×10^{-1} (1.18×10^{-1})	3.0533×10^{-5} (1.35×10^{-4})	
DTLZ5	10	9.2770×10^{-3} (1.45×10^{-2})	0.0000×10^{0} (0.00×10^{0})	
DTLZ6	10	0.0000×10^{0} (0.00×10^{0})	5.1471×10^{-2} (4.58×10^{-2})	

Table 8. Comparison of HV values between M-E-AWA and contrast algorithms on DTLZ1–DTLZ6(The target number is 15).

Problem	M	M-E-AWA	MOEAD	NSGAII
DTLZ1	15	3.2311×10^{-1} (2.50×10^{-1})	5.6578×10^{-2} (1.48×10^{-1})	0.0000×10^{0} (0.00×10^{0})
DTLZ2	15	2.4345×10^{-1} (4.20×10^{-2})	2.5775×10^{-1} (5.54×10^{-2})	7.2152×10^{-3} (9.69×10^{-3})
DTLZ3	15	7.3166×10^{-6} (4.01×10^{-5})	0.0000×10^{0} (0.00×10^{0})	0.0000×10^{0} (0.00×10^{0})
DTLZ4	15	4.4725×10^{-1} (7.58×10^{-2})	3.4450×10^{-1} (3.56×10^{-2})	2.5591×10^{-3} (1.18×10^{-2})
DTLZ5	15	9.1957×10^{-2} (2.00×10^{-4})	8.6435×10^{-2} (2.18×10^{-2})	4.7512×10^{-4} (2.29×10^{-3})
DTLZ6	15	9.1972×10^{-2} (2.53×10^{-4})	4.7385×10^{-2} (4.37×10^{-2})	0.0000×10^{0} (0.00×10^{0})
Problem	**M**	**NSGAIII**	**MOEAPSL**	
DTLZ1	15	1.2245×10^{-1} (2.65×10^{-1})	5.6172×10^{-1} (1.50×10^{-1})	
DTLZ2	15	6.5685×10^{-1} (3.69×10^{-2})	0.0000×10^{0} (0.00×10^{0})	
DTLZ3	15	0.0000×10^{0} (0.00×10^{0})	0.0000×10^{0} (0.00×10^{0})	
DTLZ4	15	7.8394×10^{-1} (2.33×10^{-2})	0.0000×10^{0} (0.00×10^{0})	
DTLZ5	15	7.8891×10^{-2} (1.47×10^{-2})	0.0000×10^{0} (0.00×10^{0})	
DTLZ6	15	0.0000×10^{0} (0.00×10^{0})	2.4254×10^{-2} (4.09×10^{-2})	

4.2.1. Comparative Analysis of Multi-Algorithm Conventional Data Sets

Shown in this section are data comparisons of the M-E-AWA algorithm with the classical MOEA/D, MOEA/D-AWA, and NSGA-III algorithms. As shown in Tables 3–7, the algorithm proposed in this paper has the smallest IGD and the largest HV value on the data sets DTLZ2, DTLZ3, DTLZ5, and DTLZ6 compared to the other algorithms, so the uniformity and convergence of the solutions obtained by the M-E-AWA algorithm on these data sets are better than the comparison algorithms. Since DTLZ1 and DTLZ3 have the property of testing the convergence performance of the algorithms, it indicates that the solutions of M-E-AWA are closer to the real Pareto frontier than the solutions of MOEA/D, MOEA/D-AWA, and NSGA-III algorithms. The performance of this algorithm is slightly worse on two data sets, DTLZ2 and DTLZ4. The DTLZ2 and DTLZ4 test functions are designed to test the diversity of the algorithms, so the distributivity of the M-E-AWA algorithm needs further improvement.

4.2.2. Comparative Analysis of Algorithms on Complex Pareto Frontier Dataset

This section compares M-E-AWA to MOEA/D-AWA using complex Pareto on the datasets IMOP1 and IMOP2. Figure 1a displays the results of the M-E-AWA algorithm against MOEA/D-AWA on the IMOP1 dataset, which is a complex dataset with spikes.

The solution distribution of the M-E-AWA algorithm is shown in red, while the distribution of the solutions of the MOEA/D-AWA algorithm is shown in grey. It can be seen that the improved algorithm is more efficient, with a slightly more uniform solution distribution compared to the MOEA/D-AWA algorithm. Figure 1b shows the comparison of the CPF results between the M-E-AWA algorithm and the MOEA/D-AWA on the IMOP2 dataset. The solution distribution of the improved algorithm is displayed in blue, while the solution distribution of the MOEA/D-AWA algorithm is shown in grey. It can be observed that the present algorithm fits the PF better and has a more uniform solution distribution than the MOEA/D-AWA algorithm within the same number of iterations. Additionally, this paper observes CPFs of both algorithms when lengthening the number of iterations and experimentally demonstrates that MOEA/D-AWA can achieve similar results after about 30,000 iterations. These further highlights that the improved algorithm proposed in this paper reduces the time complexity of the original algorithm and speeds up the evolution.

