future internet

MDPI

Article

Task-Driven Virtual Machine Optimization Placement Model and Algorithm

Ran Yang [1], Zhaonan Li [1], Junhao Qian [2] and Zhihua Li [1,*]

[1] School of Artificial Intelligence and Computer Science, Jiangnan University, Wuxi 214000, China; 6223115007@stu.jiangnan.edu.cn (R.Y.); zhaonanli2022@163.com (Z.L.)
[2] School of Internet of Things Engineering, Jiangnan University, Wuxi 214000, China; qjhao@jiangnan.edu.cn
* Correspondence: zhli@jiangnan.edu.cn

Abstract: In cloud data centers, determining how to balance the interests of the user and the cloud service provider is a challenging issue. In this study, a task-loading-oriented virtual machine (VM) optimization placement model and algorithm is proposed integrating consideration of both VM placement and the user's computing requirements. First, the VM placement is modeled as a multi-objective optimization problem to minimize the makespan of the loading tasks, user rental costs, and energy consumption of cloud data centers; then, an improved chaos-elite NSGA-III (CE-NSGAIII) algorithm is presented by casting the logistic mapping-based population initialization (LMPI) and the elite-guided algorithm in NSGA-III; finally, the presented CE-NSGAIII is employed to solve the aforementioned optimization model, and further, through combination of the above sub-algorithms, a CE-NSGAIII-based VM placement method is developed. The experiment results show that the Pareto solution set obtained using the CE-NSGAIII exhibits better convergence and diversity than those of the compared algorithms and yields an optimized VM placement scheme with shorter makespan, less user rental costs, and lower energy consumption.

Keywords: cloud computing; cloud data center; VM placement; task scheduling; multi-objective optimization

Academic Editor: Gianluigi Ferrari

Received: 21 December 2024
Revised: 22 January 2025
Accepted: 26 January 2025
Published: 7 February 2025

Citation: Yang, R.; Li, Z.; Qian, J.; Li, Z. Task-Driven Virtual Machine Optimization Placement Model and Algorithm. *Future Internet* **2025**, *17*, 73. https://doi.org/10.3390/fi17020073

1. Introduction

Numerous computing, storage, and network hardware resources are housed in cloud data centers. Virtualization technology is utilized to abstract these physical resources into virtual resources that can be allocated on demand. The scheduling and management of resources in cloud data centers are aimed at optimizing and automatically scheduling these virtual resources to support various cloud services and computing requests. The ability to quickly respond to user computing requirements and efficiently allocate virtual resources is crucial in balancing the interests of cloud service providers and users, who mutually impact and constrain each other. The computing requests of users are assigned to different virtual machines (VMs) through resource scheduling. Appropriately scheduling tasks can reduce task execution times and cloud service response times and improve user experience. VM placement involves deploying VMs to destination physical machines (PMs) within the constraints of available resources. Further, it can effectively reduce the energy consumption and operational costs for cloud service providers if employed appropriately. In general, task scheduling and VM placement in cloud data centers can be modeled as bin-packing problems [1,2], which are NP-hard problems [3]. The expansion of cloud data centers poses considerable challenges in terms of efficient task loading and VM placement.

Multiple algorithms or schemes for task scheduling and VM placement have been proposed; these can generally be categorized into three types: models and methods for task scheduling, models and methods for VM placement, and joint models and methods for task scheduling and VM placement. Task scheduling strategies have primarily been investigated by employing intelligent algorithms such as ant colony optimization [4], particle swarm optimization (PSO) [5], and genetic algorithms [6] to build fine-grained scheduling plans and optimize multiple objectives. However, in these situations, the physical resources in cloud data centers were assumed to be unlimited, and the mapping between VMs and PMs was neglected in the above studies. Consequently, an ineffective response to the random fluctuations in cloud data center resource demands was achieved. VM placement strategies and schemes were primarily explored in [7–14], and heuristic or meta-heuristic algorithms were employed in [7,8] to solve VM placement optimization problems. However, most of these algorithms optimize only a single objective, and thus, they are unsuitable for the stochastic computational demands of multiple users and tasks in real cloud data centers. In [9–14], attempts were made to overcome the limitations of single-objective optimization by developing multi-objective optimization algorithms that consider the constraints among multiple objectives. Although VM placement strategies were mainly evaluated in these studies, the impact of task loading on virtual resource consumption, which is a crucial aspect of VM placement, was not covered. VM placement schemes in which task loading is not considered cannot fully adapt to the multi-user, multi-task, and dynamically changing environment of real cloud data centers.

The loading of user computing tasks directly impacts virtual resource consumption and further affects virtual resource utilization and energy consumption. Therefore, an appropriate task-loading-oriented VM placement model, algorithm, or scheme can efficiently reduce the makespan, control user rental costs, and lower the energy consumption of cloud data centers. Thus, the processes of task scheduling and VM placement must be considered together. In [15–17], the mutual influence of task scheduling and VM placement was considered in the study of a VM placement optimization method oriented toward task scheduling. However, in [16], the focus was only on minimizing the energy consumption of a cloud data center under virtual resource constraints, and factors such as the VM rental costs and the makespan of user tasks, which significantly impact user experience, were ignored. Alboaneen et al. [15] considered both VM rental costs and the resource utilization of the cloud data center; however, the optimization model was excessively simplified by converting the multi-objective optimization problem into a single-objective one through weighted sums. This resulted in the loss of some valuable information from the original multi-objective problem. Consequently, several Pareto optimal solutions may be overlooked, and practical and meaningful optimization solutions may be missed. To balance the interests of cloud service providers and users, optimization of the makespan, rental costs, and energy consumption must be performed simultaneously. In this study, task scheduling and VM placement are integrated to investigate the task-loading-oriented VM placement problem (TLVMP). Real-world resource limitations and constraints are incorporated in cloud data centers to reduce makespan, user rental costs, and energy consumption. To realize these three objectives, the VM placement problem is modeled as a multi-objective optimization issue. Moreover, as the non-dominated sorting genetic algorithm III (NSGA-III) can solve multi-objective optimization problems with considerable flexibility and provide a reasonable solution to TLVMP problems, this improved algorithm is adapted to solve the VM placement models. Based on the improved chaos-elite NSGA-III (CE-NSGAIII), the chaos-elite VM placement (CE-VMP) method is further proposed. The main contributions are as follows:

(1) To reduce the makespan, user rental costs, and energy consumption of cloud data center, the task-loading-oriented VM placement issue is abstracted as a multi-objective optimization model.

(2) An improved CE-NSGAIII algorithm is presented depending on the NSGA-III to solve the aforementioned model and improve the convergence and diversity of the Pareto solution set.

(3) Summarizing and concluding the above solution processes, a CE-NSGAIII-based method named the CE-VMP is further proposed for VM placement.

(4) The experimental results show that the CE-NSGAIII can efficiently solve the TLVMP problem. Also, the CE-VMP method is capable of achieving better results in terms of makespan, user rental costs, and energy consumption.

2. Related Works

In the context of resource management in cloud data centers, VM placement plays a key role in balancing the interests of both cloud service providers and their users. This section first introduces some VM placement methods, then describes some multi-objective optimization algorithms that can be used to solve the VM placement problem in cloud data centers, and finally describes some techniques employed in the proposed method of this paper.

2.1. VM Placement Method

First-fit decreasing VM placement (FFD-VMP) [8] is generally used as a benchmark method for evaluating VM placement; a first-fit heuristic algorithm is employed to place VMs. PABFD-VM placement (PABFD-VMP) [7] involves the use of an adaptive heuristic algorithm to adjust VM placement dynamically. COFFGA-VM placement (COFFGA-VMP) [9] requires using a genetic algorithm to address combinatorial optimization problems with varying capacity constraints to determine the optimal allocation of VMs to PMs. Duan et al. [10] proposed an enhanced energy-efficient ant colony optimization strategy (EEACS) for VM placement, which prioritizes PMs by energy efficiency and improves server selection and pheromone update rules. This approach ensures efficient VM placement using pheromones and heuristic information. Karmakar et al. [18] proposed a multi-objective algorithm based on ant colony optimization, which improved the service quality and resource utilization of data centers by optimizing multiple objectives such as resource allocation, energy consumption, and network latency. Nagadevi et al. [19] improved the VM placement process by introducing a multicore-aware VM placement policy combined with constraint programming techniques. This approach aims to improve the resource utilization, system performance, and scalability of cloud data centers. Torre et al. [20] proposed a dynamic evolutionary multi-objective heuristic algorithm for VM placement; it dynamically adapted to changing resource demands while optimizing energy efficiency, reducing network latency, and guaranteeing service quality. However, in these studies, VM placement was examined without considering the impact of task loading; thus, dynamically adjusting to actual workloads failed. Consequently, the makespan could not be effectively optimized, ultimately impacting cloud data center performance and user experience.

Alboaneen et al. [15] considered both task scheduling and VM placement to develop a multi-objective optimization framework that optimized both VM rental costs and resource utilization in cloud data centers. Meta-heuristic algorithms were employed to solve this joint optimization problem to achieve a comprehensive balance between multiple optimization objectives. However, the use of a weighted sum method to transform a multi-objective optimization problem into a single-objective one lacks flexibility and may result in the omission of many valuable solutions. To address resource management and optimization issues

in cloud data centers, a decomposition-based multi-objective evolutionary algorithm for optimizing both VM placement and task scheduling was developed by considering three objectives: minimizing the completion time, costs, and total tardiness [17]. Although the focus was only on minimizing energy consumption, Liu et al. [16] introduced an integrated optimization framework for task scheduling and VM placement. This resulted in a more comprehensive optimization method when compared with existing research. Inspired by previous studies [15–17], this work improved NSGA-III to explore VM placement schemes tailored to user task loading, aiming to balance the interests of both cloud service providers and users.

2.2. Multi-Objective Optimization Algorithms

Multi-objective optimization algorithms can determine an optimal solution set in complex and dynamically changing environments by simultaneously pursuing multiple potentially conflicting goals. Thus, they also can yield efficient solutions to multi-objective resource management issues in cloud data centers.

Sierra et al. [21] proposed an improved multi-objective optimization algorithm based on PSO by integrating crowding strategies, mutation operations, and the ε-dominance concept. This enhanced the ability of the algorithm to solve complex problems such as multi-objective task scheduling and resource allocation in cloud data centers. Nebro et al. [22] introduced a multi-objective optimization mechanism to effectively handle complex scenarios, such as multi-objective task scheduling and resource allocation in cloud data centers, wherein multiple conflicting objectives exist. Deb et al. [23] developed an evolutionary multi-objective optimization algorithm based on a reference point non-dominated sorting method, aimed at solving optimization problems with numerous conflicting objectives. Liu et al. [24] introduced a dynamic adaptation mechanism in an evolutionary algorithm for solving multi-objective optimization problems in dynamic environments. This algorithm can cope with dynamic problems in which the objective function and constraints change over time, ensuring that the solution set is continuously optimized and the diversity of the solution set is maintained as the environment changes. Zhang et al. [25] introduced a multi-objective evolutionary algorithm based on a fuzzy logic adaptive selection operator. This algorithm was designed to handle complex optimization problems with multiple competing objectives by incorporating a fuzzy logic system that dynamically adjusted operators such as crossover and mutation during the evolutionary process. Santiago et al. [26] proposed a fuzzy adaptive NSGA-III to handle large-scale optimization problems. This algorithm incorporated a fuzzy logic system to adaptively adjust parameters and strategies; and thus, the complexity and uncertainty typical of large-scale problems were better managed. In summary, multi-objective optimization algorithms are well-suited for handling scenarios with multiple conflicting objectives, providing robust solutions that effectively balance competing demands in complex cloud computing environments to guarantee efficient and optimal performance.

2.3. Related Techniques

Chaotic maps are mathematical functions that generate a highly arbitrary pattern based on the initial seed value [27]. Varol Altay et al. [28] incorporated chaotic mapping into the Bird Swarm Algorithm (BSA), using chaotic sequences to adjust the search process dynamically, replacing linear updates with complex nonlinear changes to enhance global search capability and avoid falling into local optima. Fan et al. [29] integrated tent chaotic mapping into the African Vulture Optimization Algorithm (AVOA), where chaotic sequences in the early stages help explore the entire search space, preventing premature convergence to local optima. The application of chaotic mapping in multi-objective op-

timization algorithms enhances coverage and global exploration in the search space by generating initial populations or optimizing random sequences along the search path.

In general, optimization problems in discrete spaces struggle to converge to the global optimum, integrating elite solutions to guide population evolution is an effective way to enhance global search capability [30]. Zhang et al. [31] improved the particle swarm algorithm by employing multi-elite particles, enhancing global search capability and accelerating convergence, and applied it to multi-reservoir system scheduling. Kong et al. [32] enhanced the artificial bee colony (ABC) algorithm with an elite guidance mechanism, enabling a more effective search towards potential global optima and improving global search performance in complex problems. The elite guidance strategy enables the algorithm to rapidly converge to the Pareto optimal frontier by selecting the best individuals or subsets to provide reference directions for others during the optimization process.

In summary, chaotic mapping and elite guidance techniques play a crucial role in enhancing the diversity and global search capability of multi-objective optimization algorithms. Therefore, this paper also employs these techniques to improve NSGA-III.

3. System Model and Problem Description

3.1. Cloud Data Center Assumptions

As illustrated in Figure 1, a logical relationship exists between task loading and virtual resource scheduling in a cloud data center, which is composed of three layers: user layer, broker layer, and resource allocation layer. In the user layer, tasks are submitted to the cloud data center, where they are temporarily stored in a task queue. In the broker layer, the task queue passes to the task scheduler, which then assigns the tasks to VMs. Based on the requirements of the tasks, the PMs instantiate a group of heterogeneous VMs. After a scheduling computation has been performed between the loaded tasks and the VMs, the tasks are pushed to the corresponding VM ready queue and loaded onto these VMs. The resource allocation layer comprises heterogeneous PMs that utilize technologies such as hypervisors to create multiple VMs. Technologies such as VM migration or VM consolidation support completing VM placement to the appropriate destination PMs. Consequently, the VM placement results are sent to the broker layer, which returns the final results of task execution to the user.

In this study, an Infrastructure-as-a-Service (IaaS) cloud platform is assumed to provide pay-as-you-go storage, network services, and virtualization services. In addition, the data center is assumed to contain N PMs, represented as set $H = \{h_1, h_2, ..., h_k, ..., h_N\}$. Further, the tasks submitted by users are denoted as set $T = \{t_1, t_2, ..., t_i, ..., t_L\}$ with the tasks being independent of each other. On the N PMs, M heterogeneous VMs are created, with the VMs indicated as set $V = \{v_1, v_2, ..., v_j, ..., v_M\}$. The symbols used in this paper and their corresponding meanings are listed in Table 1.

Table 1. Definitions of Parameters.

