applied sciences

MDPI

Article

Cloud Computing Considering Both Energy and Time Solved by Two-Objective Simplified Swarm Optimization

Wei-Chang Yeh [1], Wenbo Zhu [2,*], Ying Yin [1] and Chia-Ling Huang [3]

[1] Integration and Collaboration Laboratory, Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu 300, Taiwan
[2] School of Mechatronical Engineering and Automation, Foshan University, Foshan 528000, China
[3] Department of International Logistics and Transportation Management, Kainan University, Taoyuan 33857, Taiwan
* Correspondence: zhuwenbo@fosu.edu.cn

Abstract: Cloud computing is an operation carried out via networks to provide resources and information to end users according to their demands. The job scheduling in cloud computing, which is distributed across numerous resources for large-scale calculation and resolves the value, accessibility, reliability, and capability of cloud computing, is important because of the high development of technology and the many layers of application. An extended and revised study was developed in our last work, titled "Multi Objective Scheduling in Cloud Computing Using Multi-Objective Simplified Swarm Optimization MOSSO" in IEEE CEC 2018. More new algorithms, testing, and comparisons have been implemented to solve the bi-objective time-constrained task scheduling problem in a more efficient manner. The job scheduling in cloud computing, with objectives including energy consumption and computing time, is solved by the newer algorithm developed in this study. The developed algorithm, named two-objective simplified swarm optimization (tSSO), revises and improves the errors in the previous MOSSO algorithm, which ignores the fact that the number of temporary nondominated solutions is not always only one in the multi-objective problem, and some temporary nondominated solutions may not be temporary nondominated solutions in the next generation based on simplified swarm optimization (SSO). The experimental results implemented show that the developed tSSO performs better than the best-known algorithms, including nondominated sorting genetic algorithm II (NSGA-II), multi-objective particle swarm optimization (MOPSO), and MOSSO in the convergence, diversity, number of obtained temporary nondominated solutions, and the number of obtained real nondominated solutions. The developed tSSO accomplishes the objective of this study, as proven by the experiments.

Keywords: energy; computing time; two-objective simplified swarm optimization (tSSO); cloud computing; job scheduling

Citation: Yeh, W.-C.; Zhu, W.; Yin, Y.; Huang, C.-L. Cloud Computing Considering Both Energy and Time Solved by Two-Objective Simplified Swarm Optimization. *Appl. Sci.* **2023**, *13*, 2077. https://doi.org/10.3390/app13042077

Academic Editor: Jose Machado

Received: 31 October 2022
Revised: 18 November 2022
Accepted: 22 November 2022
Published: 6 February 2023

1. Introduction

The computing tasks of cloud computing, which play a very important role nowadays due to the high development of technology and the many layers of application, resulting in the increasing application and demand for cloud computing, involve the delivery of on-demand computing resources ranging from applications to remote data centers over the internet on a pay-for-use basis. Therefore, many scholars and practitioners have devoted their efforts to strengthening or innovating this related research. Section 2 summarizes the existing literature and demonstrates how this work differs in its approach.

In essence, the computing tasks of cloud computing, which are distributed across numerous resources for large-scale calculation and resolve the value, accessibility, reliability, and capability of cloud computing, comprise a computing style in which dynamically scalable and often virtualized resources are provided as an Internet service [1]. The service

model of computing tasks includes cloud platforms, users, applications, virtual machines, etc. Within each cloud platform, there are multiple platform users, each with the ability to run multiple applications on the platform. Each application corresponds to the jobs requested by a user and uses a certain quota of virtual machines, through which the application finishes and returns the jobs, thus completing the procedure.

Countless discussions and research have been conducted on the two prevailing issues in both grid and cloud computing: resource allocation and job scheduling [2–4]. The job scheduling problem revolves around exploring how the provider of cloud computing services assigns the jobs of each client to each processor according to certain rules and regulations to ensure the cost-effectiveness of the job scheduling process. The efficiency and performance of cloud computing services are usually associated with the efficiency of job scheduling, which affects not only the performance efficiency of users' jobs but also the utilization efficiency of system resources. Hence, the interdependent relationship between the efficiency and performance of cloud computing services and the efficiency of job scheduling necessitates research on the job scheduling problem of the computing tasks of cloud computing.

The job scheduling problem of the computing tasks of cloud computing services assign jobs to individual processors. It is an NP-hard combinatorial problem, which renders it difficult to obtain the global optima within polynomial time. The greater the problem size, e.g., the number of resources or jobs, the more difficult it is for traditional job scheduling algorithms to solve. Hence, alongside improving traditional algorithms, many scholars have introduced machine learning algorithms, e.g., the Pareto-set cluster genetic algorithm [2] and particle swarm optimization [2], the binary-code genetic algorithm [3] and integer-code particle swarm optimization [3], the simulated annealing algorithm [4], particle swarm optimization (PSO) [5], the genetic algorithm [6], the machine learning algorithm [7], Multi-Objective PSO [8], the artificial bee colony algorithm [9], the hybrid optimization algorithm [10], etc., to solve the job scheduling problem of the computing tasks of cloud computing.

Numerous different objectives have been discussed to measure service performance, e.g., cost, reliability, makspan, and power consumption [1–10]. However, when conducting research on the job assignment problem, most scholars emphasize the single-objective job scheduling problem and overlook job scheduling problems with more than one objective [11–17]. By doing so, they fail to acknowledge other goals that may influence the quality of the cloud-computing service. It is thus necessary to balance various aspects when evaluating job scheduling problems [15–17]; for example, the objective of minimizing total energy consumption and the objective of minimizing makspan are conflicting objectives in real-life applications [17].