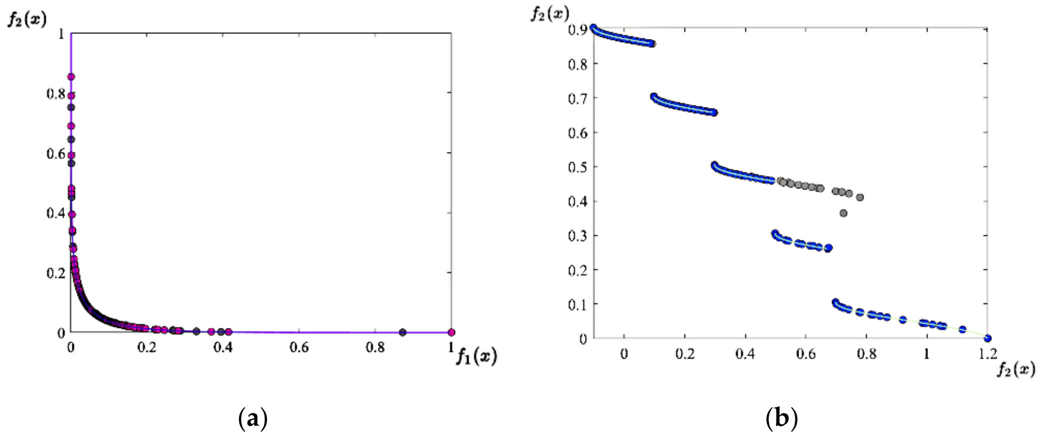

Figure 1. Comparison of MOEA/D-AWA and M-E-AWA on complex datasets. ((**a**) Comparison of MOEA/D-AWA and M-E-AWA on IOMP1. (**b**) Comparison of MOEA/D-AWA and M-E-AWA on IMOP2).

4.2.3. Comparison of the Number of Adjustments under Different Iterations

The experiments in this section mainly serve to verify the superiority of the improved algorithm M-E-AWA over the original algorithm in terms of the number of iterations required to change the weights adaptively for increasing numbers of iterations. As depicted in Figures 2–4, at the maximum number of evolutionary generations of 10,000, 15,000, and 20,000, the MOEA/D-AWA algorithm increases proportionally to the maximum number of iterations. However, this algorithm merely adjusts based on the population characteristics of the dataset without significant interference from changes in the number of iterations. This highlights that the weight adjustment efficiency of this algorithm is higher than that of the original algorithm.

4.3. Experimental Setup for Algorithm Validation

The convergence and distribution of the solution set by the M-E-AWA algorithm on the IMOP1 and IMOP2 data sets, having complex Pareto fronts, were analyzed to verify its effectiveness. Additionally, a simulation experiment was conducted in a fog computing environment using 5, 10, 15, and 20 fog nodes to solve the multi-objective optimization of the task scheduling model. Table 9 presents the clock rate and available memory size [27,28] of each CPU in the 20 simulated fog nodes that were utilized to perform tasks during the experiment. Table 10 illustrates the main parameter settings of the M-E-AWA algorithm, which includes population size (PS), CPU time (CT), neighbor size (T), crossover probability (α), and mutation probability (β).