Parameter	Definition	Parameter	Definition
i	Index for task	X	Mapping relationships among tasks, VMs, and PMs
j	Index for VMs	C_j^{tran}	Transmission cost of bandwidth of v_j
k	Index for PMs	C_k^{rent}	Rental cost for the physical resources consumed by v_j placement on h_k
l_i	Length of t_i	U_k	Resource utilization rate of h_k

Table 1. *Cont.*

Parameter	Definition	Parameter	Definition
s_{fi}	Input file size of t_i	VT_j	Total active time of v_j
s_{oi}	Output file size of t_i	HT_k	Total active time of h_k
r	Resource type	P_k^{static}	Static energy consumption of PM
v_j	Resource capacity of v_j	$P_k^{dynamic}$	Dynamic energy consumption of PM
C_j	Rental prices for resources of v_j	$x_{i,j}$	Binary variable that takes value 1 if t_i is loaded onto v_j, 0 otherwise
$C(v_j)$	Processing power of v_j	$y_{j,k}$	Binary variable that takes value 1 if v_j is deployed on h_k, 0 otherwise
c_k	Rental price of h_k	F	Mutation probability
$E_{i,j}$	Estimated completion time of t_i processed on v_j	CR	Crossover probability
C_j^{exec}	Computing resource rental costs of v_j	pe_j	Number of CPU cores on v_j

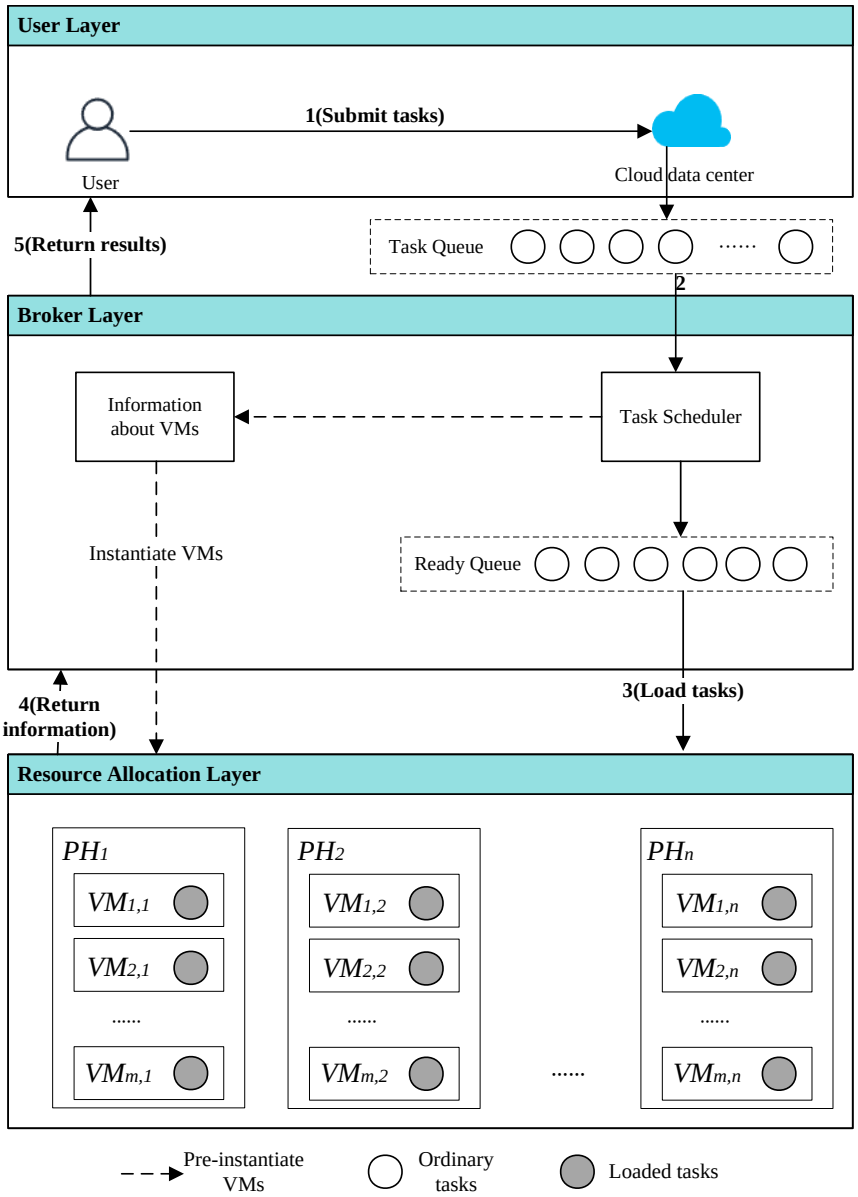

Figure 1. The logical relationship between task loading and virtual resource scheduling.

3.2. Makespan Estimation

In general, the shorter the makespan, the better the user experience, indicating that the task scheduling and resource allocation algorithms of the system are more efficient. Therefore, the makespan is an indirect indicator of the rationality of resource allocation. If $E_{i,j}$ is assumed to be the estimated task completion time for task t_i to be submitted to VM v_j, then it can be calculated as Equation (1):

$$E_{i,j} = \frac{l_i}{C(v_j)} \tag{1}$$

where l_i represents the length of t_i defined as the total computational workload required to complete task t_i, usually in Million Instructions (MI). In practice, l_i can be determined based on the specifications of the task when users submit tasks or estimated through the analysis of similar tasks. In this study, the l_i is determined from the dataset based on the specifications of the task. $C(v_j)$ indicates the processing power of v_j calculated by Equation (2):

$$C(v_j) = v_j^{cpu} pe_j \tag{2}$$

where v_j^{cpu} represents the CPU processing capacity of v_j, measured in millions of instructions per second and pe_j represents the number of CPU cores on v_j. Thus, the maximum completion time for all tasks, or the makespan, can be computed using Equation (3):

$$\text{makespan} = \max\left\{\sum_{i=1}^{L} x_{i,j} E_{i,j}\right\} \tag{3}$$

where $x_{i,j} = 1$ indicates t_i is loaded onto v_j, $x_{i,j} = 0$ otherwise.

3.3. Estimation of User Rental Costs

The allocated CPU resources of a VM are closely related to its ability to execute tasks efficiently. The greater the CPU resources available, the quicker a task can be completed, thereby reducing the actual execution time. However, for cloud service providers, a higher CPU performance inevitably leads to higher operational costs owing to the increased power consumption and cooling requirements. This results in cloud service providers charging a higher rental fee. Thus, to balance the efficiency of the cloud data center with the interests of users, this study also examined user rental costs.

Inspired by [17], the execution costs for user tasks mainly include: the rental cost of VM computing resources, transmission cost of bandwidth, and rental cost of physical resources consumed by the running VM.

The rental cost of VM computing resources per unit of time can be estimated using Equation (4):

$$C_j^{exec} = \left(v_j^{cpu} + v_j^{mem}\right) \times \left(c_j^{cpu} + c_j^{mem}\right) \tag{4}$$

where v_j^{mem} represents the memory capacity of v_j, c_j^{cpu} denotes the rental prices for CPU of v_j and c_j^{mem} indicates the rental prices for memory of v_j.

The transmission cost of bandwidth can be estimated using Equation (5):

$$C_j^{tran} = \sum_{j=1}^{M} \left(\frac{\sum_{i=1}^{L} x_{i,j}(sf_i + so_i)}{v_j^{bw}} \times c_j^{bw} \right) \tag{5}$$

where $x_{i,j}$ is the same as that in Equation (3), sf_i and so_i represents the input and output file size of t_i, respectively. v_j^{bw} represents the memory capacity of v_j and c_j^{bw} expresses the rental prices for bandwidth of v_j.

The rental cost for the physical resources consumed by the running VM on PM is estimated using Equation (6):

$$C_k^{rent} = y_{j,k} c_k \tag{6}$$

where $y_{j,k} = 1$ indicates v_j is deployed on h_k, $y_{j,k} = 0$ otherwise. c_k represents the rental price of h_k. Therefore, the user rental costs can be estimated using Equation (7):

$$\begin{aligned} &\text{cost} \\ &= \sum_{i=1}^{L} \sum_{j=1}^{M} \left(x_{i,j} E_{i,j} C_j^{exec} \right) + \sum_{i=1}^{L} \sum_{j=1}^{M} C_j^{tran} + \sum_{k=1}^{N} C_k^{rent} \\ &= \sum_{i=1}^{L} \sum_{j=1}^{M} \left(x_{i,j} E_{i,j} C_j^{exec} + C_j^{tran} \right) + \sum_{k=1}^{N} C_k^{rent} \end{aligned} \tag{7}$$

3.4. Energy Consumption Estimation

Let U_k be the resource utilization rate for PM h_k, U_k can be calculated [33] using Equation (8):

$$U_K = \frac{\sum_{j=1}^{M_k} VT_j}{M_k \times HT_k} \tag{8}$$

where VT_j represents the total active time of a VM in a PM, M_k denotes the number of VMs in a PM and HT_k indicates the total active time of a PM.

The energy consumption of h_k is composed of static and dynamic energy consumption [34]. Assume that P_k^{static} represents the static energy consumption when the PM is idle, which is generally set as 50% of the total energy consumption of the PM [34]. The dynamic energy consumption of the PM during runtime, denoted as $P_k^{dynamic}$, is modeled using a nonlinear energy consumption model [34], as defined in Equation (9):

$$P_k^{dynamic} = \omega_1 \cdot U_k + \omega_2 \cdot U_k^2 \tag{9}$$

where ω_1 and ω_2 are nonlinear model parameters calculated based on the resource utilization of the PM. Therefore, the total energy consumption (EC) of all PMs in the cloud data center can be estimated using Equation (10):

$$\text{EC} = \sum_{k=1}^{N} \int_0^t \left(P_k^{static} + P_k^{dynamic} \right) dt \tag{10}$$

3.5. Problem Definition

In this study, task loading and VM placement are integrated to balance the interests of both cloud service providers and users. The optimization objectives included minimizing makespan, minimizing user rental costs, and minimizing energy consumption in the cloud data center. The VM placement problem is thus modeled as a multi-objective optimization problem, as defined in Equation (11).

$$\arg\min_X F(X) = (f_1(X), f_2(X), f_3(X))^T \tag{11}$$

$$f_1(X) = \text{makespan} \tag{12}$$

$$f_2(X) = \text{cost} \tag{13}$$

$$f_3(X) = \text{EC} \tag{14}$$

s.t.

$$\text{C1} : \sum_{j=1}^{M} x_{i,j} = 1, i = 1, 2, ..., L \tag{15a}$$

$$C2: \sum_{i=1}^{L} d_i^r \cdot x_{i,j} < C_j^r, j = 1, 2, ..., M \quad (15b)$$

$$C3: \sum_{k=1}^{N} y_{j,k} = 1, j = 1, 2, ..., M \quad (15c)$$

$$C4: \sum_{j=1}^{M} d_j^r \cdot y_{j,k} < C_k^r, k = 1, 2, ..., N \quad (15d)$$

Here, X represents the mapping relationships among tasks, VMs, and PMs, that is, the correspondence between tasks loading onto VMs and VMs deployment onto PMs, thereby forming the task-loading-oriented VM placement strategies and schemes. Further, $r \in \mathrm{RS}$, $\mathrm{RS} = \{\mathrm{CPU}, \mathrm{MEM}, \mathrm{BW}\}$, represents the set of CPU, memory, and bandwidth resources for PMs or VMs.

Equation (11) represents a multi-objective optimization issue. Constraint C1 states that a task can only be loaded onto one VM. Whereas, Constraint C2 specifies that the total requested resources of type r for t_i on v_j cannot exceed the total configured capacity of resource type r on that VM. Further, Constraint C3 indicates that a VM can only be placed on one PM. Finally, Constraint C4 stipulates that the total requested resources of type r for all VMs on h_k cannot exceed the total configured capacity of resource type r on that PM.

4. VM Placement Optimization Method

Since the processed data in VM placement optimization are discrete [11], thus the NSGA-III [23] is improved in this study to solve the VM placement optimization model defined in Equation (11) and obtain the optimal mapping among tasks, VMs, and PMs, that is, the VM placement scheme X. First, a three-layer encoding scheme is designed for the TLVMP problem: the top layer represented tasks, the middle layer represented VMs, and the bottom layer represented PMs. In this context, each encoding scheme corresponded to a VM placement solution oriented toward task loading. In the context of genetic algorithms, a specific VM placement solution is analogous to an individual in a population, which is a collection of all possible mappings between tasks, VMs, and PMs. Next, chaotic mapping is used to initialize the population, and the crossover and mutation operations in the differential evolution (DE) algorithm [35] are improved to enhance the algorithm's search capability and expand its search space. Further, to balance the convergence and diversity of the Pareto front, a penalty-based boundary intersection (PBI) distance is adopted during environmental selection to integrate the elite solution algorithm and guide population evolution. This enhanced the global search capability of the algorithm. Finally, the improvements to NSGA-III are integrated to propose a new hybrid algorithm combining chaotic mapping and elite guidance (CE-NSGAIII). The CE-NSGAIII is then employed to solve the optimization model defined in Equation (11). The flowchart of the CE-NSGAIII is shown in Figure 2.

4.1. Chaos-Elite NSGA-III

4.1.1. Encoding Method

The VM placement optimization scheme X comprises a three-layer string: the first layer representes the task indexes, the second layer indicates the VM indexes that the tasks are loaded onto, and the third layer denotes the PM indexes where the VMs are placed. Figure 3 provides an example of a task-driven VM placement scheme, where eight tasks are randomly assigned to four VMs, and the four VMs are deployed on three PMs. For example, t_7 is loaded onto v_1, and v_1 is placed on h_2.

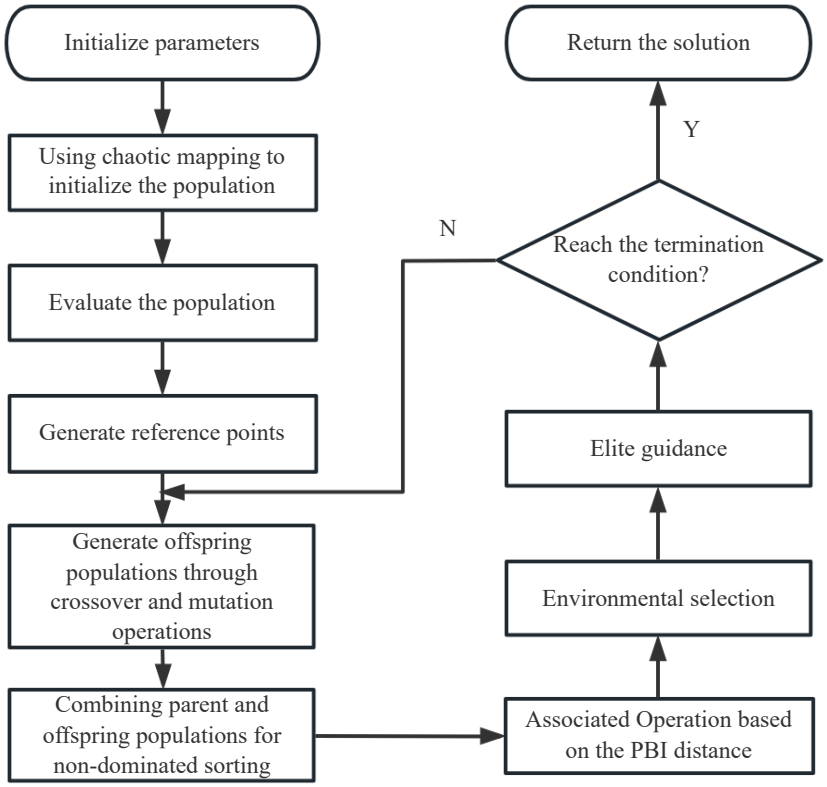

Figure 2. Flowchart of CE-NSGAIII.

task	t_4	t_1	t_2	t_7	t_8	t_3	t_6	t_5
VM	v_1	v_3	v_2	v_1	v_1	v_4	v_2	v_4
PM	h_1	h_1	h_3	h_2	h_1	h_3	h_1	h_2

Figure 3. The example of a VM placement scheme.