With the ever-advancing development of cloud computing, more and more data centers have successively been established to run a large number of applications that require considerable computations and storage capacity, but this wastes vast amounts of energy [12,15–17]. Reducing power consumption and cutting down energy costs has become a primary concern for today's data center operators. Thus, striking a balance between reducing energy consumption and maintaining high computation capacity has become a timely and important challenge.

Furthermore, with more media and public attention shifting onto progressively severe environmental issues, governments worldwide have now adopted a stronger environmental protection stance [12,15–17]. This puts increasing pressure on enterprises to pursue higher output and also to focus on minimizing power consumption [12,15–17].

Stemming from real-life concerns, as mentioned above, our previous work considered a two-objective time-constrained job scheduling problem to measure the performance of a certain job schedule plan with two objectives: the quality of the cloud computing service in terms of the makspan and the environmental problem based on the energy consumption [12,15–17].

There are two different types of algorithms used to solve multi-objective problems. While one converts multi-objective problems to ones that are single objective in nature through methods such as ε-constraint, LP-metrics, goal programming, etc., the other solves multi-objective problems based on the concept of Pareto optimality. The latter is the one we adapted, both here and in our previous study [10–17].

Moreover, a new algorithm called the multi-objective simplified swarm optimization (MOSSO) was proposed to solve the above problem [17]. However, there are some errors in the MOSSO source code, which ignores the multi-objective problem; the number of temporary nondominated solutions is not always only one, and some temporary nondominated solutions may not be temporary nondominated solutions in the next generation. Therefore, the performance comparison is limited to MOSSO and MOPSO, despite both having an iterative local search [17].

In order to rectify our previous source code with new concepts based on the Pareto optimality to solve the two-objective time-constrained job scheduling, we proposed a new algorithm called two-objective simplified swarm optimization (tSSO). It draws from the SSO update mechanism to generate offspring [18], the crowding distance to rank nondominated solutions [19], and new hybrid elite selection to select parents [20], and the limited number of nondominated solutions adapted from multi-objective particle swarm optimization (MOPSO) serves to guide the update [13].

The motivation and contribution of this work are highlighted as follows:

1. An improved algorithm named two-objective simplified swarm optimization (tSSO) is developed in this work to revise and improve errors in the previous MOSSO algorithm [17], which ignores the fact that the number of temporary nondominated solutions is not always only one in the multi-objective problem, and some temporary nondominated solutions may not be temporary nondominated solutions in the next generation. The algorithm is based on SSO to deliver the job scheduling in cloud computing.
2. More new algorithms, testing, and comparisons have been implemented to solve the bi-objective time-constrained task scheduling problem in a more efficient manner.
3. In the experiments conducted, the proposed tSSO outperforms existing established algorithms, e.g., NSGA-II, MOPSO, and MOSSO, in the convergence, diversity, number of obtained temporary nondominated solutions, and the number of obtained real nondominated solutions.

The remainder of this paper is organized as follows. Related work in the existing literature is discussed in Section 2, which also demonstrates how this work is different in its approach. Section 3 presents notations, assumptions, and the mathematical modeling of the proposed two-objective time-constrained job scheduling problem to address energy consumption and service quality in terms of the makspan. Section 4 introduces the simplified swarm optimization (SSO) [18], the concept of crowd distance [19], and traditional elite selection [20]. The proposed tSSO is presented in Section 5, together with the discussion of its novelties and pseudo code. The section also explains the errors in the previous MOSSO algorithm and how the proposed new algorithm overcomes them. Section 6 compares the proposed tSSO in terms of nine different parameter settings with the MOSSO proposed in [17], the MOPSO proposed in [13], and the famous NSGA-II [19,20]. Three benchmark problems, one small-size, one medium-size, and one large-size, are utilized and analyzed from the viewpoint of convergence, diversity, and number of obtained nondominated solutions in order to demonstrate the performance of the tSSO. Our conclusions are given in Section 7.

2. Literature Review

The related work of the existing literature is provided in this section, which also demonstrates how this work is different in its approach. The job scheduling problem of cloud-computing services is very important, meaning that there is a significant amount of related research on it, which can be classified into the following types.

2.1. A Review of Job Scheduling in Cloud-Computing

Houssein et al. presented a review of numerous meta-heuristics for solving job scheduling in cloud-computing [21], and a literature review of the related methods used for solving job scheduling in cloud-computing has been conducted by Arunarani et al. [22] and Kumar et al. [23].

2.2. Algorithm Research for Solving the Single-Objective Problem for Job Scheduling in Cloud-Computing

Chen et al. proposed an improved meta-heuristic whale optimization algorithm to solve resource utilization for job scheduling in cloud-computing [24]. Attiya et al. studied improved simulated annealing (SA) to optimize makspan for job scheduling in cloud-computing [25]. Gąsior and Seredyński worked on a distributed security problem for job scheduling in cloud computing by a method based on the game theoretic approach [26]. Mansouri and Javidi resolved response time using a cost-based job scheduling method [27]. A deep reinforcement learning (DRL) algorithm was proposed by Cheng et al. [28] to reduce the cost of job scheduling in cloud computing.