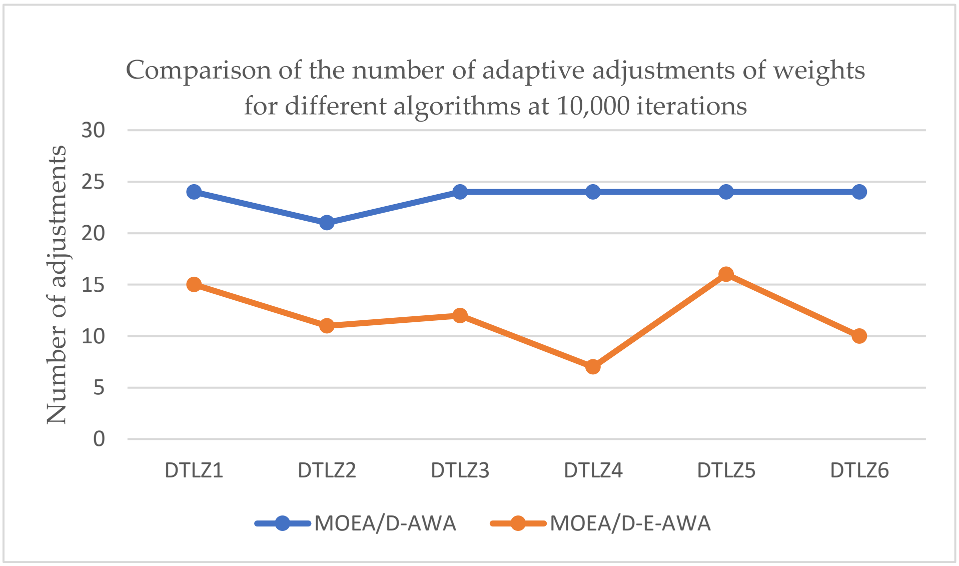

Figure 2. Comparison of weight adaptive adjustment times of M-E-AWA and MOEA/D-AWA algorithms when the maximum number of iterations is 10,000.

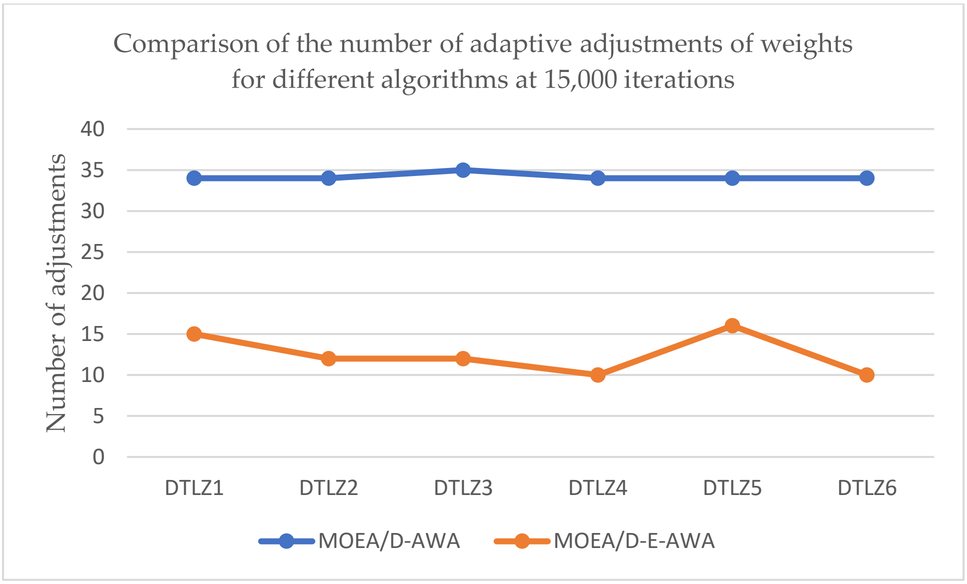

Figure 3. Comparison of weight adaptive adjustment times of M-E-AWA and MOEA/D-AWA algorithms when the maximum number of iterations is 15,000.

Table 9. The *j*-th column represents the data information when the number of fog nodes is *j*.

FN_j	1	2	3	4	5	6	7	8	9	10
CPU clock rate	1.25	1.00	0.83	1.00	0.83	1.25	0.90	0.77	1.11	1.00
Available memory size	1.0	0.9	1.4	1.5	1.4	1.0	1.2	0.9	1.0	1.2
FN_j	11	12	13	14	15	16	17	18	19	20
CPU clock rate	0.77	1.11	1.00	0.90	1.25	0.83	0.83	1.00	1.25	1.00
Available memory size	1.2	1.1	1.0	0.8	1.2	1.4	1.2	1.5	1.0	0.8