4.1.2. Population Initialization Based on Chaotic Mapping

Chaotic mapping is a dynamic system that is highly sensitive to initial conditions. Even slight changes in these conditions can result in significant differences in system behavior. This characteristic is beneficial for genetic algorithms as it enhances the diversity of individuals and ensures a broad coverage of the search space, thereby improving the algorithm's search efficiency and global search capability. It also aids in avoiding premature convergence to local optima and promotes the exploration of a wider solution space. In the traditional NSGA-III [23], the initial population is randomly generated, which, when handling large-scale tasks, is not conducive to determining the true optimal solution such as VM placement. Therefore, chaotic mapping is employed in NSGA-III to initialize the population. The process of chaos-mapping-based population initialization is as follows:

Step 1st: Generating a chaotic sequence

A chaotic sequence is generated using Equation (16).

$$x_{n+1} = \mu x_n (1 - x_n) \tag{16}$$

where μ determines the dynamic behavior of the system, including the chaotic region. x_n is the current state and x_{n+1} is the next state.

Step 2nd: Mapping chaotic values to VM assignments

The mapping of the loaded tasks to VMs is one of the components of individual. Each task index is assigned a chaotic value from the sequence. This chaotic value is mapped to a bounded integer within a specified range defined by $lowerbound$ and $upperbound$, which represents the range for possible VM allocations for each task. The mapped chaotic value determines the VM index for each task in $individual.setVariables(j, val)$, ensuring diverse task-to-VM allocations.

Step 3rd: Mapping chaotic values to PM assignments

The mapping of VMs to PMs is another component of the individual. In this process, each VM index is assigned a chaotic value from the sequence. This chaotic value is mapped to a bounded integer within the specified range defined by $Plowerbound$ and $Pupperbound$, which represents the range of possible PM assignments. These mapped chaotic values are stored in $individual.setVariablesP(k, Pval)$, which represents the PM index where each VM resides.

Step 4th: Adding individuals to the population

After each individual is initialized, the chaosIndex is updated to keep track of the position in the chaotic sequence, ensuring that each individual receives a unique sequence of chaotic values. Instances of individuals that have completed their initialization are then added to the population.

Finally, the population initialization is completed once all individuals are mapped. The logistic mapping-based population initialization algorithm (LMPI) is described in the pseudocode of Algorithm 1.

Algorithm 1: LMPI

Input: The length of sequence L, initial value of chaotic $CHAOS\_PARAM$, the number of tasks $taskNum$, the number of VMs $vmNum$

Output: Initialized population P

for $i = 0$ to $L - 1$ **do**
  Use Equation (16) to compute the new chaos state value x
  $S[i] = x$ `// Put chaotic values into chaotic sequence`
end
for *each individual i from 0 to* $popSize - 1$ **do**
  for $j = 0$ to $taskNum - 1$ **do**
    Retrieve $chaoticVal$ from S at index $chaosIndex + innerLoopCounter$
    Map $chaoticVal$ to val within the bounds:
    $val = (lowerbound + upperbound - lowerbound) * chaoticVal$
    $individual.setVariables(j, val)$ `// Assign val to the jth variable of individual`
  end
  for $k = 0$ to $vmNum - 1$ **do**
    Retrieve $PchaoticVal$ from S at index $chaosIndex + innerLoopCounter$
    Map $PchaoticVal$ to $Pval$ within the bounds:
    $Pval = (Plowerbound + Pupperbound - Plowerbound) * PchaoticVal$
    $individual.setVariables(k, Pval)$ `// Assign Pval to the kth variable of individual`
  end
  Update $chaosIndex$ by adding $innerLoopCounter$
  Add $individual$ to P
end
return P

The time cost of Algorithm 1 mainly originates from lines 2–12, which involve iterating over the individuals in the population and mapping the chaotic sequence. If the size of the population is assumed to be N and the number of the task is T, then the time complexity of Algorithm 1 is $O(NT)$.

4.1.3. Adaptive Crossover and Mutation Operations

Exploration is a process of accessing a new region in the search space, which facilitates the algorithm to search the region containing dominant solutions in the global scope. Exploitation involves digging deeper into the neighborhood of the previously visited points, which is conducive to the improvement of the algorithm search results [36]. Higher crossover and mutation probabilities can enhance exploration by generating more diverse solutions and covering a broader search space. Lower crossover and mutation probabilities improve exploitation by fine-tuning solutions in promising areas [37]. Leveraging a balance between exploration and exploitation, with properly tuned crossover and mutation probabilities, enables the algorithm to avoid falling into local optima and ensures both global search capability and local refinement. Inspired by [38], the mutation probability, F, and crossover probability, CR, in the DE [35] are adaptively adjusted, and these adjustments are incorporated into NSGA-III for improvement. The processes for adjusting the mutation and crossover probabilities are expressed as Equation (17) and Equation (18), respectively.

$$F = \begin{cases} \frac{k_1(f_{\max} - f_{crossover})}{f_{\max} - f_{avg}}, f_{crossover} \geq f_{avg} \\ k_2, f_{crossover} < f_{avg} \end{cases} \tag{17}$$

$$CR = \begin{cases} \frac{k_3(f_{\max} - f_{mutation})}{f_{\max} - f_{avg}}, f_{mutation} \geq f_{avg} \\ k_4, f_{mutation} < f_{avg} \end{cases} \tag{18}$$

where $f_{\max}$ denotes the current maximum fitness value of the population, f_{avg} indicates the current average fitness value of the population, $f_{\text{crossover}}$ denotes the larger fitness value between the two crossover individuals, and f_{mutation} expresses the fitness value of the mutant individual. Based on multiple experimental observations, the constants in Equations (17) and (18) are empirically set as follows: $k_1 = 1.0, k_2 = 1.0, k_3 = 0.5, k_4 = 0.5$.

In traditional NSGA-III, polynomial mutation and simulated binary crossover are typically used during mutation and crossover operations. This approach easily lets the algorithm fall into local optimization. To quickly propagate beneficial traits and accelerate convergence while maintaining diversity in the solutions, the adaptive adjustments to mutation and crossover operations shown in Equations (17) and (18) are integrated with NSGA-III, and the polynomial mutation and simulated binary crossover are replaced with the improved operations. The mutation operation is calculated as shown in Equation (19).

$$v_i = x_{r_1} + F(x_{r_2} - x_{r_3}) \tag{19}$$

where v_i represents the mutant individual, $x_{r_1}, x_{r_2}, x_{r_3}$, are three different individuals randomly selected from the current population, F is the probability of variation shown in Equation (17). After the mutation operation, the crossover operation is performed to mix the mutant individual v_i and the original individual x_i to generate the test individual u_i, as shown in Equation (20).

$$u_i = \begin{cases} v_i^j, \text{if } rand_j \leq CR \text{ or } \mathrm{j} = \mathrm{j}_{\text{rand}} \\ x_i^j, \text{otherwise} \end{cases} \tag{20}$$

where v_i^j denotes the jth component of v_i, x_i^j represents the jth component of x_i. $rand_j$ is a random number uniformly distributed within $[0, 1]$ and CR is the crossover probability shown in Equation (18). j_{rand} is an integer randomly chosen within $[1, D]$, where D is the dimensionality of the problem, and it is guaranteed that at least one component comes from the variational vector.

4.1.4. PBI Distance

To balance the convergence and diversity of the population, a PBI distance [39] is used to define the distance between the population and the reference vectors. Let $f^n(y) = (f_1^n(y), f_2^n(y), \ldots, f_M^n(y))^T$ be the normalized vector of an individual's objective values. As shown in Figure 4, $d_{j,n}(y)$ denotes the projection distance from $f^n(y)$ to the jth reference line for assessing the convergence of the population and $d_{j,2}(y)$ indicates the vertical distance from $f^n(y)$ to the jth reference line for assessing the diversity of the population. $d_{j,1}(y)$ and $d_{j,2}(y)$ are calculated according to Equation (21) and Equation (22), respectively.

$$d_{j,1}(y) = \left\| f^n(y)^T w_j \right\| \Big/ \|w_j\| \tag{21}$$

$$d_{j,2}(y) = \|f^n(y) - d_{j,1}(y)(w_j/\|w_j\|)\| \tag{22}$$

where w_j refers to the direction of the axis of the objective function.

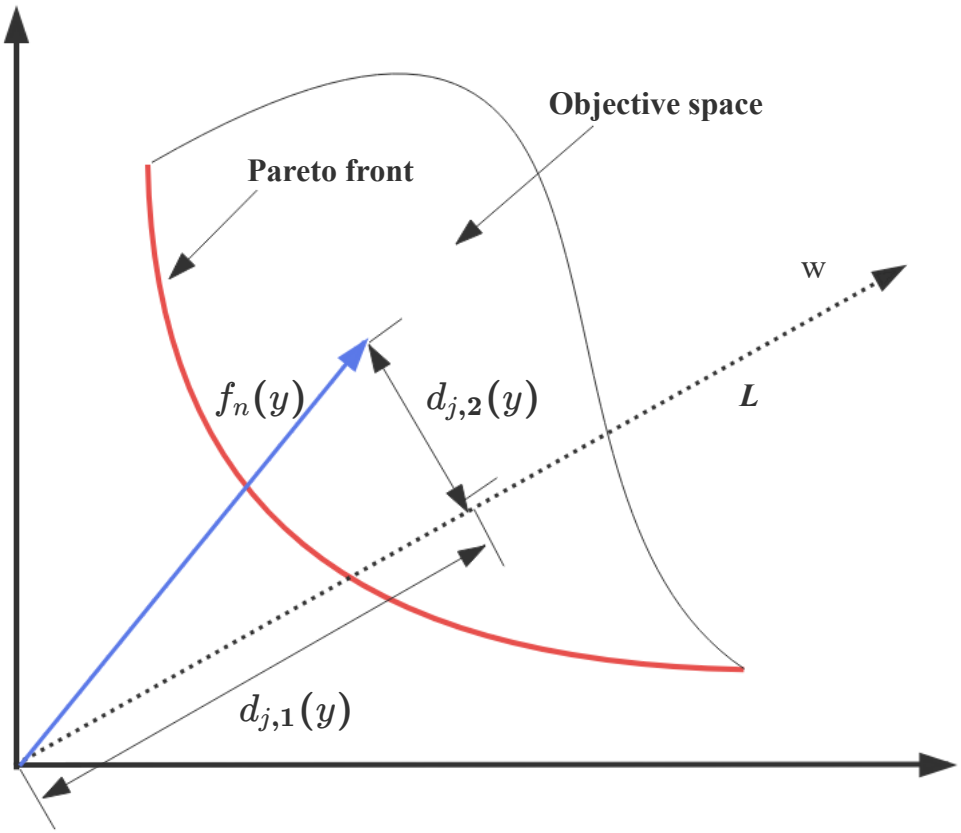

Figure 4. The schematic of distance $d_{j,1}(y)$ and $d_{j,2}(y)$.

4.1.5. Elite Guidance

To accelerate its convergence to a global optimum of the algorithm, integrating elite solutions to guide population evolution can enhance the algorithm's global search capability [30]. Therefore, an elite-guided algorithm is proposed to guide the population toward the Pareto optimal solutions. This ensured that the algorithm continuously explored new

non-dominated solutions while maintaining the discovered high-quality solution set and accelerating convergence.

The process to guide the population evolution toward the Pareto front is as follows. First, randomly select individuals from the generated set of all the elite individuals, and add them to the leader set. Next, generate a random component index set, and extract individuals t_1 and t_2 from the current population and the leader set, respectively. Finally, replace the components of t_2 at the current index positions with the corresponding components of t_1, place the modified individual t_1 back into the population, and return the updated population. Here, the population corresponds to the set of possible VM placement optimization schemes generated during the genetic evolution process. As a result, summarized the above process, the elite-guided algorithm is proposed, which is depicted as Algorithm 2.

Algorithm 2: Elite-guided algorithm

Input: Initialized population P of size N, Archive A of elite individuals
Output: The updated population P'

1 Initialize an empty list L for leaders
2 Instantiate a random number generator RandomGen
3 **for** $i = 0$ *to* $N - 1$ **do**
4 Temp $\leftarrow$ RandomGen.nextInt(0, $|A| - 1$)
5 L.append(A[Temp]) `// Randomly select individuals from Archive`
6 **end**
7 Set crossover probability $\rho = 0.05$
8 Calculate crossover length $cLen = \lceil \rho \times d \rceil$, where d is the dimensionality
9 **for** $i = 0$ *to* $N - 1$ **do**
10 Generate $cLen$ distinct random indices R from $[0, d - 1]$
11 $t_1 \leftarrow P[i]$ `// Take` t_1 `from the current population`
12 $t_2 \leftarrow L[i]$ `// Take` t_2 `from the leader set`
13 **for** *each index* $\in R$ **do**
14 t_1.variables[$index$] $\leftarrow$ t_2.variables[$index$]
`// Replace the variable at index of` t_1 `with the corresponding index of` t_2
15 $P[i] \leftarrow t_1$ `// Put` t_1 `back into the population`
16 **end**
17 **end**
18 **return** P' `// Return the updated population` P'

The time cost of Algorithm 2 mainly yields from lines 11–18, which involve iterating over the individuals in the population and replacing components at index positions. If the number of individuals in the population is assumed to be N and the length of the index set is L, then the time complexity of Algorithm 2 is $O(NL)$.

4.1.6. CE-NSGAIII

Based on the above, through casting the developed chaotic-mapping-based population initialization (e.g., Algorithm 1), adaptive crossover and mutation operations and elite guidance (e.g., Algorithm 2) into the classical NSGA-III, a new Chaos-Elite NSGA-III (CE-NSGAIII) is proposed.

The CE-NSGAIII first performs initialization; then, the initialized population is subjected to crossover and mutation operations according to Equations (19) and (20), respectively, to generate an offspring population. The initialized population and the offspring population

are merged for non-dominated sorting, and each individual is associated according to Equations (21) and (22). Thereafter, environmental selection is performed on the sorted population, followed by elite guidance on the resulting population. Finally, the iterated population is returned, representing the set of Pareto solutions. Correspondingly, as for the issue of VM placement, the yielded Pareto solution set represents all optimal non-dominated "tasks-VMs-PMs" maps.