2.3. Algorithm Research for Solving the Multi-Objective Problem for Job Scheduling in Cloud-Computing

Processing time and cost of job scheduling in cloud computing were optimized by Shukri et al. [29] using an improved multi-verse optimizer. Jacob and Pradeep minimized the multi-objective problem, including makspan, cost, and deadline violation rate, using a hybrid algorithm of cuckoo search (CS) and particle swarm optimization (PSO) [30]. Abualigah and Diabat optimized the makspan and resource utilization using an improved algorithm based on the elite-based differential evolution method [31]. Sanaj and Prathap proposed a chaotic squirrel search algorithm (CSSA) to solve the makspan and cost issues [32], and Abualigah and Alkhrabsheh optimized the cost and service availability in the multi-objective problem [33].

This work aims to optimize a two-objective problem that includes both energy and makspan, but the objectives of this work are different from the multi-objective contents of the existing literatures, as shown in the third part of the literature review [29–33]. Furthermore, this work proposes an improved algorithm to solve the two-objective problem for job scheduling in cloud computing, which is obviously different from the first and second parts of the existing literatures above [21–28].

3. Assumptions, and Mathematical Problem Description

The formal mathematical model of the two-objective constrained job scheduling problem of cloud-computing services [12,15–17] is described as follows, together with the assumptions used in the problem, as well as the algorithms.

3.1. Assumptions

In cloud-computing services, all requests from cloud-computing users are collected as jobs and subsequently divided into multiple jobs, which are then executed by data centers with multiple processors [2–4,10–12,15–17]. To simplify our model without foregoing generality, the following assumptions are utilized to prompt the two-objective constrained job scheduling problem to focus on the energy consumption and makspan of the cloud-computing service [2–4,10–12,15–17].

1. All jobs are available, independent, and equal in importance with two attributes, $size_i$ and $time_i$, for $i = 1, 2, \ldots, N_{var}$ for processing simultaneously at time zero.
2. Each processor is available at any time with two attributes, denoted as $speed_j$ and $power_j$, $j = 1, 2, \ldots, N_{cpu}$, and cannot process two or more jobs simultaneously.
3. All processing times include set-up time.
4. Job pre-emption and the splitting of jobs are not permitted.
5. There is an infinite buffer between the processors.

3.2. The Mathematical Model

A feasible job scheduling solution is a plan that assigns jobs to processors in a sequence such that the makspan, which is the time at which the final job is complete before a predefined limited time. Let solution $X = (x_1, x_2, \ldots, x_{\mathrm{Nvar}})$ be a feasible job scheduling plan and x_i be the type of processor assigned to the job i for $i = 1, 2, \ldots, \mathrm{N_{var}}$.

To maintain efficiency and ensure environmental protection, the two-objective constrained job scheduling problem considered here is to assign $\mathrm{N_{var}}$ jobs to $\mathrm{N_{cpu}}$ processors to minimize both the energy consumption and makspan so that the makspan is not overdue [17]:

$$Min\ F_{\mathrm{e}}(X) = \sum_{i=1}^{n} \left(t_{i,x_i} \cdot e_{x_i}\right) \tag{1}$$

$$Min\ F_{\mathrm{m}}(X) = \underset{j}{Max} \sum_{i=1}^{n} t_{i,j} \tag{2}$$

$$\text{s.t. } F_{\mathrm{m}}(X) \leq T_{\mathrm{ub}}. \tag{3}$$

Equation (1) is the total energy consumption of a job scheduling plan, and it is the sum of the usage of energy of all jobs assigned to processors based on X. Note that the energy consumption of each job is the product of the power cost per unit time and the running time is equal to the size of the job divided by the execution speed, as shown in formula (4).

$$t_{i,j} = \begin{cases} size_i / speed_j & \text{if task } i \text{ is processed on processor } j \\ 0 & \text{otherwise} \end{cases}. \tag{4}$$

Equation (2) is the makspan, which is the time the last job is finished. Equation (3) is the time constraint for each job, such that each job must be finished before the deadline $\mathrm{T_{ub}}$.

4. SSO, Crowding Distance, and Elite Selection

The proposed tSSO, based on SSO [18], is used to update solutions from generation to generation intelligently; the crowding distance [19] is used to rank temporary nondominated solutions systematically, and the elite selection [34] is used to select solutions to act as parents. Hence, SSO, crowding distance, and elite selection are introduced briefly in this section.

4.1. Simplified Swarm Optimization

SSO, developed by Yeh in 2009 [18], is a simple but powerful machine leaning algorithm that is a hybrid of leader-solution swarm intelligence and population-based evolutionary computation.

In traditional SSO, all variables need to be updated (called the all-variable update in SSO) such that the jth variable of the ith solution (i.e., $x_{i,j}$) is obtained from either the jth variable of the P_{gBest} (i.e., $p_{gBest,j}$) with probability c_g of the P_i with probability c_p of its current value (i.e., $x_{i,j}$) with probability c_w of a randomly generated feasible new value (say x) with probability c_r, where the P_{gBest} is the best solution among all existing solutions; P_i is the best ith solution in its evolutionary history, and $c_g + c_p + c_w + c_r = 1$.

The above update mechanism of SSO is very simple, efficient, and flexible [17,18,35–40], and can be presented as a stepwise-function update:

$$x_{i,j} = \begin{cases} p_{gBest,j} & \text{if } \rho_{[0,1]} \in [0, C_g) \\ p_{i,j} & \text{if } \rho_{[0,1]} \in [C_g, C_p) \\ x_{i,j} & \text{if } \rho_{[0,1]} \in [C_p, C_w) \\ x & \text{if } \rho_{[0,1]} \in [C_w,\ 1] \end{cases}, \tag{5}$$

where $C_g = c_g$, $C_p = C_g + c_p$, and $C_w = C_p + c_w$.