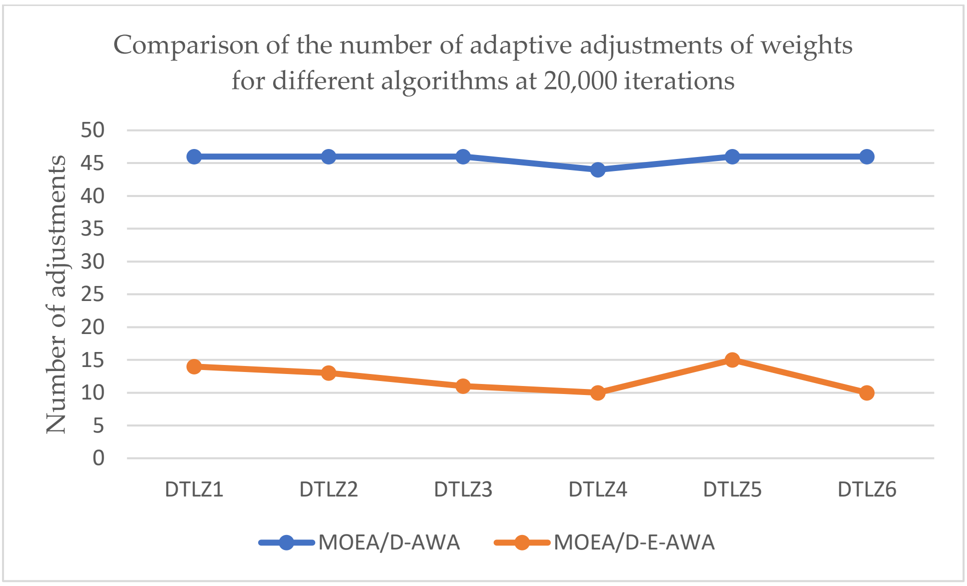

Figure 4. Comparison of weight adaptive adjustment times of M-E-AWA and MOEA/D-AWA algorithms when the maximum number of iterations is 20,000.

Table 10. Parameters Table.

Parameter	**Parameter Type**		
	1	**2**	**3**
PS	50	80	100
CT	0.1	0.2	0.25
T	2	5	10
α	0.8	0.85	0.9
β	0.05	0.1	0.2

The experiment uses latency and execution time, SLA violation rate and service cost as evaluation metrics.

The experiments use latency and execution time, and SLA violation rate as evaluation metrics. A brief description of these metrics follows:

1. Execution time: The completion time of the task, T_{realF}, is the most intuitive reflection of the speed of the algorithm execution.
2. Delay Time: Delay time is the difference between the completion time of a task and the time it is expected to be completed. The longer the delay, the worse the execution. The calculation is shown in Equation (19)

$$T_{delay} = T_{realF} - T_{\mathrm{exp}F} \tag{19}$$

3. SLA violation rate: The *task* in the formula represents the set of tasks, $task_i$ is to represent the ith task; *TASK* represents all tasks submitted by the user to generate SLA violations. Indicates the proportion of tasks that have generated SLA violations in the scheduling process for all tasks submitted by users, The calculation is shown in Equations (20) and (21).

$$DER = \frac{\sum_{i=1}^{|Task|} t_i^{def}}{|Task|} \tag{20}$$

$$task_i^{def} = \begin{cases} 1 & t_i^{sta} == ab \ or \ t_i^{ft} > t_i^{dl} \\ 0 & otherwise \end{cases} \tag{21}$$

4. Service cost: the resources consumed by the algorithm during its operation, which mainly contain the resources consumed by transmission, delay, processing, and other processes as well as the energy consumption due to SLO violation; the calculation formula is shown in (22).

$$COST = C_{exe} + C_{trans} + V \tag{22}$$

4.4. Experimental Results and Analysis

In the fog computing multi-objective task scheduling model constructed in Section 2, the model uses three objectives: transmission time, processing time, and resource cost, to establish a multi-objective problem. In this section, the M-E-AWA algorithm and comparison algorithm are applied to the scheduling model for simulation experiments. The comparison algorithms include the classic round-robin scheduling algorithm (CRRSA) [29] and the simple genetic algorithm (SGA) [30,31]. The experiment is divided into two parts: the first part is to compare the delay time and execution time; the second part is the experiment comparing the violation rate and service cost.

4.4.1. Time Delay and Total Execution Time

By setting different numbers of fog computing nodes, the execution time and delay time of each algorithm are compared with those of other algorithms. Figures 5 and 6 show the execution and delay times (time in seconds) obtained using different scheduling algorithms. As shown in Figure 5, the advantage of the M-E-AWA method increases as the number of fog computing nodes increases, indicating that the increase in computing nodes can enhance the performance of resource allocation in the face of the same number of tasks, and therefore the algorithm becomes more and more efficient. As shown in Figure 6, when the number of computing nodes is small, the M-E-AWA algorithm can better represent the advantage with the same available resources because its own good distribution can better handle the same number of tasks and use the evaluation function to evaluate the performance of the solutions in each solution set, while with the increase in the number of computing nodes, since the number of tasks always remains at a fixed amount, this algorithm's advantage is decreasing.