The time cost of Algorithm 3 mainly originates from lines 8–19, which is the time consumed by iteratively updating the population for selection, crossover, and mutation, as well as calling Algorithm 2 at line 18. Let the number of iterations be K, then the time complexity of Algorithm 3 is $O(KNL)$.

Algorithm 3: CE-NSGAIII

Input: The length of sequence L, initial value of chaotic parameter *CHAOS_PARAM*, the number of tasks $taskNum$, the number of VMs $vmNum$

Output: The final population P' // Namely the "tasks-VMs-PMs" maps

1 Initialize iterations *Iters*, population size *popSize*
2 $P \leftarrow$ LMPI(L, *CHAOS_PARAM*) // Call Algorithm 1 to initialize the population
3 Evaluate and calculate the fitness of P
4 Generate the reference points
5 Generate the elite archive A of P
6 **for** $i = 0$ *to Iters* $- 1$ **do**
7 $\quad$ *offP* $\leftarrow$ TournamentSelection(P)
8 $\quad$ **for** $j = 0$ *to popSize* $- 1$ **do**
9 $\quad\quad$ Use Equation (19) to perform mutation operations and update *offP*
10 $\quad\quad$ Use Equation (20) to perform crossover operations and update *offP*
11 $\quad$ **end**
12 $\quad$ $\tilde{P} \leftarrow P \cup$ *offP* // Merge the initialized population with the offspring
13 $\quad$ Perform non-dominated sort on $\tilde{P}$
14 $\quad$ Use Equation (21) and Equation (22) to perform association operation
15 $\quad$ Perform environment selection on $\tilde{P}$
16 $\quad$ $P' \leftarrow$ ED($\tilde{P}$, A) // Call Algorithm 2
17 **end**
18 **return** P' // Return the updated population

4.2. VM Placement Optimization Method

To perform VM placement, an individual must be selected from the population obtained using the proposed CE-NSGAIII algorithm. This individual represents the final VM placement scheme among "task-VM-PM". To achieve task-loading-oriented VM placement optimization results, this study further proposed a VM placement optimization method, CE-VMP, based on the presented CE-NSGAIII. The pseudocode of the CE-VMP is shown as Algorithm 4.

Algorithm 4: CE-VMP

Input: The loading tasks *inputFolder*, the length of sequence L, initial value of chaotic parameter *CHAOS_PARAM*

Output: $< Task, VM, PM >$

```
                                                    // The VM placement scheme
Initialize brokerId
broker ← createBroker()                                    // Create a broker
vmList ← createVmList(brokerId)                                 // Create VMs
hostList ← createHostList()                                     // Create PMs
cloudletList ← createCloudletList(brokerId, inputFolder)  // Create cloudlets
broker.submitVmList(vmList)                        // Submit VMs to the broker
individual ← CE-NSGAIII(L, CHAOS_PARAM)            // Randomly select
  individuals from the set (Algorithm 3)
broker.setMapping(individual)                  // Map according to the scheme
broker.submitCloudletList(cloudletList)         // Submit tasks to the broker
return <Task, VM, PM>
```

The time cost of Algorithm 4 mainly originates from calling Algorithm 3 to generate the mapping scheme, so the time complexity is $O(KNL)$.

5. Experiment Results and Analysis

5.1. Evaluation Metrics

The CE-NSGAIII is essentially a multi-objective optimization algorithm. To evaluate its performance, we use four Quality Indicators (QIs): Generational Distance (GD) [40], Inverted Generational Distance (IGD) [41], Hypervolume (HV) [42], and IGD+ [43]. GD measures convergence by calculating the average distance from the obtained solutions to the true Pareto front. A smaller GD indicates better convergence. IGD evaluates both convergence and diversity by measuring the distance from the true Pareto front to the closest point in the obtained solution set, with smaller values indicating better performance in both aspects. HV assesses the spread and quality of the solution set by calculating the volume of the dominated region in the objective space, with larger values indicating better diversity. IGD+ is an extension of IGD that is Pareto-compliant, providing a more balanced evaluation of both convergence and diversity. A smaller IGD+ value indicates both better convergence to the true Pareto front and better diversity in the solution set. Unlike IGD, IGD+ provides a more balanced measure that avoids biases toward convergence or diversity alone. Using a combination of these QIs allows for a comprehensive evaluation, ensuring that the algorithm's performance is accurately assessed in terms of both convergence to the true Pareto front and the diversity of the solution set. As the true Pareto front is unknown, this study independently executed each comparative algorithm 10 times, combined all solutions, and selected the non-dominated solutions among them as the true Pareto front.

5.2. Experiment Setup

CloudSim [7] is a general-purpose, extensible cloud computing simulation software. This software is used to evaluate the effectiveness and efficiency of the proposed models, algorithms, and methods. The running environment is on a personal computer with Windows 11 and Intel Core i7-9700 CPU@3.00 GHz. In the experiment, an infrastructure-as-a-service (IaaS) provider is simulated in a cloud data center. Referring to Amazon Elastic Compute Cloud, the resource rental prices ranged from [0.7, 2.5] RMB ¥/h, covering CPU, memory, bandwidth, and storage resources, while the operating costs of PMs ranged from [3, 20] RMB ¥/h. The data center is assumed to have four different types of PMs, with

their parameters detailed in Table 2. The experiment involved tasks extracted from a real computing environment, which can be accessed from the website (accessed on 27 January 2024) (https://github.com/Hangyoo/VTJS). The tasks are divided into different sets: small task sets (200 tasks), medium task sets (400 and 600 tasks), and large task sets (800 tasks). These tasks are allocated to 8–10 VMs (in incremental steps of 1) for computation. The parameters of the VM are configured as shown in Table 3. The three types of task sets and varying numbers of VMs are combined to create 12 instances, denoted as set $T1, T2, \ldots, T12$.

Table 2. The PM parameter settings.

PM	CPU	MIPs	PE	Memory (GB)	Bandwidth (Mbps)
PH1	Intel Core 2 Extreme X6800	27079	2	24	20
PH2	Intel Core i5-2500K	83000	2	24	20
PH3	Intel Core i7–920	82300	4	32	20
PH4	Intel Core i7-875K	92100	4	32	20

Table 3. The VM parameter settings.

Parameters	Value Range
CPU ($\times 10^4$ MIPS)	[2, 8]
Memory (GB)	[1, 4]
Bandwidth (Mbps)	[2, 10]
PE	[1, 4]

To effectively eliminate random errors and improve the reliability of the experiment results, the results of each experiment are averaged over 10 runs.

5.3. The Effectiveness of the Presented CE-NSGAIII

The effectiveness of the presented CE-NSGAIII is evaluated by using IGD and HV metrics. To assess the effectiveness of the improvements made to the NSGA-III, comparisons are made with the D-NSGAIII, which incorporates only adaptive crossover and mutation operations, and the D-E-NSGAIII, which includes both adaptive crossover and mutation operations and elite guidance.

The results of the statistical tests on the IGD and HV values for the 12 instances (mean and standard deviation) are shown in Table 4, in which the compared algorithm obtaining the best performance is marked with dark gray shading and the second best is highlighted with light gray shading. Here, the Wilcoxon rank-sum test is adopted to compare the results at a significance level of 0.05. Symbols "+", "−" and "≈" mean that the competing algorithm is significantly better than, significantly worse than, and statistically tied by CE-NSGAIII respectively.

Table 4. Comparison of CE-NSGAIII with others on IGD and HV values by statistical tests (mean and standard deviation).

Instance	IGD			HV		
	D-NSGAIII	**D-E-NSGAIII**	**CE-NSGAIII**	**D-NSGAIII**	**D-E-NSGAIII**	**CE-NSGAIII**
T1(200 × 8)	$1.04 \times 10^{-2}(7.98 \times 10^{-4})$ –	$1.02 \times 10^{-2}(6.52 \times 10^{-4})$ –	$7.70 \times 10^{-3}(8.47 \times 10^{-4})$	$9.17 \times 10^{-1}(4.53 \times 10^{-2})$ ≈	$9.17 \times 10^{-1}(3.44 \times 10^{-2})$ ≈	$9.22 \times 10^{-1}(4.59 \times 10^{-1})$
T2(200 × 9)	$1.73 \times 10^{-2}(1.42 \times 10^{-3})$ –	$1.70 \times 10^{-2}(1.04 \times 10^{-3})$ –	$8.73 \times 10^{-3}(4.60 \times 10^{-4})$	$9.29 \times 10^{-1}(5.49 \times 10^{-2})$ +	$9.62 \times 10^{-1}(3.34 \times 10^{-2})$ +	$8.89 \times 10^{-1}(3.68 \times 10^{-2})$
T3(200 × 10)	$2.18 \times 10^{-2}(1.15 \times 10^{-3})$ –	$2.09 \times 10^{-2}(1.22 \times 10^{-3})$ –	$9.92 \times 10^{-3}(4.03 \times 10^{-4})$	$9.80 \times 10^{-1}(2.99 \times 10^{-2})$ +	$9.95 \times 10^{-1}(7.04 \times 10^{-3})$ +	$9.55 \times 10^{-1}(4.42 \times 10^{-2})$
T4(400 × 8)	$1.87 \times 10^{-2}(1.37 \times 10^{-3})$ –	$1.80 \times 10^{-2}(1.25 \times 10^{-3})$ –	$8.64 \times 10^{-3}(3.58 \times 10^{-4})$	$7.92 \times 10^{-1}(1.04 \times 10^{-1})$ –	$8.12 \times 10^{-1}(8.48 \times 10^{-2})$ –	$9.13 \times 10^{-1}(5.54 \times 10^{-2})$
T5(400 × 9)	$2.98 \times 10^{-2}(2.18 \times 10^{-3})$ –	$2.75 \times 10^{-2}(1.99 \times 10^{-3})$ –	$9.84 \times 10^{-3}(5.87 \times 10^{-4})$	$8.22 \times 10^{-1}(8.68 \times 10^{-2})$ –	$8.51 \times 10^{-1}(5.72 \times 10^{-2})$ –	$9.53 \times 10^{-1}(3.29 \times 10^{-2})$
T6(400 × 10)	$3.48 \times 10^{-2}(1.96 \times 10^{-3})$ –	$3.29 \times 10^{-2}(3.30 \times 10^{-3})$ –	$9.89 \times 10^{-3}(5.32 \times 10^{-4})$	$9.51 \times 10^{-1}(6.42 \times 10^{-2})$ +	$9.66 \times 10^{-1}(4.16 \times 10^{-2})$ +	$8.59 \times 10^{-1}(5.33 \times 10^{-2})$
T7(600 × 8)	$2.72 \times 10^{-2}(1.98 \times 10^{-3})$ –	$2.68 \times 10^{-2}(1.58 \times 10^{-3})$ –	$8.55 \times 10^{-3}(4.28 \times 10^{-4})$	$5.56 \times 10^{-1}(7.78 \times 10^{-2})$ –	$6.42 \times 10^{-1}(1.05 \times 10^{-1})$ –	$8.06 \times 10^{-1}(4.39 \times 10^{-2})$
T8(600 × 9)	$3.74 \times 10^{-2}(2.92 \times 10^{-3})$ –	$3.52 \times 10^{-2}(3.12 \times 10^{-3})$ –	$9.90 \times 10^{-3}(2.95 \times 10^{-4})$	$6.55 \times 10^{-1}(1.31 \times 10^{-1})$ –	$7.12 \times 10^{-1}(1.49 \times 10^{-1})$ –	$9.57 \times 10^{-1}(2.41 \times 10^{-2})$
T9(600 × 10)	$4.56 \times 10^{-2}(3.14 \times 10^{-3})$ –	$4.41 \times 10^{-2}(3.65 \times 10^{-3})$ –	$1.03 \times 10^{-2}(6.05 \times 10^{-4})$	$7.95 \times 10^{-1}(8.19 \times 10^{-2})$ –	$8.61 \times 10^{-1}(6.77 \times 10^{-2})$ ≈	$8.76 \times 10^{-1}(3.87 \times 10^{-2})$
T10(800 × 8)	$2.75 \times 10^{-2}(1.75 \times 10^{-3})$ –	$2.69 \times 10^{-2}(1.48 \times 10^{-3})$ –	$7.83 \times 10^{-3}(2.75 \times 10^{-4})$	$5.13 \times 10^{-1}(8.60 \times 10^{-2})$ –	$5.30 \times 10^{-1}(6.39 \times 10^{-2})$ –	$8.40 \times 10^{-1}(3.91 \times 10^{-2})$
T11(800 × 9)	$4.30 \times 10^{-2}(2.55 \times 10^{-3})$ –	$4.10 \times 10^{-2}(3.50 \times 10^{-3})$ –	$8.90 \times 10^{-3}(4.17 \times 10^{-4})$	$5.29 \times 10^{-1}(1.08 \times 10^{-1})$ –	$6.04 \times 10^{-1}(1.72 \times 10^{-1})$ –	$9.10 \times 10^{-1}(4.43 \times 10^{-2})$
T12(800 × 10)	$5.39 \times 10^{-2}(3.53 \times 10^{-3})$ –	$5.32 \times 10^{-2}(4.18 \times 10^{-3})$ –	$8.84 \times 10^{-3}(3.45 \times 10^{-4})$	$7.32 \times 10^{-1}(1.51 \times 10^{-1})$ –	$7.60 \times 10^{-1}(1.38 \times 10^{-1})$ –	$8.80 \times 10^{-1}(3.39 \times 10^{-2})$
+/–/≈	0/11/1	0/11/1	–	3/8/1	3/7/2	–

As presented in Table 4, the CE-NSGAIII exhibits the best overall performance in terms of the IGD and HV metrics. D-NSGAIII exhibits the worst performance, and D-E-NSGAIII is ranked second. This is because the CE-NSGAIII incorporates an elite guidance strategy that employs elite solutions to guide the entire population toward the Pareto front, thereby enhancing the algorithm's global search capability and improving the quality of the solutions. In addition, the chaotic mapping mechanism introduced in the CE-NSGAIII generates sequences with high randomness and ergodicity. This characteristic aids in the creation of a diverse initial population at the early stages of the algorithm [15]. The PBI distance is used to associate reference points and aids in fairly evaluating the crowding degree of individuals, thus better balancing convergence and diversity during the selection process. However, in smaller instances, the HV values of the CE-NSGAIII are slightly lower than those of the other two algorithms being compared. This is because the complex dynamic characteristics of chaotic mapping are not necessarily required for small-scale TLVMP problems and can even lead to reduced convergence owing to over exploration of the solution space.

5.4. Evaluation of the CE-VMP Method

5.4.1. Effectiveness

We adopt the classic evaluation metrics employed in [7], including: service level agreement violation rate (SLAV), a low value of SLAV represents fewer violations of the service level agreement, which ensures the quality of service. The number of virtual machine migrations (VMMs), a low number of VMMs indicates fewer service suspensions caused by VM migrations, which in turn guarantees the quality of service. Performance degradation due to VM migrations (PDM), the lower the PDM value, the less resource shortage due to VM migration. SLA violation time of a single active PM (SLATAH), which reflects the SLA violation due to resource shortage during the activity of PMs.