The SSO pseudo code is provided below [17,18,35–39]:

STEP S0. Generate X_i randomly, find $gBest$, and let $t = 1$, $k = 1$, and $P_i = X_i$ for $i = 1, 2, \ldots, N_{sol}$.
STEP S1. Update X_k.
STEP S2. If $F(X_k)$ is better than $F(P_k)$, let $P_k = X_k$. Otherwise, go to STEP S5.
STEP S3. If $F(P_k)$ is better than $F(P_{gBest})$, let $gBest = k$.
STEP S4. If $k < N_{sol}$, let $k = k + 1$ and go to STEP S1.
STEP S5. If $t < N_{gen}$, let $t = t + 1$, $k = 1$, and go to STEP S1. Otherwise, halt.

The stepwise-function update mechanism is very simple and efficient, but it is also very powerful and has various successful applications, e.g., the redundancy allocation problem [35,36], flexible grid trading [37], the disassembly sequencing problem [38], artificial neural networks [39], power systems [40], energy problems [41,42], and various other problems [17,18,43–52]. Moreover, the stepwise-function update mechanism is easier to customize by replacing any item of its stepwise function either with other algorithms [36,39] or hybrid algorithms [41], in sequence or in parallel [42].

4.2. Crowding Distance

Let

$$\widehat{d}_{l,i} = Min\{\frac{F_l(X_i) - F_l(X_j)}{Max(F_l) - Min(F_l)}\} \tag{6}$$

be the shortest normalized Euclidean distance between the ith temporary nondominated solution and any other temporary nondominated solutions based on the lth objective function, where $Min(F_l)$ is the minimal value of the lth objective function and $Max(F_l)$ is the maximal value of the lth objective function.

The crowding distance is the sum of the individual distance and can be calculated as follows [19]:

$$\sum_i \sqrt{\widehat{d}^2_{1,i} + \widehat{d}^2_{2,i}}, \tag{7}$$

for all temporary nondominated solutions, X_i. The crowding distance is only used to rank temporary nondominated solutions for selecting as parents if the total number of determine are over a limited value, which is defined as N_{non} and $N_{non} = N_{sol}$ in this study.

4.3. The Elite Selection

Among numerous different selection policies, the elite selection is the simplest and is thus adopted in the proposed tSSO. The elite selection chooses the best solutions in the current generation to generate and update solutions for the next generation [34]. For example, let $N_{sol} = 50$, X_1, X_2, ... , X_{100} be the solutions required for selection. Elite selection ranks and selects the best 500 solutions among X_1, X_2, ... , X_{100} and renumbers these selected solutions as X_1, X_2, ... , X_{50}.

In the tSSO, these best solutions are temporary nondominated solutions. Due to the fact that the number of temporary nondominated solutions may be less than or larger than the number of parents, a new elite selection called the hybrid elite selection is developed and used in the proposed tSSO, and the details are discussed in Section 5.3.

5. Proposed Algorithm

The proposed tSSO is a population-based, all-variable update, and it is a stepwise function-based method, i.e., there are a fixed number of solutions in each generation and all variables must be updated based on the stepwise function for each solution. Details of the proposed tSSO are discussed in this section.

5.1. Purpose of tSSO

The developed tSSO algorithm revises and improves the errors in the previous MOSSO algorithm, which ignores the fact that the number of temporary nondominated solutions in

the multi-objective problem is not always only one, and some temporary nondominated solutions may not be temporary nondominated solutions in the next generation.

The developed tSSO algorithm overcomes the errors in the previous MOSSO algorithm by the following methods:

1. The novel update mechanism of the role of *gBest* and *pBest* in the proposed tSSO, which are introduced in Section 5.3.
2. The hybrid elite selection in the proposed tSSO, which is introduced in Section 5.4.

5.2. Solution Structure

The first step of most machine learning algorithms is to define the solution structure [2–4,35–45]. The solution in the tSSO for the proposed problem is defined as a vector, and the value of the ith coordinate is the processor utilized to process the ith job. For example, let X_5 = (3, 4, 6, 4) be the 5th solution in the 10th generation of the problem with 4 jobs. In X_5, jobs 1, 2, 3, and 4 are processed by processors 3, 4, 6, and 4.

5.3. Novel Update Mechanism

The second step in developing machine learning algorithms is to create an update mechanism to update solutions [2–4,35–42]. The stepwise update function is a unique update mechanism of SSO [17,18,35–42]. In the single-objective problem, there is only one *gBest* for the traditional SSO. However, in the multi-objective problem, the number of temporary nondominated solutions is not always only one, and some temporary nondominated solutions may not be temporary nondominated solutions in the next generation [17]. Note that a nondominated solution is not dominated by any other solution, while a temporary nondominated solution is a solution nondominated by other found solutions that are temporary in the current generation, and it may be dominated by other updated solutions later [13,17,19].