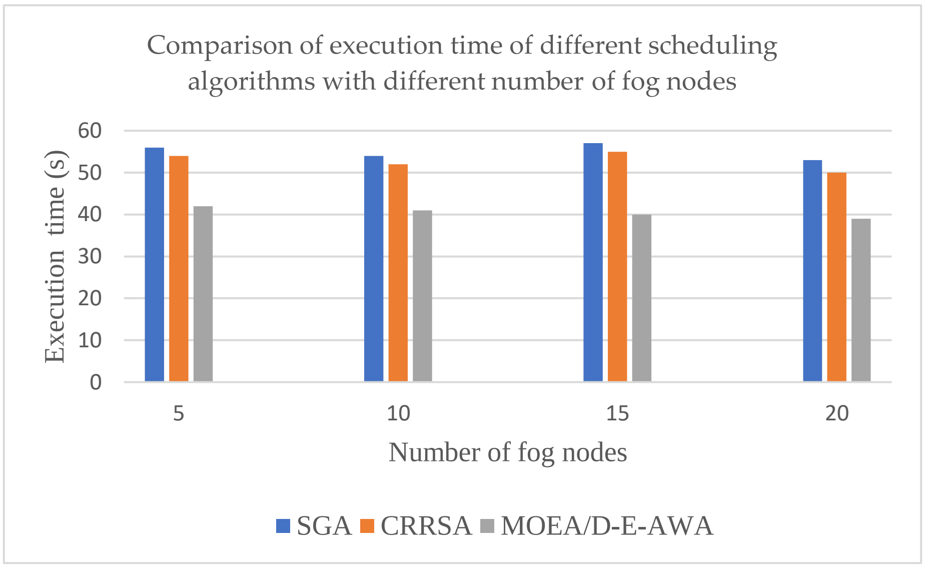

Figure 5. Comparison of total execution time.

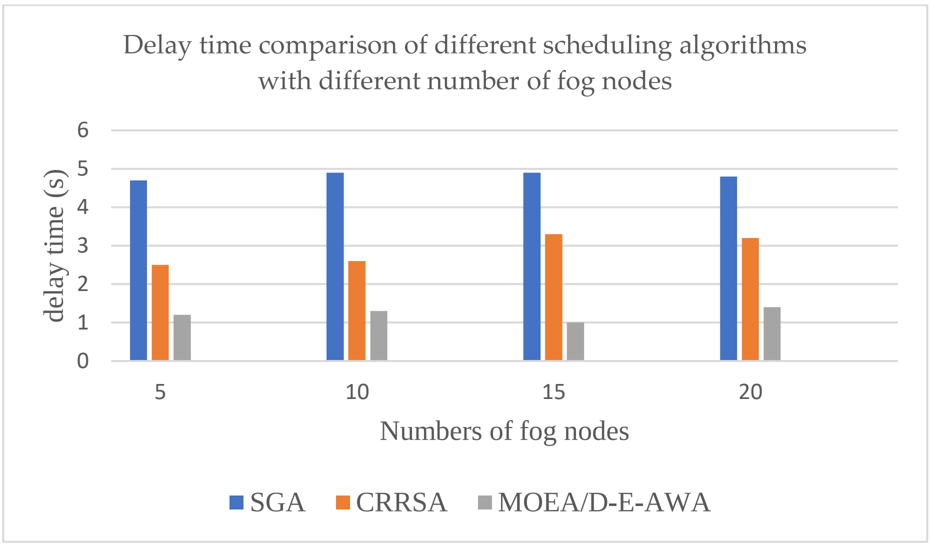

Figure 6. Comparison of delay time.

4.4.2. Violation Rate and Service Cost

As shown in Figure 7, the overall violation rate of M-E-AWA is lower than that of CRRSA and SGA, mainly because the optimization of multiple objectives is considered in the M-E-AWA algorithm, which has a better performance in obtaining the optimal solution and effectively shortens the response time, thus reducing the violation rate of SLO. In addition, as the number of fog computing nodes increases, the capacity of M-E-AWA to handle the task is greatly enhanced, and the growth rate of violations is slowing down. As shown in Figure 8, the overall service cost of M-E-AWA is also lower than other comparative methods. The main reason is that the overall violation rate of the assignment strategy in this paper is lower, which leads to a decrease in the service cost, thus effectively reducing the overall service cost.