The proposed CE-VMP method is compared with the PABFD-VMP [7], FFD-VMP [8], COFFGA-VMP [9] and EEACS-VMP [10] methods. To ensure the fairness of the experiment evaluation, all methods are combined with the Inter Quartile Range (IQR) [44] host overload detection algorithm and the Minimum Migration Time (MMT) VM selection algorithm for the experiments. Different numbers of tasks are selected for the experiments: 200, 400, 600, 1000, 1200, and 2000.

Figure 5a shows the experimental results for the SLAV metric. For example, with 1000 tasks, the SLAV value for the CE-VMP is reduced by 41.78% to 74.01% compared with the other compared algorithms. This is because the CE-VMP considered the impact of task loading and matched resource supply more accurately based on actual task requirements, thereby reducing SLAV. Figure 5b shows the VMM results. When the number of tasks is high, the CE-VMP performs better because the CE-NSGAIII initially places VMs on PMs that can satisfy their resource needs, thereby avoiding unnecessary VM migrations. However, when the number of tasks is low, the results are similar to those of the other algorithms because the VM placement issue is relatively simple, and all algorithms quickly find effective solutions, resulting in fewer VM migration occurrences. Thus, the CE-VMP is not particularly advantageous in this case. Figure 5c displays the PDM metric results. When the number of tasks is low, the CE-VMP performs moderately. This is because when fewer VMs and PMs exist, the algorithm has a smaller search space, and the use of chaotic mapping leads to over-exploration, thereby reducing the algorithm's convergence. Consequently, less-than-optimal placement solutions that do not effectively match the VM requirements with PM resources are obtained. The PABFD-VMP is ranked second, as it sorts VMs according to their CPU resource requirements, ensuring that crucial VMs receive sufficient resources first, thereby reducing resource shortages caused by migrations.

Figure 5d shows the SLATAH results. For example, with 600 tasks, the SLATAH value for the CE-VMP decreased by 20.05% to 39.46% compared with those of the other algorithms. The VM placement scheme that considers task loading ensures more reasonable resource allocation and reduces instances where VM resource requests exceed allocated resources.

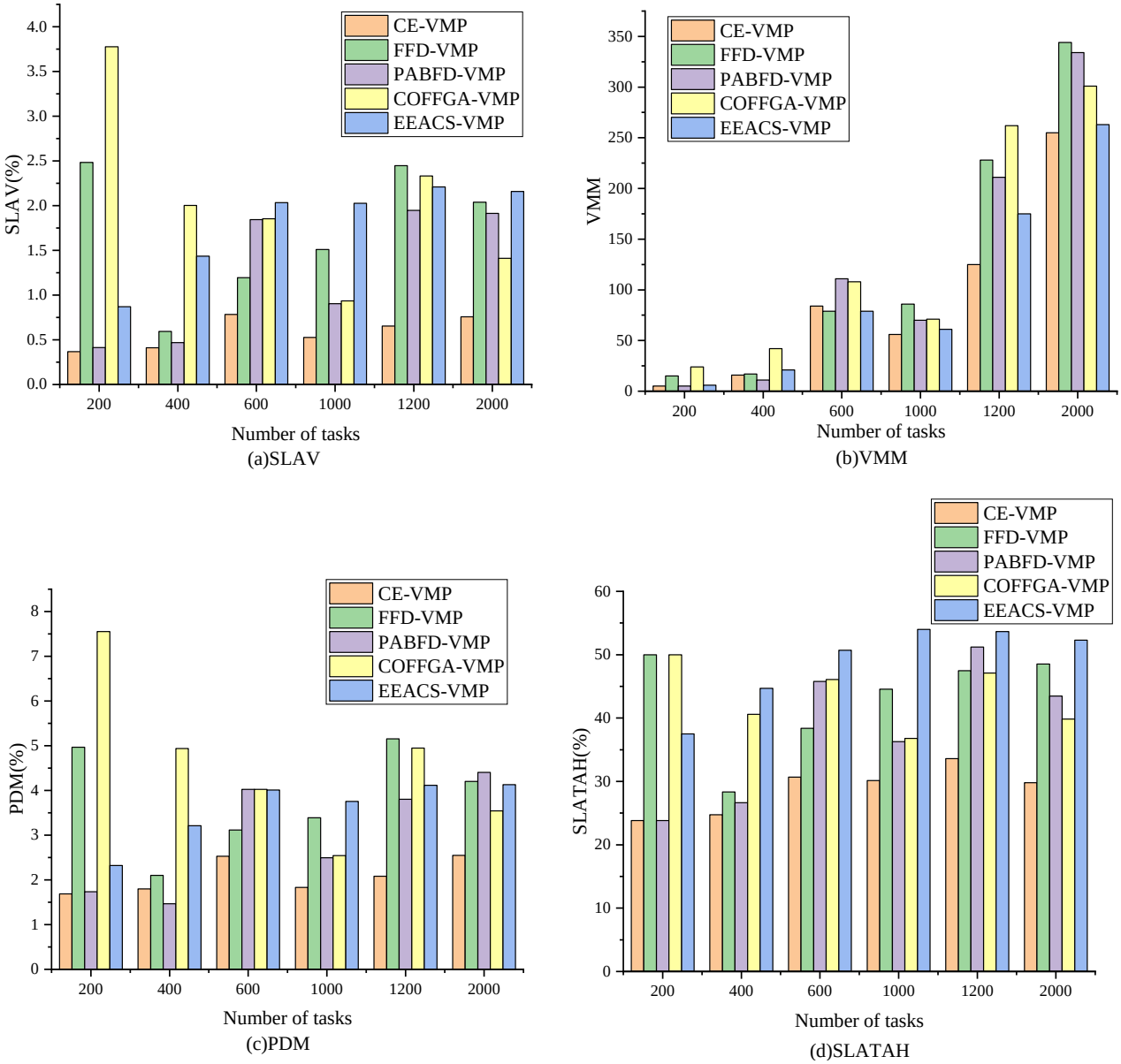

Figure 5. Comparison of metrics.

5.4.2. Efficiency

The CE-VMP method proposed in this study aimed to balance the interests of cloud service providers and users, considering the energy consumption of the cloud data center and the makespan and user rental costs. In most studies [7–10], the makespan and user rental cost have not been considered as optimization objectives. In this study, the energy consumption, makespan, and user rental costs are compared to evaluate the efficiency of the presented CE-VMP method.

Figure 6 depicts the experiment results on energy consumption of the proposed CE-VMP method and the other compared methods. As the number of tasks increases, the total active time of the VMs increases as the number of tasks undertaken by each VM increases. Meanwhile, PM resource utilization rises, leading to an increase in PM energy consumption

and total energy consumption. As can be seen from Figure 6, for instances $T2, T3, \ldots, T12$, the CE-VMP method has the lowest EC value. This is because the CE-VMP considers EC as one of the optimization objectives, and it also takes into account the impact of task loading on VM placement, which results in tasks being loaded to VMs more efficiently, shortening the task scheduling time, and indirectly reducing the activity time of the PMs which in turn reduces EC. For $T1$, when the number of tasks is small, the needed VMs are fewer, and the EC of the PM is relatively small, in this case, the optimization policy does not have a significant impact on the EC, resulting in an average performance of CE-VMP. The EEACS-VMP is the second best, as it considers each VM as an energy-consuming block, taking into account its individual energy requirements. EEACS-VMP ranks the PMs in descending order based on their energy efficiency and optimizes both server selection and pheromone updating rules within the ACS. Since the FFD-VMP places VMs in order of the size of their resource requirements, it does not take into account the optimization of EC and thus performs the worst.

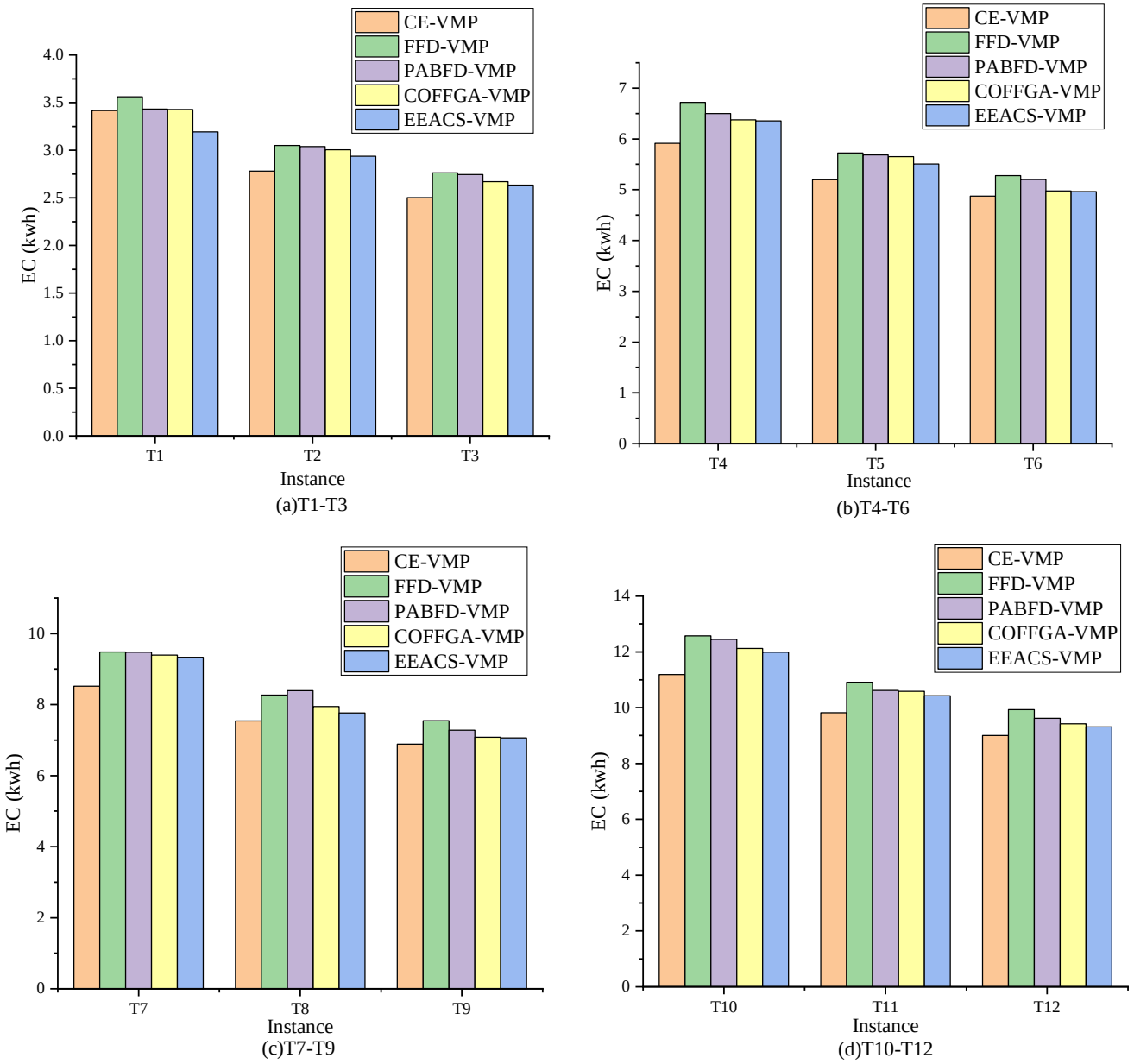

Figure 6. Comparison of EC.

Typically, the better the convergence of an algorithm for solving the TLVMP problem, the closer the solution set generated by the algorithm is to the true Pareto front. This achieves the optimal balance between makespan, user rental costs, and the energy consumption of cloud data centers. To assess the effectiveness of the proposed CE-VMP method in terms of makespan and user rental costs, we evaluated it indirectly by comparing the convergence of the primary model-solving algorithms within the CE-VMP method. Here, five algorithms, OMOPSO [21], SMPSO [22], NSGA-III [23], F-NSGAIII [25] and FAME [26] are selected as model-solving algorithms within the CE-VMP method and are compared and evaluated against the CE-NSGAIII.

Figures 7 and 8 presents the Pareto front experimental results of the six compared algorithms on 12 instances, covering the optimization results for the makespan, user rental costs, and cloud data center energy consumption. As depicted in Figure 7a,b, when the number of tasks is relatively small, the Pareto fronts of all methods are considerably close, with the CE-NSGAIII displaying only a slight advantage. As the number of tasks increases, the benefits of the CE-NSGAIII become more outperformed. This is because for intelligent optimization algorithms, when the number of tasks is high, the search space expands, requiring the algorithm to explore the possible solution space extensively. The proposed CE-NSGAIII casts the chaotic mapping mechanism, and the initial population it generates has high randomness and diversity, which is conducive to the global search of large-scale problems, and is not easy to fall into local optima prematurely; at the same time, the elite guidance strategy accelerates the search process of local optimal solutions. These factors help the CE-NSGAIII generate a better Pareto front, thereby allowing it to ensure a shorter makespan, lower user rental costs, and reduce the energy consumption of cloud data centers.

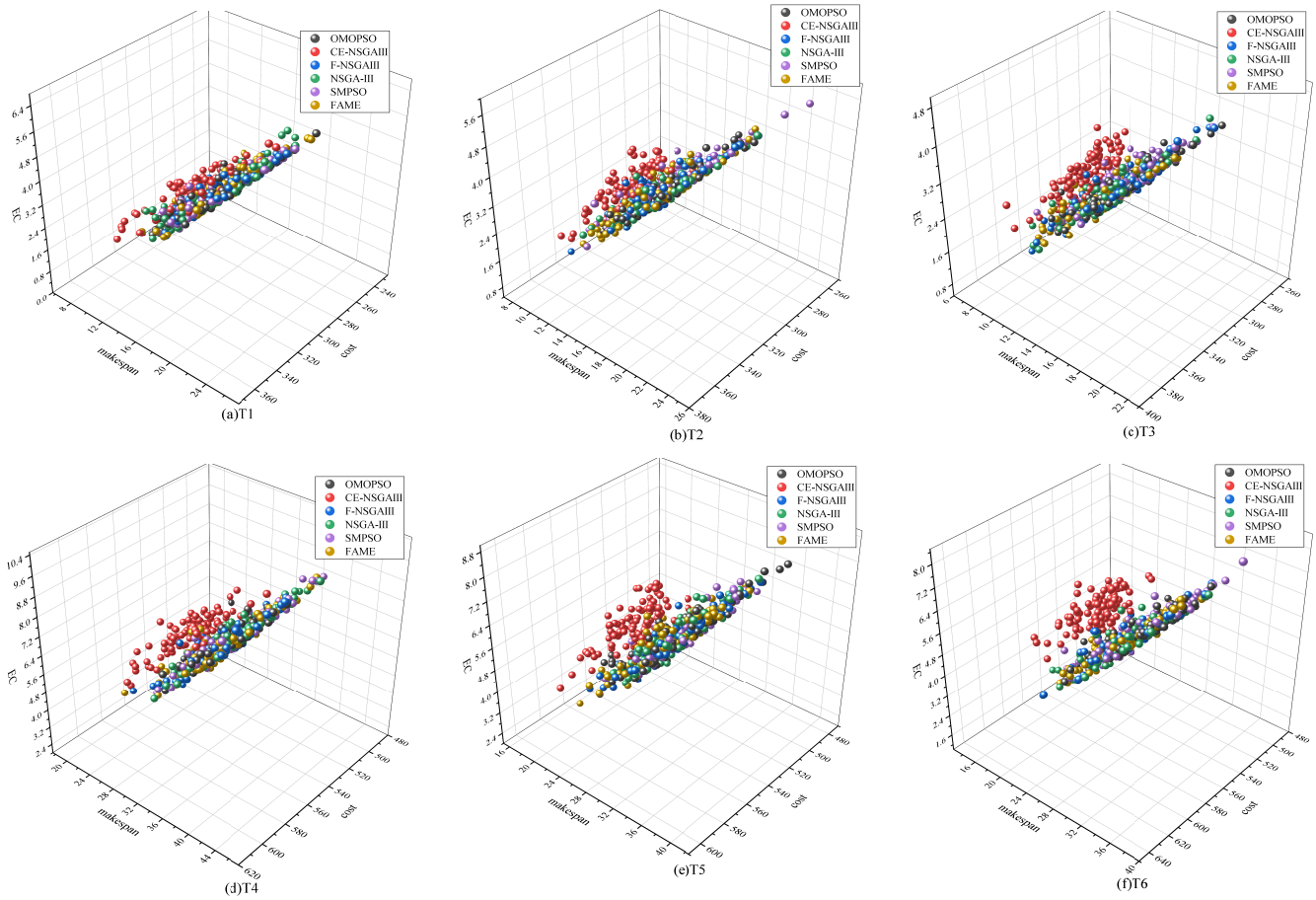

Figure 7. Comparison of Pareto front (T1–T6).