Hence, in the proposed tSSO, the role of *gBest* is removed completely and the *pBest* used in the tSSO is not the original definition in the SSO. The *pBest* for each solution in the proposed tSSO is selected from temporary nondominated solutions, i.e., there is no need to follow its previous best predecessor. The stepwise update function used in the proposed tSSO is listed below for multi-objective problems for each solution X_i, with i = 1, 2, . . . , N_{sol}:

$$x_{i,j} = \begin{cases} x_i^* & \text{if } \rho_{[0,1]} \in [0, C_p) \\ x_{i,j} & \text{if } \rho_{[0,1]} \in [C_p, C_w) \\ x & \text{otherwise} \end{cases}, \quad (8)$$

where $X^* = (x_1^*, x_2^*, \ldots, x_k^*)$ is one of the temporary nondominated solutions selected randomly, $\rho_{[0,1]}$ is a random number generated uniformly in [0, 1], and x is a random integer generated from 0, 1, 2, . . . , N_{var}.

For example, let X_6 = (1, 2, 3, 2, 4) and X^* = (2, 1, 4, 3, 3) be a temporary nondominated solution selected randomly. Let C_p = 0.50, C_w = 0.95, and $\rho = (\rho_1, \rho_2, \rho_3, \rho_4, \rho_5)$ = (0.32, 0.75, 0.47, 0.99, 0.23). The procedure to update X_6 based on the stepwise function provided in Equation (8) is demonstrated in Table 1.

Table 1. Example of the update process in the proposed tSSO.

Variable	1	2	3	4	5
X_5	1	**2**	3	2	4
X^*	**2**	1	**4**	3	**3**
ρ	0.32	0.75	0.47	0.99	0.23
New X_5	2	2	4	4 #	3

"#" indicates that the corresponding feasible value is generated randomly.

From the above example, the simplicity, convenience, and efficiency of the SSO can also be found in the update mechanism of the proposed tSSO.

5.4. Hybrid Elite Selection

The last step is to determine the selection policy to decide which solutions, i.e., parents, are selected to generate solutions in the next generation. In the proposed tSSO, hybrid selection is harnessed.

Let π_t be a set to store the temporary nondominated solutions found in the tth generation. It is impossible to have all temporary nondominated solutions because their number is infinite, and also because temporary nondominated solutions may not be real nondominated solutions. The value of $|\pi_t|$ is limited to N_{sol}, and parts of temporary nondominated solutions are abandoned to keep $|\pi_t| = N_{sol}$.

If $N_{sol} \leq |\pi_t|$, the crowding distances need to be calculated for each temporary nondominated solution, and only the best N_{sol} solutions will be selected from π_t to be parents. However, all temporary nondominated solutions in π_t are used to serve as parents and $(N_{sol} - |\pi_t|)$ solutions are selected randomly from offspring if $N_{sol} > |\pi_t|$. As discussed above, there are always N_{sol} parents for each generation, and these temporary nondominated solutions are usually chosen first to serve their role as parents.

Note that these temporary nondominated solutions are abandoned if they are not selected as parents.

5.5. Group Comparison

Referring to Section 4.3, the temporary nondominated solutions play a paramount role in most multi-objective algorithms. Up to now, the most popular method to achieve the above goal is pairwise comparison, which takes $O(N^2)$ [13,17,19] for each solution in each generation, where N is the number of solutions from which we determine temporary nondominated solutions. Hence, the corresponding related time complexity of the tSSO and NSGA-II are both $O(4N_{sol}{}^2)$. Therefore, the most time-consuming aspect of solving multi-objective problems using machine learning algorithms is the search for all temporary nondominated solutions from all offspring.

In the proposed tSSO, the parents are selected from temporary nondominated solutions found in the previous generation and offspring generated in the current generation. To reduce the computation burden, a new method called group comparison is proposed in tSSO. All temporary nondominated solutions in offspring are found first using the pairwise comparison, which takes $O(N_{sol}{}^2)$ as the number of offspring in N_{sol}. The temporary nondominated solutions obtained in the previous generations are then compared with the new temporary nondominated solutions found in the current generation. The number of both sets of temporary nondominated solutions are at most N_{sol}, i.e., the time complexity is $O(N_{sol} \log(N_{sol}))$ based on the merge sort.

Hence, the time complexity is $O(N_{sol} \log(N_{sol})) + O(N_{sol}{}^2) = O(N_{sol}{}^2)$, which is only one quarter of the pairwise comparison.

5.6. Proposed tSSO

The procedure of the proposed tSSO, which has nine different parameter settings, denoted as $tSSO_0$, $tSSO_1$, $tSSO_2$, $tSSO_3$, $tSSO_4$, $tSSO_5$, $tSSO_6$, $tSSO_7$, and $tSSO_8$, has a detailed introduction in Section 6, based on the solution structure, update mechanism, hybrid elite selection, and group comparison discussed in this section, which are presented in pseudo code as follows.

PROCEDURE tSSO

STEP 0. Create initial population X_i randomly for $i = 1, 2, \ldots, N_{sol}$ and let $t = 2$.

STEP 1. Let π_t be all temporary nondominated solutions in $S^* = \{X_k \mid \text{for } k = 1, 2, \ldots, 2N_{sol}\}$ and X_{Nsol+k} is updated from X_k based on Equation (8) for $k = 1, 2, \ldots, N_{sol}$.

STEP 2. If $N_{sol} \leq |\pi_t|$, let $S = \{\text{top } N_{sol} \text{ solutions in } \pi_t \text{ based on crowding distances}\}$ and go to STEP 4.

STEP 3. Let $S = \pi_t \cup \{N_{sol} - |\pi_t| \text{ solutions selected randomly from } S^* - \pi_t\}$.
STEP 4. Re-index these solutions in S such that $S = \{X_k \mid \text{for } k = 1, 2, \ldots, N_{sol}\}$.
STEP 5. If $t < N_{gen}$, then let $t = t + 1$ and go back to STEP 1. Otherwise, halt.