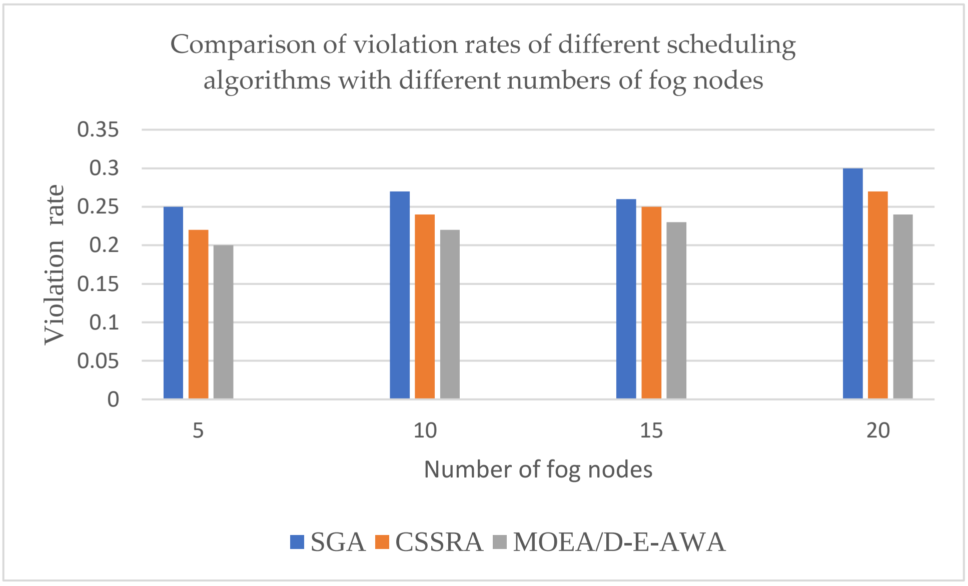

Figure 7. Comparison of Violation Rate.

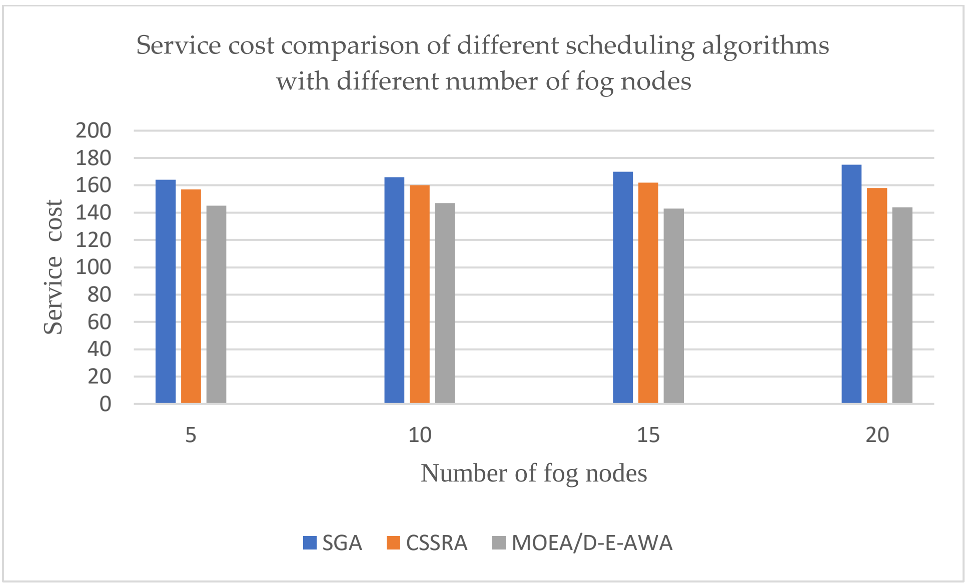

Figure 8. Comparison of Service Cost.

To sum up, based on the established multi-objective task scheduling model, the M-E-AWA algorithm performs well when the number of fog nodes increases. When dealing with the two task scheduling goals of execution time and service cost, it can make the two goals reach the optimal value at the same time.

5. Conclusions

In this paper, aiming at the problem that multi-objective conflict is difficult to balance and the Pareto front to the optimization model is complex and discontinuous in the fog computing task scheduling field, a multi-objective optimization model of task scheduling with dynamic priority adjustment is constructed, and a multi-objective evolutionary algorithm M-E-AWA based on weight vector adaptive update is proposed to solve the multi-objective task scheduling model. The experimental results show that the M-E-AWA algorithm outperforms the metrics on several data sets and demonstrates the tuning efficiency of the M-E-AWA algorithm compared to the original algorithm. M-E-AWA algorithm can deal with the multi-objective optimization problem with a complex Pareto front and has great advantages in the indicators of total computing time and service cost of task scheduling. With the increase in the number of tasks, the M-E-AWA method can achieve the purpose of dimensionality reduction based on its decomposition idea, so that it can still maintain the advantages of high computing efficiency when solving the high-dimensional tasks. The subsequent two issues are still an issue:

1. The task scheduling model proposed in this paper, in order to simplify the problem model when calculating the time cost, ignores that there are still some uncertainties in task scheduling in fog computing and should consider turning the uncertainties into intermediate parameters to establish an objective multi-objective task scheduling model for fog computing, which can make the solution results more accurate and more consistent with the actual environment.
2. The improved algorithm M-E-AWA proposed in this paper outperforms the original algorithm in a complex Pareto environment, while the calculation of population sparsity requires the calculation of individuals sequentially due to the inclusion of population sparsity judgment in it. Although the experiment proves to be able to significantly reduce the update frequency of the algorithm, the time complexity of the algorithm increases when the number of individuals in the population increases dramatically. Based on this, new judgment methods need to be considered later

that can make the frequency of population updates and the time consumed decrease simultaneously and reduce the time complexity.