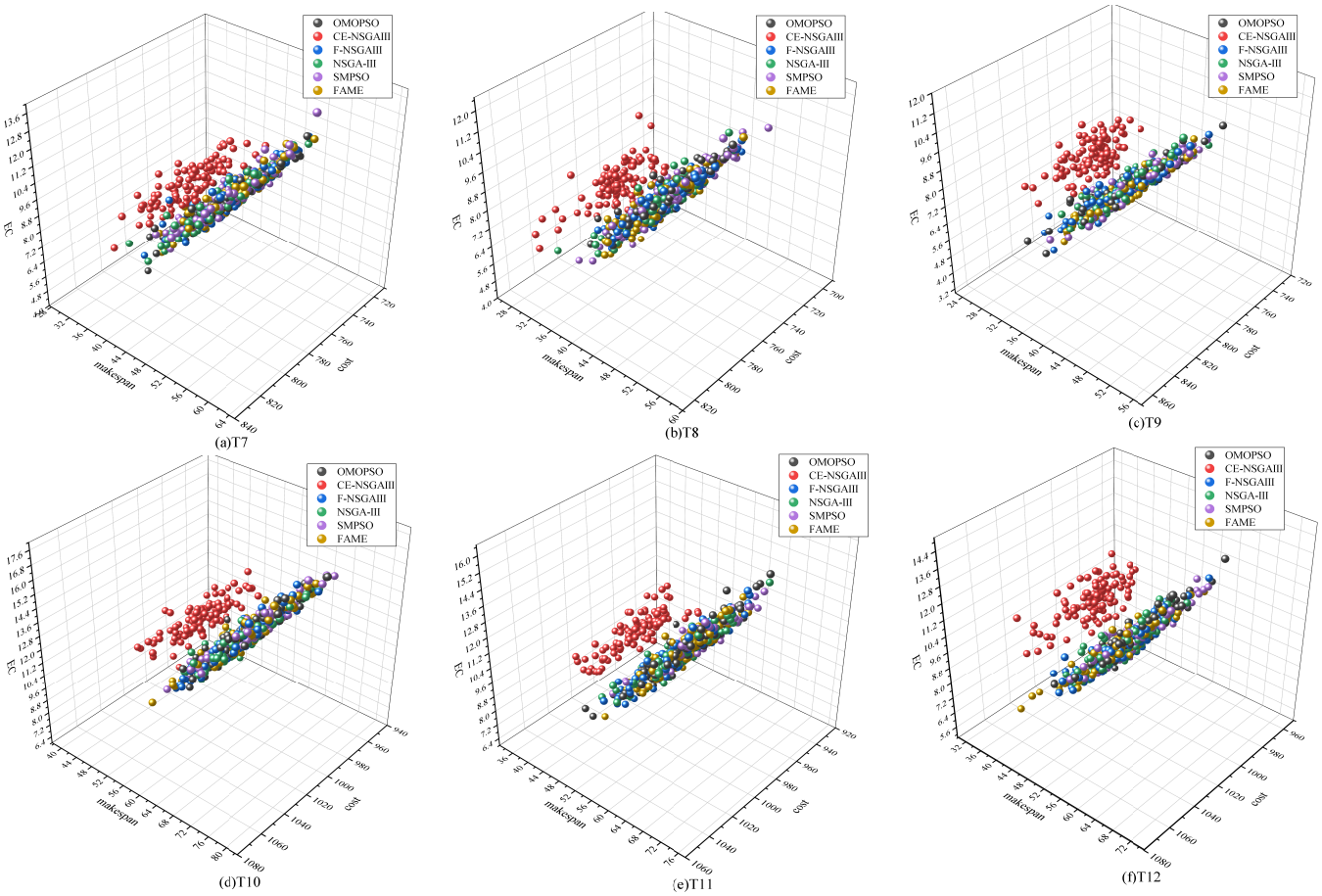

Figure 8. Comparison of Pareto front (T7–T12).

Figure 9 presents the results of the GD metrics. It can be seen that the GD metrics for the CE-NSGAIII proposed in this study are superior to those of the other compared algorithms. This suggests that the solution set generated by the CE-NSGAIII is closer to the optimal Pareto front, indicating better convergence performance. This improvement is attributed to the introduction of the PBI association strategy, which guides the search process by incorporating a penalty term. This scheme helps the algorithm approximate the Pareto frontier more accurately while avoiding the generation of infeasible solutions. Furthermore, as shown in Figure 9a,b, all algorithms exhibit significant fluctuations in GD values. These fluctuations can be explained by the limited solution space when the number of tasks is small, making the algorithms more susceptible to falling inside local optima. As a result, during the iterative process, the algorithms are prone to being trapped in various local optima, leading to considerable variations in the generated Pareto solution sets.

The experiment results of the HV values are presented in Table 5. The compared algorithm obtaining the best performance is marked with dark gray shading. It is observed that when the number of tasks is high, the HV values for the CE-NSGAIII outperform those of the other compared algorithms across eight instances ($T4, T5, T7, ..., T12$). This performance is attributed to the incorporation of the chaotic mapping mechanism and the PBI association strategy, which collectively enhance both the convergence and diversity of the obtained Pareto solution set. However, the CE-NSGAIII shows generally in the four instances of $T1$, $T2$, $T3$, and $T6$. This is likely due to the fact that, for small-scale problems, the random initialization strategies used by the other compared algorithms are simple and effective, allowing them to cover the solution space more quickly. In contrast, when the population size is limited, the elite guidance strategy in CE-NSGAIII may prematurely

fixate on a specific set of solutions, potentially overlooking other beneficial search directions and thus reducing diversity during the initial exploration phase. In instances *T*2, *T*3, and *T*6, the NSGA-III achieves the best performance, likely because its original reference point association strategy more accurately guides the population evolution compared to the PBI association strategy used in CE-NSGAIII.

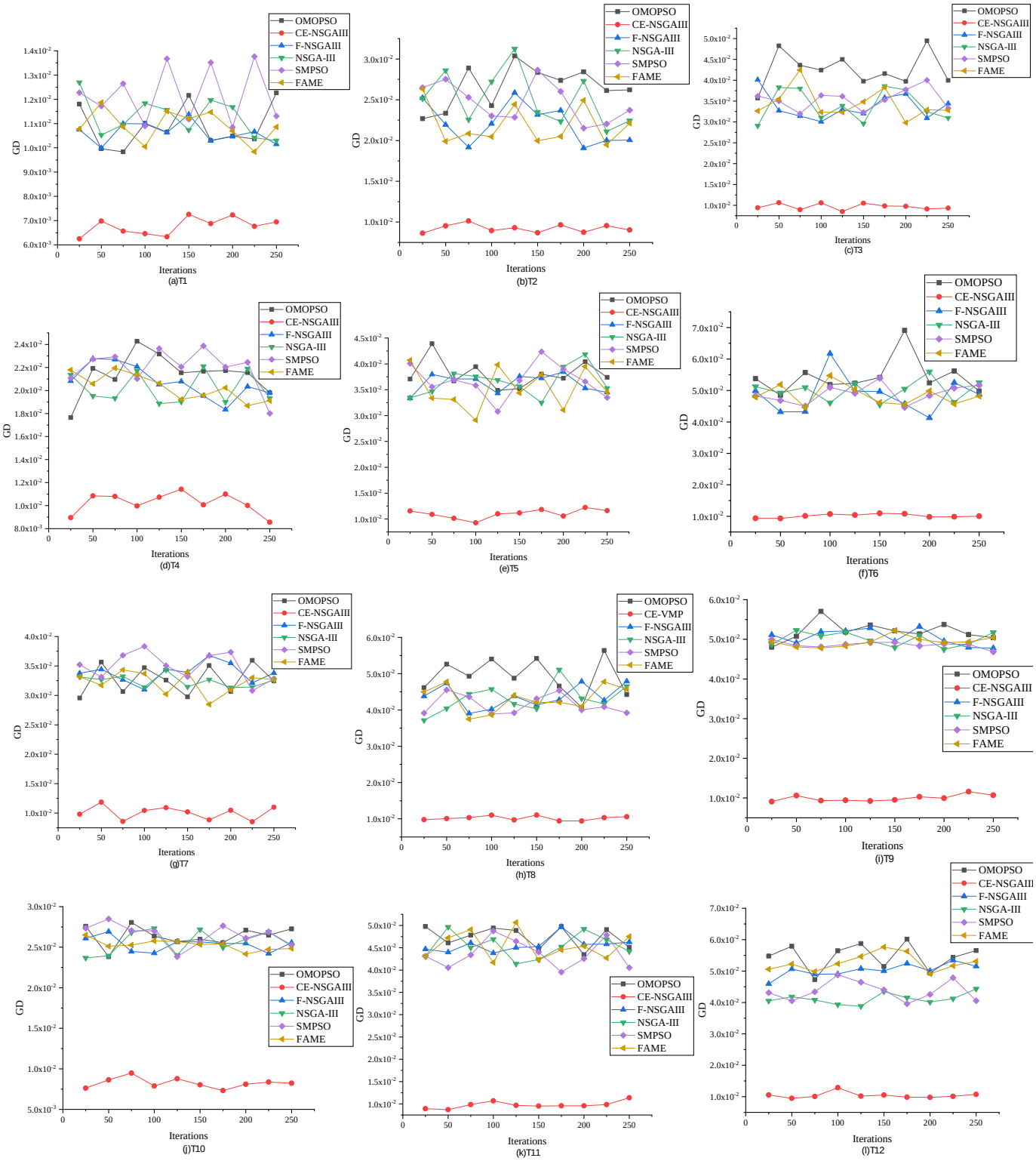

Figure 9. Comparison of GD.

The results of the IGD values, as shown in Table 6, demonstrate that the CE-NSGAIII algorithm outperforms the other compared algorithms, with the best-performing algorithm highlighted in dark gray. The results imply that the CE-NSGAIII generates a solution set closer to the true Pareto front with a more uniform distribution. This performance advantage is primarily due to the chaotic mapping mechanism employed by the CE-NSGAIII, which produces initial solutions with high randomness and comprehensive exploration capabilities, enabling more efficient exploration of multiple regions of the problem space. Additionally, the elite guidance strategy facilitates a rapid return to promising regions, effectively enhancing the convergence of the algorithm. As a result, the CE-NSGAIII achieves better IGD metrics when solving the TLVMP problem.

The experiment results of the IGD+ values are presented in Table 7. The compared algorithm obtaining the best performance is marked with dark gray shading. It can be observed that the IGD+ value of CE-NSGAIII outperforms the other comparative algorithms in 10 instances ($T1$, $T4$, $T5$, ..., $T12$), as the number of tasks increases. This improvement is attributed to the PBI association strategy, which effectively guides population evolution, and the chaotic mapping mechanism, which enhances the randomness of the generated population and broadens the search space coverage. These combined effects enable CE-NSGAIII to find solutions closer to the true Pareto front while covering a more diverse range of solution regions, resulting in a larger set of high-quality solutions. However, in instances $T2$ and $T3$, where the number of tasks is smaller, the performance of CE-NSGAIII is somewhat diminished. This can be explained by the fact that the elite strategy may narrow the search scope to small-scale problems, limiting the diversity and coverage of the Pareto front.

In summation, the proposed CE-NSGAIII exhibits optimal performance in terms of the IGD metric and performs well in terms of the HV metric in medium and large test instances. This demonstrates that the CE-NSGAIII can effectively and efficiently solve the TLVMP problem, particularly when the number of tasks is high. Consequently, obtaining a Pareto front with better convergence and diversity is easier. This characteristic of the CE-NSGAIII ensures that when using the CE-VMP method based on the CE-NSGAIII for the issue of VM placement, the makespan is shorter, user rental costs are lower, and the energy consumption of cloud data centers reduces.

Table 5. Comparison of CE-NSGAIII with others on HV values by statistical tests (mean and standard deviation).