6. Simulation Results and Discussion

For fair comparison with MOSSO [17], which was the previous work of the extended topic, the numerous experiments of the parameter-setting procedure sof the three different-sized benchmark problems used in [17] were carried out using tSSO. The experimental results obtained by the proposed tSSO were compared with those obtained using MOSSO [17], which were acquired using MOPSO [13] and NSGA-II [19,20].

6.1. Parameter-Settings

All machine learning algorithms have parameters in their update procedures and/or the selection procedure. Thus, there is a need to tune parameters for better results. To determine the best parameters of the proposed tSSO, three different levels of the two factors of c_p and c_w, the low value of 0.1, the middle value of 0.3, and the high value of 0.5, have been combined, e.g., nine different parameter settings denoted as $tSSO_0$, $tSSO_1$, $tSSO_2$, $tSSO_3$, $tSSO_4$, $tSSO_5$, $tSSO_6$, $tSSO_7$, and $tSSO_8$, as shown in Table 2.

Table 2. Nine different parameter setting of the proposed tSSO.

ID	$tSSO_0$	$tSSO_1$	$tSSO_2$	$tSSO_3$	$tSSO_4$	$tSSO_5$	$tSSO_6$	$tSSO_7$	$tSSO_8$
C_p	0.1	0.1	0.1	0.3	0.3	0.3	0.5	0.5	0.5
C_w	0.2	0.4	0.6	0.4	0.6	0.8	0.6	0.8	1.0

Note that $C_p = c_p$ and $C_w = c_p + c_w$. The following provided the parameter settings for the other algorithms:

NSGA-II: $c_{crossover} = 0.7$, $c_{mutation} = 0.3$

MOPSO: $w = 0.871111$, $c_1 = 1.496180$, $c_2 = 1.496180$ [17]

MOSSO: $C_g = 0.1 + 0.3t/N_{gen}$, $C_p = 0.3 + 0.4t/N_{gen}$, $C_w = 0.4 + 0.5t/N_{gen}$, where t is the current generation number [17].

6.2. Experimental Environments

To demonstrate the performance of the proposed tSSO and select the best parameter settings, tSSOs with nine parameter settings were utilized for three job scheduling benchmarks [17], namely $(N_{job}, N_{cpu}) = (20, 5)$, (50, 10), and (100, 20), and the deadlines were all set as 30 for each test [17].

The following lists the special characteristics of the processor speeds (in MIPs), the energy consumptions (in KW/per unit time), and the job sizes (in MI) in these three benchmark problems [17]:

1. Each processor speed is generated between 1000 and 10,000 (MIPs) randomly, and the largest speed is ten times the smallest one.
2. The power consumptions grow polynomial as the speed of processors grow, and the value range is between 0.3 and 30 (KW) per unit time.
3. The values of job sizes are between 5000 and 15,000 (MIs).

Alongside the proposed tSSO with nine parameter settings, there are three other multi-objective algorithms; MOPSO [13,17], MOSSO [17], and NSGA-II [19,20] were tested and compared to further validate the superiority of the proposed tSSO. NSGA-II [19,20] is the current best multi-objective algorithm and is based on genetic algorithms, while MOPSO is based on particle swarm optimization [13], and MOSSO on SSO [17].

The proposed tSSO with its nine parameter settings and NSGA-II have no iterative local search to improve the solution quality. To ensure a fair comparison, the iterative local search is removed from both MOPSO and MOSSO. All algorithms were coded in DEV

C++ on a 64-bit Windows 10 PC, implemented on an Intel Core i7-6650U CPU @ 2.20 GHz notebook with 16 GB of memory.

In addition, for a fair comparison between all algorithms, $N_{sol} = N_{non} = 50$, $N_{gen} = 1000$, and $N_{run} = 500$, i.e., the same solution number, generation number, size of external repository, and run number for each benchmark problem were used. Furthermore, the calculation number of the fitness function of all algorithms was limited to $N_{sol} \times N_{gen}$ = 50,000, 100,000, and 150,000 in each run for the small-size, medium-size, and large-size benchmarks, respectively.

Note that the reason for the large value of N_{run} is to simulate the Pareto front, and the details are discussed in Section 6.3.

6.3. Performance Metrics

The convergence metrics and diversity metrics are always both used to evaluate the performances of the multi-objective algorithms in the solution quality. Among these metrics, the general distance (GD), introduced by Van Veldhuizen et al. [53], and spacing (SP), introduced by Schott [54], are two general indexes for the convergence metrics and diversity metrics, respectively. Let d_i be the shortest Euclidean distance between the ith temporary nondominated solution and the Pareto front, and $\underline{d}$ be the average sum of all d_i for all i. The GD and SP are defined below:

$$GD = \frac{\sqrt{\sum_{i=1}^{N_{sol}} {d_i}^2}}{N_{sol}}, \tag{9}$$

$$SP = .\sqrt{\frac{\sum_{i=1}^{N_{sol}} (\underline{d} - d_i)^2}{N_{sol} - 1}}. \tag{10}$$

The GD is the average of the sum of the squares of d_i, and the SP is very similar to the standard deviation in probability theory and statistics. If all temporary nondominated solutions are real nondominated solutions, we have GD = 0. The solutions are equality far from $\underline{d}$, we have SP = 0. Hence, in general, the smaller the SP is, the higher the diversity of solutions along the Pareto front and the better the solution quality becomes [13,17,19,46,47].