Author Contributions: Conceptualization, W.D. and C.G.; Methodology, Z.D.; Software, Z.D. and Q.M.; Validation, Z.D., Q.M. and X.S.; Formal analysis, Q.M. and B.Y.; Investigation, X.S.; Resources, Q.M. and B.Y.; Data curation, B.Y. and X.S.; Writing—original draft, Z.D. and Q.M.; Writing—review & editing, W.D.; Visualization, X.S. and B.Y.; Supervision, C.G.; Project administration, W.D.; Funding acquisition, W.D. and C.G. All authors have read and agreed to the published version of the manuscript.

Funding: This work was sponsored by the Nature Science Foundation of Shanghai, China, 23ZR1414900; Shanghai Sailing Program, China, 20YF1410900; Shanghai Science and Technology Innovation Action Plan, China, 22ZR1416500; Shanghai Science and Technology Innovation Action Plan, China, 20dz1201400.

Data Availability Statement: The data presented in this study are available in the manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Al-Sarawi, S.; Anbar, M.; Abdullah, R.; Al Hawari, A.B. Internet of things market analysis forecasts, 2020–2030. In Proceedings of the 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), London, UK, 27–28 July 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 449–453.
2. Amiri, A.; Zdun, U.; Van Hoorn, A. Modeling and empirical validation of reliability and performance trade-offs of dynamic routing in service-and cloud-based architectures. *IEEE Trans. Serv. Comput.* **2021**, *15*, 3372–3386. [CrossRef]
3. Syed, S.A.; Rashid, M.; Hussain, S.; Azim, F.; Zahid, H.; Umer, A.; Waheed, A.; Zareei, M.; Vargas-Rosales, C. QoS Aware and Fault Tolerance Based Software-Defined Vehicular Networks Using Cloud-Fog Computing. *Sensors* **2022**, *2022*, 401. [CrossRef]
4. Hoseiny, F.; Azizi, S.; Shojafar, M.; Tafazolli, R. Joint QoS-aware and Cost-efficient Task Scheduling for Fog-Cloud Resources in a Volunteer Computing System. *ACM Trans. Internet Technol.* **2020**, *21*, 1–21. [CrossRef]
5. Pallewatta, S.; Kostakos, V.; Buyya, R. QoS-aware placement of microservices-based IoT applications in Fog computing environments. *Future Gener. Comput. Syst. FGCS* **2022**, *131*, 121–136. [CrossRef]
6. Lenuwat, P.; Boon-Itt, S. Information technology management and service performance management capabilities: An empirical study of the service supply chain management process. *J. Adv. Manag. Res.* **2022**, *19*, 55–77. [CrossRef]
7. Alizadeh, M.R.; Khajehvand, V.; Rahmani, A.M.; Akbari, E. Task scheduling approaches in fog computing: A systematic review. *Int. J. Commun. Syst.* **2020**, *33*, e4583. [CrossRef]
8. Liu, Y.; Lee, M.J.; Zheng, Y. Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system. *IEEE Trans. Mob. Comput.* **2015**, *15*, 2398–2410. [CrossRef]
9. Li, K. Heuristic Computation Offloading Algorithms for Mobile Users in Fog Computing. *ACM Trans. Embed. Comput. Syst.* **2021**, *20*, 3426852. [CrossRef]
10. Hoang, D.; Dang, T.D. FBRC: Optimization of task scheduling in fog-based region and cloud. In Proceedings of the 2017 IEEE Trustcom/BigDataSE/ICESS, Sydney, NSW, Australia, 1–4 August 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1109–1114.
11. Lu, H.; Gu, C.; Luo, F.; Ding, W.; Liu, X. Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning. *Future Gener. Comput. Syst.* **2020**, *102*, 847–861. [CrossRef]
12. Hosseinioun, P.; Kheirabadi, M.; Tabbakh, S.R.K.; Ghaemi, R. A new energy-aware tasks scheduling approach in fog computing using hybrid meta-heuristic algorithm. *J. Parallel Distrib. Comput.* **2020**, *143*, 88–96. [CrossRef]
13. Bitam, S.; Zeadally, S.; Mellouk, A. Fog computing job scheduling optimization based on bees swarm. *Enterp. Inf. Syst.* **2018**, *12*, 373–397. [CrossRef]
14. Pei, S.; Wu, Y.; Qiu, M. Neural Network Compression and Acceleration by Federated Pruning. In Proceedings of the 20th International Conference on Algorithm and Architecture for Parallel Processing (ICA3PP 2020), New York, NY, USA, 2–4 October 2020; pp. 1–10.
15. Lu, H.; Gu, C.; Luo, F.; Ding, W.; Zheng, S.; Shen, Y. Optimization of Task Offloading Strategy for Mobile Edge Computing Based on Multi-Agent Deep Reinforcement Learning. *IEEE Access* **2020**, *8*, 202573–202584. [CrossRef]
16. Pei, S.; Wu, Y.; Guo, J.; Qiu, M. Neural Network Pruning by Recurrent Weights for Finance Market. *ACM Trans. Internet Technol.* **2022**, *22*, 1–23. [CrossRef]
17. Alabbadi, A.A.; Abulkhair, M.F. Multi-Objective Task Scheduling Optimization in Spatial Crowdsourcing. *Algorithms* **2021**, *14*, 77. [CrossRef]
18. Nikseresht, M. MOGATS: A multi-objective genetic algorithm-based task scheduling for heterogeneous embedded systems. *Int. J. Embed. Syst.* **2021**, *14*, 171–184. [CrossRef]
19. Ali, I.M.; Sallam, K.M.; Moustafa, N.; Chakraborty, R.; Ryan, M.; Choo, K.K.R. An Automated Task Scheduling Model using Non-Dominated Sorting Genetic Algorithm II for Fog-Cloud Systems. *IEEE Trans. Cloud Comput.* **2020**, *10*, 2294–2308. [CrossRef]