Instance	OMOPSO	NSGA-III	F-NSGAIII	SMPSO	FAME	CE-NSGAIII
T1(200 × 8)	$9.11 \times 10^{-1}(4.29 \times 10^{-1}) \approx$	$9.13 \times 10^{-1}(4.53 \times 10^{-1}) \approx$	$9.44 \times 10^{-1}(4.68 \times 10^{-1}) \approx$	$9.34 \times 10^{-1}(4.04 \times 10^{-2}) \approx$	$9.21 \times 10^{-1}(5.30 \times 10^{-2}) \approx$	$9.22 \times 10^{-1}(4.59 \times 10^{-1})$
T2(200 × 9)	$9.44 \times 10^{-1}(3.41 \times 10^{-2}) +$	$9.54 \times 10^{-1}(5.51 \times 10^{-2}) +$	$9.03 \times 10^{-1}(6.90 \times 10^{-2}) \approx$	$9.56 \times 10^{-1}(3.92 \times 10^{-2}) +$	$9.28 \times 10^{-1}(4.20 \times 10^{-2}) +$	$8.89 \times 10^{-1}(3.68 \times 10^{-2})$
T3(200 × 10)	$9.82 \times 10^{-1}(2.40 \times 10^{-2}) +$	$9.92 \times 10^{-1}(1.25 \times 10^{-2}) +$	$9.75 \times 10^{-1}(2.11 \times 10^{-2}) +$	$9.99 \times 10^{-1}(8.85 \times 10^{-4}) +$	$9.88 \times 10^{-1}(1.96 \times 10^{-2}) +$	$9.55 \times 10^{-1}(4.42 \times 10^{-2})$
T4(400 × 8)	$7.52 \times 10^{-1}(7.89 \times 10^{-2}) -$	$7.86 \times 10^{-1}(6.71 \times 10^{-2}) -$	$7.95 \times 10^{-1}(6.20 \times 10^{-2}) -$	$8.28 \times 10^{-1}(8.45 \times 10^{-2}) -$	$7.70 \times 10^{-1}(5.36 \times 10^{-2}) -$	$9.13 \times 10^{-1}(5.54 \times 10^{-2})$
T5(400 × 9)	$8.25 \times 10^{-1}(1.02 \times 10^{-2}) -$	$8.32 \times 10^{-1}(4.96 \times 10^{-2}) -$	$9.28 \times 10^{-1}(5.55 \times 10^{-2}) -$	$8.56 \times 10^{-1}(6.81 \times 10^{-2}) -$	$8.28 \times 10^{-1}(7.18 \times 10^{-2}) -$	$9.53 \times 10^{-1}(3.29 \times 10^{-2})$
T6(400 × 10)	$9.17 \times 10^{-1}(5.93 \times 10^{-2}) +$	$9.38 \times 10^{-1}(4.36 \times 10^{-2}) +$	$8.58 \times 10^{-1}(1.47 \times 10^{-2}) \approx$	$9.28 \times 10^{-1}(8.11 \times 10^{-2}) +$	$9.36 \times 10^{-1}(6.67 \times 10^{-2}) +$	$8.59 \times 10^{-1}(5.33 \times 10^{-2})$
T7(600 × 8)	$5.52 \times 10^{-1}(8.93 \times 10^{-2}) -$	$5.39 \times 10^{-1}(6.21 \times 10^{-2}) -$	$5.44 \times 10^{-1}(1.17 \times 10^{-1}) -$	$6.02 \times 10^{-1}(9.11 \times 10^{-2}) -$	$5.64 \times 10^{-1}(7.77 \times 10^{-2}) -$	$8.06 \times 10^{-1}(4.39 \times 10^{-2})$
T8(600 × 9)	$6.87 \times 10^{-1}(9.89 \times 10^{-2}) -$	$6.94 \times 10^{-1}(8.12 \times 10^{-2}) -$	$6.35 \times 10^{-1}(1.13 \times 10^{-1}) -$	$6.65 \times 10^{-1}(6.88 \times 10^{-2}) -$	$7.38 \times 10^{-1}(9.49 \times 10^{-2}) -$	$9.57 \times 10^{-1}(2.41 \times 10^{-2})$
T9(600 × 10)	$7.83 \times 10^{-1}(8.83 \times 10^{-2}) -$	$8.54 \times 10^{-1}(1.11 \times 10^{-1}) -$	$7.99 \times 10^{-1}(8.24 \times 10^{-2}) -$	$8.76 \times 10^{-1}(3.87 \times 10^{-2}) -$	$8.44 \times 10^{-1}(1.06 \times 10^{-1}) -$	$8.98 \times 10^{-1}(7.00 \times 10^{-2})$
T10(800 × 8)	$5.60 \times 10^{-1}(1.14 \times 10^{-1}) -$	$5.28 \times 10^{-1}(9.08 \times 10^{-2}) -$	$5.08 \times 10^{-1}(7.29 \times 10^{-2}) -$	$5.85 \times 10^{-1}(1.15 \times 10^{-1}) -$	$5.52 \times 10^{-1}(1.19 \times 10^{-1}) -$	$8.40 \times 10^{-1}(3.91 \times 10^{-2})$
T11(800 × 9)	$5.59 \times 10^{-1}(1.14 \times 10^{-1}) -$	$5.16 \times 10^{-1}(1.06 \times 10^{-1}) -$	$5.12 \times 10^{-1}(8.25 \times 10^{-2}) -$	$5.92 \times 10^{-1}(1.33 \times 10^{-1}) -$	$6.26 \times 10^{-1}(9.35 \times 10^{-2}) -$	$9.10 \times 10^{-1}(4.43 \times 10^{-2})$
T12(800 × 10)	$6.73 \times 10^{-1}(8.47 \times 10^{-2}) -$	$6.74 \times 10^{-1}(9.98 \times 10^{-2}) -$	$7.40 \times 10^{-1}(1.02 \times 10^{-1}) -$	$7.87 \times 10^{-1}(1.07 \times 10^{-1}) -$	$7.05 \times 10^{-1}(9.62 \times 10^{-2}) -$	$8.80 \times 10^{-1}(3.39 \times 10^{-2})$
+/–/≈	3/8/1	3/8/1	1/8/3	3/8/1	3/8/1	–

Table 6. Comparison of CE-NSGAIII with others on IGD values by statistical tests (mean and standard deviation).

Instance	OMOPSO	NSGA-III	F-NSGAIII	SMPSO	FAME	CE-NSGAIII
T1(200 × 8)	$1.07 \times 10^{-2}(8.23 \times 10^{-4}) \approx$	$1.06 \times 10^{-2}(4.75 \times 10^{-4}) \approx$	$1.04 \times 10^{-2}(5.89 \times 10^{-4}) \approx$	$1.03 \times 10^{-2}(6.35 \times 10^{-4}) \approx$	$1.07 \times 10^{-2}(5.37 \times 10^{-4}) \approx$	$7.70 \times 10^{-3}(8.47 \times 10^{-4})$
T2(200 × 9)	$1.08 \times 10^{-2}(1.20 \times 10^{-3}) -$	$1.62 \times 10^{-2}(6.74 \times 10^{-4}) -$	$1.71 \times 10^{-2}(1.04 \times 10^{-3}) -$	$1.68 \times 10^{-2}(6.51 \times 10^{-4}) -$	$1.69 \times 10^{-2}(8.43 \times 10^{-4}) -$	$8.73 \times 10^{-3}(4.60 \times 10^{-4})$
T3(200 × 10)	$2.23 \times 10^{-2}(1.73 \times 10^{-3}) -$	$2.23 \times 10^{-2}(1.32 \times 10^{-3}) -$	$2.09 \times 10^{-2}(1.45 \times 10^{-3}) -$	$2.09 \times 10^{-2}(1.25 \times 10^{-3}) -$	$2.00 \times 10^{-2}(1.17 \times 10^{-3}) -$	$9.92 \times 10^{-3}(4.03 \times 10^{-4})$
T4(400 × 8)	$1.83 \times 10^{-2}(1.05 \times 10^{-3}) -$	$1.82 \times 10^{-2}(5.95 \times 10^{-4}) -$	$1.77 \times 10^{-2}(8.10 \times 10^{-4}) -$	$1.83 \times 10^{-2}(1.11 \times 10^{-3}) -$	$1.78 \times 10^{-2}(1.15 \times 10^{-3}) -$	$8.64 \times 10^{-3}(3.58 \times 10^{-4})$
T5(400 × 9)	$2.85 \times 10^{-2}(2.35 \times 10^{-3}) -$	$3.04 \times 10^{-2}(3.68 \times 10^{-3}) -$	$2.82 \times 10^{-2}(1.82 \times 10^{-3}) -$	$2.76 \times 10^{-2}(1.45 \times 10^{-3}) -$	$2.76 \times 10^{-2}(2.74 \times 10^{-3}) -$	$9.84 \times 10^{-3}(5.87 \times 10^{-4})$
T6(400 × 10)	$3.37 \times 10^{-2}(2.25 \times 10^{-3}) -$	$3.32 \times 10^{-2}(2.66 \times 10^{-3}) -$	$3.31 \times 10^{-2}(1.41 \times 10^{-3}) -$	$3.37 \times 10^{-2}(2.27 \times 10^{-3}) -$	$3.33 \times 10^{-2}(1.08 \times 10^{-3}) -$	$9.89 \times 10^{-3}(5.32 \times 10^{-4})$
T7(600 × 8)	$2.60 \times 10^{-2}(1.97 \times 10^{-3}) -$	$2.67 \times 10^{-2}(1.23 \times 10^{-3}) -$	$2.71 \times 10^{-2}(1.29 \times 10^{-3}) -$	$2.66 \times 10^{-2}(1.91 \times 10^{-3}) -$	$2.64 \times 10^{-2}(1.56 \times 10^{-3}) -$	$8.55 \times 10^{-3}(4.28 \times 10^{-4})$
T8(600 × 9)	$3.42 \times 10^{-2}(3.11 \times 10^{-3}) -$	$3.81 \times 10^{-2}(2.64 \times 10^{-3}) -$	$3.53 \times 10^{-2}(1.95 \times 10^{-3}) -$	$3.67 \times 10^{-2}(3.72 \times 10^{-3}) -$	$3.56 \times 10^{-2}(3.13 \times 10^{-3}) -$	$9.90 \times 10^{-3}(2.95 \times 10^{-4})$
T9(600 × 10)	$4.40 \times 10^{-2}(1.97 \times 10^{-3}) -$	$4.40 \times 10^{-2}(2.24 \times 10^{-3}) -$	$4.38 \times 10^{-2}(3.13 \times 10^{-3}) -$	$4.42 \times 10^{-2}(2.12 \times 10^{-3}) -$	$4.58 \times 10^{-2}(1.47 \times 10^{-3}) -$	$1.03 \times 10^{-2}(6.05 \times 10^{-4})$
T10(800 × 8)	$2.76 \times 10^{-2}(1.64 \times 10^{-3}) -$	$2.64 \times 10^{-2}(1.82 \times 10^{-3}) -$	$2.89 \times 10^{-2}(2.68 \times 10^{-3}) -$	$2.75 \times 10^{-2}(1.90 \times 10^{-3}) -$	$2.67 \times 10^{-2}(3.10 \times 10^{-3}) -$	$7.83 \times 10^{-3}(2.75 \times 10^{-4})$
T11(800 × 9)	$4.12 \times 10^{-2}(3.32 \times 10^{-3}) -$	$4.05 \times 10^{-2}(3.03 \times 10^{-3}) -$	$4.19 \times 10^{-2}(4.33 \times 10^{-3}) -$	$4.08 \times 10^{-2}(2.28 \times 10^{-3}) -$	$4.21 \times 10^{-2}(3.43 \times 10^{-3}) -$	$8.90 \times 10^{-3}(4.17 \times 10^{-4})$
T12(800 × 10)	$5.38 \times 10^{-2}(4.34 \times 10^{-3}) -$	$4.17 \times 10^{-2}(2.68 \times 10^{-3}) -$	$5.38 \times 10^{-2}(3.50 \times 10^{-3}) -$	$4.98 \times 10^{-2}(2.82 \times 10^{-3}) -$	$5.21 \times 10^{-2}(5.36 \times 10^{-3}) -$	$8.84 \times 10^{-3}(3.45 \times 10^{-4})$
+/–/≈	0/11/1	0/11/1	0/11/1	0/11/1	0/11/1	–

Table 7. Comparison of CE-NSGAIII with others on IGD+ values by statistical tests (mean and standard deviation).

Instance	OMOPSO	NSGA-III	F-NSGAIII	SMPSO	FAME	CE-NSGAIII
T1(200 × 8)	$6.44 \times 10^{-3}(3.94 \times 10^{-3})$ –	$6.64 \times 10^{-3}(5.40 \times 10^{-3})$ –	$3.45 \times 10^{-3}(4.25 \times 10^{-3}) \approx$	$5.27 \times 10^{-3}(5.09 \times 10^{-3})$ –	$6.48 \times 10^{-3}(5.26 \times 10^{-3})$ –	$2.68 \times 10^{-3}(1.55 \times 10^{-3})$
T2(200 × 9)	$2.00 \times 10^{-3}(1.80 \times 10^{-3})$ +	$1.99 \times 10^{-3}(3.31 \times 10^{-3})$ +	$5.97 \times 10^{-3}(4.45 \times 10^{-3})$ –	$3.02 \times 10^{-3}(4.76 \times 10^{-3})$ +	$3.46 \times 10^{-3}(3.00 \times 10^{-3})$ +	$3.52 \times 10^{-3}(2.01 \times 10^{-3})$
T3(200 × 10)	$3.48 \times 10^{-3}(2.49 \times 10^{-3}) \approx$	$4.99 \times 10^{-3}(8.77 \times 10^{-3})$ –	$2.46 \times 10^{-3}(2.60 \times 10^{-3})$ +	$5.92 \times 10^{-3}(8.97 \times 10^{-3})$ –	$3.24 \times 10^{-3}(5.67 \times 10^{-3}) \approx$	$2.81 \times 10^{-3}(3.11 \times 10^{-3})$
T4(400 × 8)	$3.08 \times 10^{-2}(2.16 \times 10^{-2})$ –	$2.68 \times 10^{-2}(1.29 \times 10^{-2})$ –	$2.08 \times 10^{-2}(1.20 \times 10^{-2})$ –	$2.06 \times 10^{-2}(1.82 \times 10^{-2})$ –	$2.96 \times 10^{-2}(1.22 \times 10^{-2})$ –	$2.80 \times 10^{-3}(2.01 \times 10^{-3})$
T5(400 × 9)	$3.81 \times 10^{-2}(2.96 \times 10^{-2})$ –	$3.84 \times 10^{-2}(1.88 \times 10^{-2})$ –	$7.27 \times 10^{-3}(1.34 \times 10^{-2})$ –	$1.30 \times 10^{-2}(1.33 \times 10^{-2})$ –	$2.29 \times 10^{-2}(1.87 \times 10^{-2})$ –	$1.80 \times 10^{-3}(1.68 \times 10^{-3})$
T6(400 × 10)	$7.55 \times 10^{-3}(7.95 \times 10^{-3})$ –	$6.56 \times 10^{-3}(6.61 \times 10^{-3}) \approx$	$7.55 \times 10^{-3}(1.23 \times 10^{-2})$ –	$8.17 \times 10^{-3}(1.24 \times 10^{-2})$ –	$7.37 \times 10^{-3}(1.07 \times 10^{-2})$ –	$5.47 \times 10^{-3}(2.76 \times 10^{-3})$
T7(600 × 8)	$6.39 \times 10^{-2}(3.00 \times 10^{-2})$ –	$8.21 \times 10^{-2}(3.38 \times 10^{-2})$ –	$7.66 \times 10^{-2}(3.70 \times 10^{-2})$ –	$5.81 \times 10^{-2}(3.27 \times 10^{-2})$ –	$6.92 \times 10^{-2}(1.80 \times 10^{-2})$ –	$5.62 \times 10^{-3}(2.56 \times 10^{-3})$
T8(600 × 9)	$6.72 \times 10^{-2}(3.73 \times 10^{-2})$ –	$8.35 \times 10^{-2}(4.48 \times 10^{-2})$ –	$1.05 \times 10^{-1}(5.73 \times 10^{-2})$ –	$7.99 \times 10^{-2}(3.91 \times 10^{-2})$ –	$5.35 \times 10^{-2}(4.83 \times 10^{-2})$ –	$3.28 \times 10^{-3}(3.43 \times 10^{-3})$
T9(600 × 10)	$2.48 \times 10^{-2}(1.26 \times 10^{-2})$ –	$1.48 \times 10^{-2}(1.46 \times 10^{-2})$ –	$7.66 \times 10^{-2}(3.70 \times 10^{-2})$ –	$4.73 \times 10^{-3}(6.16 \times 10^{-3}) \approx$	$1.96 \times 10^{-2}(1.66 \times 10^{-2})$ –	$4.65 \times 10^{-3}(2.87 \times 10^{-3})$
T10(800 × 8)	$1.17 \times 10^{-1}(5.08 \times 10^{-2})$ –	$9.96 \times 10^{-2}(4.40 \times 10^{-2})$ –	$1.22 \times 10^{-1}(5.25 \times 10^{-2})$ –	$1.04 \times 10^{-1}(5.24 \times 10^{-2})$ –	$1.05 \times 10^{-1}(5.69 \times 10^{-2})$ –	$3.81 \times 10^{-3}(1.61 \times 10^{-3})$
T11(800 × 9)	$1.09 \times 10^{-1}(7.23 \times 10^{-2})$ –	$1.41 \times 10^{-1}(6.66 \times 10^{-2})$ –	$1.60 \times 10^{-1}(9.90 \times 10^{-2})$ –	$9.74 \times 10^{-2}(8.01 \times 10^{-2})$ –	$8.75 \times 10^{-2}(6.35 \times 10^{-2})$ –	$3.63 \times 10^{-3}(2.97 \times 10^{-3})$
T12(800 × 10)	$5.61 \times 10^{-2}(3.51 \times 10^{-2})$ –	$9.45 \times 10^{-2}(5.11 \times 10^{-2})$ –	$4.11 \times 10^{-2}(4.11 \times 10^{-2})$ –	$3.13 \times 10^{-2}(3.01 \times 10^{-2})$ –	$3.94 \times 10^{-2}(3.12 \times 10^{-2})$ –	$3.02 \times 10^{-3}(1.90 \times 10^{-3})$
+/–/≈	1/10/1	1/10/1	1/10/1	1/10/1	1/10/1	–