The Pareto front is needed for both the GD and SP in calculating from their formulas [43,44]. Unfortunately, it requires infinite nondominated solutions to form the Pareto front, and this is impossible to accomplish, even with exhaustive methods for the job scheduling problem of cloud computing, which is an NP-hard problem that has no guarantee of having the ability to obtain the global optimal solution under the complexity of polynomial time, and the traditional algorithms are intractable, resulting to a special optimization algorithm having to be used to reduce the system search complexity and improve the overall search quality [12,15–17]. Rather than a real Pareto front, a simulated Pareto front is implemented by collecting all temporary nondominated solutions in the final generation from all different algorithm with different values of N_{sol} for the same problem.

There are 12 algorithms with three different N_{sol} = 50, 100, and 150 for N_{run} = 500, i.e., $12 \times 500 \times (50 + 100 + 150)$ = 1,800,000 solutions obtained at the end for each test problem. All temporary nondominated solutions are found from these 1,800,000 final solutions to create a simulated Pareto front to calculate the GD and SP.

6.4. Numerical Results

All numerical results attained in the experiments are listed in this section. Tables 3–5 list the averages and standard deviations of the obtained number of temporary nondominated solutions (denoted by N_n), the obtained number of nondominated solutions in the Pareto front (denoted by N_p), the converge metric GD, and the diversity metric SP, the required run time, the energy consumption values, and the makspan values for three

different-sized benchmark problems with N_{sol} = 50, 100, and 150, respectively. In these tables, the best of all the algorithms is indicated in bold, and the proposed BSSO with its nine parameter settings are denoted as $BSSO_{0–8}$, respectively, in Tables 3–5.

From Tables 3–5, we can make the following general observations:

1. The lower the c_r value, the better the performance. In the small-size problem, the proposed $tSSO_7$ with c_p = 0.5 and c_w = 0.3 is the best among all 12 algorithms. However, the proposed $tSSO_8$ with c_p = 0.5 and c_w = 0.5 is the best one for the medium- and large-size problems. The reason for this is that the number of real nondominated solutions is infinite. Even though in the update mechanism of the proposed tSSO when c_r = 0 there is only an exchange of information between the current solution itself and one of selected temporary nondominated solutions, it is already able to update the current solution to a better solution without needing any random movement.
2. The larger the size of the problem, e.g., N_{job}, the fewer the number of obtained nondominated solutions, e.g., N_n and N_p. There are two reasons why this is the case: (1) due to the characteristic of the NP-hard problems, i.e., the larger the size of the NP-hard problem, the more difficult it is to solve; (2) it is more difficult to find nondominated solutions for larger problems with the same deadline of 30 for all problems.
3. The larger the N_{sol}, the more likely it is to find more nondominated solutions, i.e., the larger N_n and N_p for the best algorithm among these algorithms no matter the size of the problem. Hence, it is an effective method to have a larger value of N_{sol} if we intend to find more nondominated solutions.
4. The smaller the N_{job}, the fewer the number of obtained nondominated solutions, e.g., There are two reasons for the above: (1) due to the characteristic of the NP-hard problems, i.e., the larger the size of the NP-hard problem, the more difficult it is to solve; (2) it is more difficult to find nondominated solutions for larger problems with the same deadline, which is set 30 for all problems.
5. The smaller the value of N_p, the shorter the run time. The most time-consuming aspect of finding nondominated solutions is filtering out these nondominated solutions from current solutions. Hence, a new method, called the group comparison, is proposed in this study to find these nondominated solutions from current solutions. However, even the proposed group comparison is more efficient than the traditional pairwise comparison on average, but it still needs $O(N_p{}^2)$ to achieve the goal.
6. There is no special pattern in solution qualities, e.g., the value of GD, SP, N_n, and N_p, from the final average values of the energy consumption and the makspan.
7. The one with the better number of obtained nondominated solutions also has better DP and SP values.
8. The MOPSO [13] and the original MOSSO [17] share one common factor: each solution must inherit and update based on its predecessor (parent) and its *pBest*, and this is the main reason that it is less likely to find new nondominated solutions. The above observation is coincident to that observed in item 1. Hence, the proposed tSSO and the NSGA-II [19,20] are much better than MOSSO [17] and MOPSO [13] in solution quality.

In general, the proposed tSSO without c_r has a more satisfying performance in all aspect of measures.

Table 3. Result for small-size problem.