20. Chatzikonstantinou, C.; Konstantinidis, D.; Dimitropoulos, K.; Daras, P. Recurrent neural network pruning using dynamical systems and iterative fine-tuning. *Neural Netw.* **2021**, *143*, 475–488. [CrossRef] [PubMed]
21. Guo, X. Multi-objective task scheduling optimization in cloud computing based on fuzzy self-defense algorithm. *AEJ Alex. Eng. J.* **2021**, *60*, 5603–5609. [CrossRef]
22. Zhang, Q.; Li, H. MOEA/D: A Multi-objective Evolutionary Algorithm Based on Decomposition. *IEEE Trans. Evol. Comput.* **2007**, *11*, 712–731. [CrossRef]
23. Yadav, A.K.; Mandoria, H.L. Study of task scheduling algorithms in the cloud computing environment: A review. *Int. J. Comput. Sci. Inf. Technol.* **2017**, *8*, 462–468.
24. Zhang, C.; Gao, L.; Li, X.; Shen, W.; Zhou, J.; Tan, K.C. Resetting Weight Vectors in MOEA/D for Multiobjective Optimization Problems with Discontinuous Pareto Front. *IEEE Trans. Cybern.* **2021**, *99*, 1–14. [CrossRef] [PubMed]
25. Kukkonen, S.; Deb, K. *A Fast and Effective Method for Pruning of Non-Dominated Solutions in Many-Objective Problems*; Parallel Problem Solving from Nature-PPSN IX; Springer: Berlin/Heidelberg, Germany, 2006; pp. 553–562.
26. Ma, X.; Yu, Y.; Li, X.; Qi, Y.; Zhu, Z. A survey of weight vector adjustment methods for decomposition-based multi-objective evolution algorithms. *IEEE Trans. Evol. Comput.* **2020**, *24*, 634–649. [CrossRef]
27. Wang, Y.; Sun, Y.; Sun, Y. Task scheduling algorithm in cloud computing based on fairness load balance and minimum completion time. In Proceedings of the 2015 4th National Conference on Electrical, Electronics and Computer Engineering, Xi'an, China, 12–13 December 2015; Atlantis Press: Amsterdam, The Netherlands, 2015; pp. 836–842.
28. Abualigah, L. A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments. *Clust. Comput.* **2021**, *24*, 205–223. [CrossRef]
29. Pradhan, P.; Behera, P.K.; Ray, B.N.B. Modified round robin algorithm for resource allocation in cloud computing. *Procedia Comput. Sci.* **2016**, *85*, 878–890. [CrossRef]
30. Moggridge, P.; Helian, N.; Sun, Y.; Lilley, M.; Veneziano, V.; Eaves, M. Revising max-min for scheduling in a cloud computing context. In Proceedings of the 2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Poznan, Poland, 21–23 June 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 125–130.
31. Agarwal, N.; Shrivastava, N.; Pradhan, M.K. Ananya Algorithm: A Simple and New Optimization Algorithm for Engineering Optimization. In Proceedings of the 2021 4th Biennial International Conference on Nascent Technologies in Engineering (ICNTE), Mumbai, India, 15–16 January 2021.