6. Conclusions

In this study, a task-driven multi-objective VM placement optimization model for cloud data center environments is proposed and examined. The model is employed to address issues such as long task completion times, high user rental costs, and significant energy consumption in cloud data centers. To efficiently solve this model, chaotic mapping is used to initialize the population and the adaptive crossover and mutation strategies are applied. Furthermore, PBI distance is utilized to associate reference points, and population evolution is achieved through elite guidance. An improved CE-NSGAIII multi-objective optimization algorithm is proposed to solve the model, and a CE-VMP method is further presented for the issue of VM placement.

The experiment results indicated that the proposed method effectively mitigated issues related to long task completion times, high rental costs, and energy consumption in cloud data centers and provided a high-quality Pareto front that exhibited better convergence and diversity compared with those of the other compared algorithms. However, the method has certain limitations. The current approach did not fully account for the dynamic changes in resource demands within cloud data center environments. In future research, virtual resource allocation schemes that adapt to dynamically changing, task-driven VM quantities and configurations in complex environments will be explored. Thus, in future studies, the algorithm's real-time performance will be enhanced while maintaining service quality to better respond to dynamically changing workloads.

Author Contributions: Conceptualization, R.Y.; methodology, R.Y. and Z.L. (Zhaonan Li); software, R.Y.; writing—original draft preparation, R.Y.; writing—review and editing, R.Y., J.Q. and Z.L. (Zhihua Li).; funding acquisition, Z.L. (Zhihua Li). All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Smart Manufacturing New Model Application Project Ministry of Industry and Information Technology (No. ZH-XZ-18004), the Future Research Projects Funds for the Science and Technology Department of Jiangsu Province (No. BY2013015-23), the Fundamental Research Funds for the Ministry of Education (No. JUSRP211A41), the Fundamental Research Funds for the Central Universities (No. JUSRP42003), and the 111 Project (No. B2018).

Data Availability Statement: Data will be made available upon request.

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

References

1. Usmani, Z.; Singh, S. A Survey of Virtual Machine Placement Techniques in a Cloud Data Center. *Procedia Comput. Sci.* **2016**, *78*, 491–498. [CrossRef]
2. Belgacem, A.; Beghdad-Bey, K.; Nacer, H. Task Scheduling in Cloud Computing Environment: A Comprehensive Analysis. In *Advances in Computing Systems and Applications*; Springer: Berlin/Heidelberg, Germany, 2018. [CrossRef]
3. Heger, D.A. Optimized Resource Allocation & Task Scheduling Challenges in Cloud Computing Environments. 2010. Available online: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=d4ebdac3dcfd1c2e10e1c4743036902b2f65d720 (accessed on 20 December 2024).
4. Gupta, A.; Garg, R. Load Balancing Based Task Scheduling with ACO in Cloud Computing. In Proceedings of the 2017 International Conference on Computer and Applications (ICCA), Doha, United Arab Emirates, 6–7 September 2017; pp. 174–179. [CrossRef]
5. Zuo, X.; Zhang, G.; Tan, W. Self-Adaptive Learning PSO-Based Deadline Constrained Task Scheduling for Hybrid IaaS Cloud. *IEEE Trans. Autom. Sci. Eng.* **2014**, *11*, 564–573. [CrossRef]
6. Sofia, A.S.; Ganeshkumar, P. Multi-objective Task Scheduling to Minimize Energy Consumption and Makespan of Cloud Computing Using NSGA-II. *J. Netw. Syst. Manag.* **2018**, *26*, 463–485. [CrossRef]
7. Beloglazov, A.; Buyya, R. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers. *Concurr. Comput. Pract. Exp.* **2012**, *24*, 1397–1420. [CrossRef]

8. Keller, G.; Tighe, M.; Lutfiyya, H.; Bauer, M.A. An analysis of first fit heuristics for the virtual machine relocation problem. In Proceedings of the 2012 8th International Conference on Network and Service Management (CNSM) and 2012 Workshop on Systems Virtualization Management (SVM), Las Vegas, NV, USA, 22–26 October 2012; pp. 406–413.
9. Hallawi, H.; Mehnen, J.; He, H. Multi-Capacity Combinatorial Ordering GA in Application to Cloud resources allocation and efficient virtual machines consolidation. *Future Gener. Comput. Syst.* **2017**, *69*, 1–10. [CrossRef]
10. Duan, L.; Wang, J.; Wang, H. An energy-aware ant colony optimization strategy for virtual machine placement in cloud computing. *Clust. Comput.* **2024**, *27*, 14269–14282. [CrossRef]
11. Shirvani, M.H. An energy-efficient topology-aware virtual machine placement in Cloud Datacenters: A multi-objective discrete JAYA optimization. *Sustain. Comput. Inform. Syst.* **2023**, *38*, 100856. [CrossRef]
12. Gharehpasha, S.; Masdari, M.; Jafarian, A. Virtual machine placement in cloud data centers using a hybrid multi-verse optimization algorithm. *Artif. Intell. Rev.* **2020**, *54*, 2221–2257. [CrossRef]
13. Zhang, W.; Chen, X.W.; Jiang, J. A multi-objective optimization method of initial virtual machine fault-tolerant placement for star topological data centers of cloud systems. *Tsinghua Sci. Technol.* **2021**, 26, 95–111. [CrossRef]
14. Abohamama, A.S.; Hamouda, E. A hybrid energy-aware virtual machine placement algorithm for cloud environments. *Expert Syst. Appl.* **2020**, *150*, 113306. [CrossRef]
15. Alboaneen, D.A.; Tianfield, H.; Zhang, Y.; Pranggono, B. A metaheuristic method for joint task scheduling and virtual machine placement in cloud data centers. *Future Gener. Comput. Syst.* **2021**, *115*, 201–212. [CrossRef]
16. Liu, H.; Zhou, X.; Gao, K.; Ju, Y. An integrated optimization method to task scheduling and VM placement for green datacenters. *Simul. Model. Pract. Theory* **2024**, *135*, 102962. [CrossRef]
17. Wang, X.; Lou, H.; Dong, Z.; Yu, C.; Lu, R. Decomposition-based multi-objective evolutionary algorithm for virtual machine and task joint scheduling of cloud computing in data space. *Swarm Evol. Comput.* **2023**, *77*, 101230. [CrossRef]
18. Karmakar, K.; Das, R.K.; Khatua, S. An ACO-based multi-objective optimization for cooperating VM placement in cloud data center. *J. Supercomput.* **2021**, *78*, 3093–3121. [CrossRef]
19. Nagadevi; Raja, K. Multi-core Aware Virtual Machine Placement for Cloud Data Centers with Constraint Programming. In Proceedings of the Intelligent Computing: Proceedings of the 2021 Computing Conference, Virtually, 15–16 July 2021; Volume 1, pp. 439–457. [CrossRef]
20. Torre, E.; Durillo, J.J.; De Maio, V.; Agrawal, P.; Benedict, S.; Saurabh, N.; Prodan, R. A dynamic evolutionary multi-objective virtual machine placement heuristic for cloud data centers. *Inf. Softw. Technol.* **2020**, *128*, 106390. [CrossRef]
21. Zhang, X.; Meng, H.; Jiao, L. Improving PSO-Based Multiobjective Optimization Using Competition and Immunity Clonal. In *Computational Intelligence and Security*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 839–845. [CrossRef]
22. Nebro, A.J.; Durillo, J.J.; García-Nieto, J.; Coello Coello, C.A.; Luna, F.; Alba, E. SMPSO: A new PSO-based metaheuristic for multi-objective optimization. In Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM), Nashville, TN, USA, 30 March–2 April 2009; pp. 66–73. [CrossRef]
23. Deb, K.; Jain, H. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems with Box Constraints. *IEEE Trans. Evol. Comput.* **2014**, *18*, 577–601. [CrossRef]
24. Liu, R.; Yang, P.; Liu, J. A dynamic multi-objective optimization evolutionary algorithm for complex environmental changes. *Knowl.-Based Syst.* **2021**, *216*, 106612. [CrossRef]
25. Zhang, S.; Xie, J.; Wang, H. Fuzzy Adaptive NSGA-III for Large-Scale Optimization Problems. *Int. J. Fuzzy Syst.* **2022**, *24*, 1619–1633. [CrossRef]
26. Pineda, A.S.; Dorronsoro, B.; Nebro, A.J.; Durillo, J.J.; Castillo, O.; Fraire, H.J. A novel multi-objective evolutionary algorithm with fuzzy logic based adaptive selection of operators: FAME. *Inf. Sci.* **2019**, *471*, 233–251. [CrossRef]
27. Naik, R.B.; Singh, U. A review on applications of chaotic maps in pseudo-random number generators and encryption. *Ann. Data Sci.* **2024**, *11*, 25–50. [CrossRef]
28. Varol Altay, E.; Alatas, B. Bird swarm algorithms with chaotic mapping. *Artif. Intell. Rev.* **2020**, *53*, 1373–1414. [CrossRef]
29. Fan, J.; Li, Y.; Wang, T. An improved African vultures optimization algorithm based on tent chaotic mapping and time-varying mechanism. *PLoS ONE* **2021**, *16*, e0260725. [CrossRef] [PubMed]
30. Li, H.; Wang, D.; Zhou, M.; Fan, Y.; Xia, Y. Multi-Swarm Co-Evolution Based Hybrid Intelligent Optimization for Bi-Objective Multi-Workflow Scheduling in the Cloud. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 2183–2197. [CrossRef]
31. Zhang, R.; Zhou, J.; Ouyang, S.; Wang, X.; Zhang, H. Optimal operation of multi-reservoir system by multi-elite guide particle swarm optimization. *Int. J. Electr. Power Energy Syst.* **2013**, *48*, 58–68. [CrossRef]
32. Kong, D.; Chang, T.; Dai, W.; Wang, Q.; Sun, H. An improved artificial bee colony algorithm based on elite group guidance and combined breadth-depth search strategy. *Inf. Sci.* **2018**, *442*, 54–71. [CrossRef]
33. Mohanapriya, N.; Kousalya, G.; Balakrishnan, P.; Chelliah, P.R. Energy efficient workflow scheduling with virtual machine consolidation for green cloud computing. *J. Intell. Fuzzy Syst.* **2018**, *34*, 1561–1572. [CrossRef]

34. Zhao, H.; Wang, J.; Liu, F.; Wang, Q.; Zhang, W.; Zheng, Q. Power-Aware and Performance-Guaranteed Virtual Machine Placement in the Cloud. *IEEE Trans. Parallel Distrib. Syst.* **2018**, *29*, 1385–1400. [CrossRef]
35. Bilal; Millie, P.; Hira, Z.; Laura, G.H.; Ajith, A. Differential Evolution: A Review of More Than Two Decades of Research. *Eng. Appl. Artif. Intell.* **2020**, *90*, 103479. [CrossRef]
36. Zhang, Y.; Chen, G.; Cheng, L.; Wang, Q.; Li, Q. Methods to balance the exploration and exploitation in differential evolution from different scales: A survey. *Neurocomputing* **2023**, *561*, 126899. [CrossRef]
37. Yang, X.S.; Deb, S.; Fong, S. Metaheuristic algorithms: Optimal balance of intensification and diversification. *Appl. Math. Inf. Sci.* **2014**, *8*, 977. [CrossRef]
38. Srinivas, M.; Patnaik, L.M. Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms. *IEEE Trans. Syst. Man, Cybern.* **1994**, *24*, 656–667. [CrossRef]
39. Bi, X.; Wang, C. An improved NSGA-III algorithm based on objective space decomposition for many-objective optimization. *Soft Comput.* **2016**, *21*, 4269–4296. [CrossRef]
40. Van Veldhuizen, D.A. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*; Air Force Institute of Technology: Wright-Patterson Air Force Base, OH, USA, 1999.
41. Coello Coello, C.A.; Reyes Sierra, M. A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm. In Proceedings of the MICAI 2004: Advances in Artificial Intelligence: Third Mexican International Conference on Artificial Intelligence, Mexico City, Mexico, 26–30 April 2004; Proceedings 3; Springer: Berlin/Heidelberg, Germany, 2004; pp. 688–697. [CrossRef]
42. Zitzler, E.; Thiele, L.; Laumanns, M.; Fonseca, C.M.; da Fonseca, V.G. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Trans. Evol. Comput.* **2003**, *7*, 117–132. [CrossRef]
43. Ishibuchi, H.; Masuda, H.; Tanigaki, Y.; Nojima, Y. Modified distance calculation in generational distance and inverted generational distance. In Proceedings of the Evolutionary Multi-Criterion Optimization: 8th International Conference, EMO 2015, Guimarães, Portugal, 29 March–1 April 2015; Proceedings, Part II 8; Springer: Berlin/Heidelberg, Germany, 2015; pp. 110–125. [CrossRef]
44. Hardi, S.M.; Zarlis, M.; Effendi, S.; Lydia, M.S. Taxonomy Genetic Algorithm For Implementation Partially Mapped Crossover In Travelling Salesman Problem. *J. Phys. Conf. Ser.* **2020**, *1641*, 012104. [CrossRef]