N_{sol}	Alg	Avg (N_n)	Std (N_n)	Avg (N_p)	Std (N_p)	Avg (GD)	Std (GD)	Avg (SP)	Std (SP)	Avg (T)	Std (T)	Avg (F_1)	Std (F_1)	Avg (F_2)	Std (F_2)
50	$tSSO_0$	48.34	2.22	0.012	0.109	0.565	0.611	3.527	4.343	2.8954	0.5839	11,937.28	261.3206	2539.957	4798.1386
	$tSSO_1$	49.634	0.904	0.018	0.133	0.26	0.25	1.475	1.781	3.4103	0.6708	11,689.93	272.5046	1015.790	1926.8740
	$tSSO_2$	49.966	0.202	0.058	0.234	0.138	**0.059**	0.722	**0.445**	4.1789	0.6901	11,185.89	273.4064	838.2936	632.7848
	$tSSO_3$	49.494	1.165	0.012	0.109	0.311	0.349	1.819	2.495	3.2951	0.6579	11,766.60	269.7630	1208.2600	2270.2913
	$tSSO_4$	49.936	0.276	0.034	0.202	0.17	0.073	0.915	0.547	3.9837	0.6919	11,317.03	289.0288	**779.7255**	448.9989
	$tSSO_5$	**50**	**0**	**0.284**	0.544	**0.118**	0.065	**0.71**	0.478	4.7604	0.7535	10,603.56	229.7030	938.6334	**22.5991**
	$tSSO_6$	49.968	0.208	0.054	0.235	0.153	0.101	0.821	0.734	4.2185	0.6928	11,230.14	290.7808	817.7619	631.6859
	$tSSO_7$	**50**	**0**	0.218	0.464	0.121	0.064	0.722	0.475	**4.86**	0.7531	10,561.73	236.0619	936.6982	24.2198
	$tSSO_8$	49.7	0.766	0.188	0.457	0.153	0.103	0.776	0.598	4.4077	0.8044	10,088.10	550.3918	1010.433	888.7098
	MOPSO	9.274	2.026	0	**0**	2.947	0.908	17.366	6.314	3.1238	0.5445	12,176.37	230.151	29,378.78	16,620.2163
	MOSSO	1.23	0.508	0	**0**	9.914	0.122	8.417	4.978	1.2282	0.187	**9833.03**	**24.8444**	488,648.8	10,786.0609
	NSGA-II	17.22	3.216	0	**0**	2.213	1.27	13.107	8.676	0.0102	**0.0202**	11,963.00	487.9212	19,913.23	20,625.1095
100	$tSSO_0$	61.856	5.503	0.024	0.153	1.996	0.502	18.548	4.721	10.3052	1.4032	24,162.67	365.2535	48,016.98	20,979.7233
	$tSSO_1$	71.016	5.808	0.05	0.227	1.284	0.506	12.06	4.968	10.2902	1.3693	23,633.28	438.6975	24,077.66	15,460.6944
	$tSSO_2$	95.07	4.779	0.184	0.463	0.289	0.306	2.712	3.064	11.282	1.6767	22,010.09	475.5156	4596.549	5938.6783
	$tSSO_3$	71.316	5.619	0.064	0.253	1.347	0.508	12.678	4.946	10.5688	1.3536	23,621.20	426.826	26,436.17	16,447.3743
	$tSSO_4$	88.894	6.061	0.116	0.362	0.537	0.387	5.102	3.839	10.8808	1.4016	22,401.50	480.8059	8531.313	8913.8709
	$tSSO_5$	99.98	0.165	0.668	0.824	0.061	0.037	0.514	0.382	21.7762	2.6418	19,982.09	420.3826	1891.765	452.4866
	$tSSO_6$	98.638	2.135	0.278	0.549	0.137	0.174	1.212	1.755	13.335	2.399	21,646.20	473.8394	2588.289	3570.9857
	$tSSO_7$	**99.996**	**0.088**	0.714	0.849	0.062	**0.035**	0.521	**0.369**	23.0932	2.6965	19,860.81	428.8591	**1879.873**	**57.8411**
	$tSSO_8$	99.89	0.546	**1.354**	1.173	**0.059**	0.04	**0.497**	0.401	**26.0226**	4.3313	**19,086.17**	614.0985	2056.674	772.1143
	MOPSO	12.042	2.554	0	**0**	2.113	0.454	17.758	4.29	12.0856	1.6538	24,347.86	307.4949	57,505.36	23,415.3032
	MOSSO	1.58	0.699	0	**0**	7.01	0.055	9.271	3.117	4.7402	0.4797	19,666.68	**33.48**	977,197.9	13,878.8233
	NSGA-II	21.096	3.25	0	**0**	2.25	0.777	19.435	7.076	0.0392	**0.0489**	24,289.14	713.0953	65,231.95	40,247.3188
150	$tSSO_0$	67.96	5.77	0.036	0.197	1.97	0.343	21.901	3.817	23.0904	3.509	36,497.8	446.2135	99,073.14	31,223.9014
	$tSSO_1$	76.334	6.347	0.078	0.276	1.552	0.349	17.458	4.014	22.8375	3.3043	35,809.46	542.9496	69,700.06	27,524.2792
	$tSSO_2$	102.624	7.73	0.22	0.473	0.929	0.312	10.699	3.723	23.8908	3.3867	33,666.71	657.8649	36,715.2	20,668.0023
	$tSSO_3$	79.2	5.726	0.098	0.304	1.465	0.327	16.518	3.814	23.6139	3.4082	35,679.86	536.1647	64,560.6	24,571.6138
	$tSSO_4$	98.296	6.9	0.166	0.413	0.932	0.276	10.655	3.313	24.0771	3.3979	33,969.23	669.9933	36,254.89	18,845.1508
	$tSSO_5$	149.92	0.427	1.362	1.177	0.042	**0.028**	0.441	0.361	43.0185	7.8536	29,176.09	593.4306	**2833.142**	**627.5616**
	$tSSO_6$	122.092	7.632	0.408	0.631	0.591	0.254	6.907	3.087	25.971	3.4265	32,432.3	657.7752	20,747.74	14,131.2079
	$tSSO_7$	**149.954**	**0.277**	1.47	1.151	0.04	**0.028**	0.414	0.356	51.0654	7.7838	28,979.39	598.0727	2870.726	638.1370
	$tSSO_8$	149.904	0.602	**3.548**	2.013	**0.036**	**0.028**	**0.39**	**0.347**	**74.6991**	13.5849	**27,451.57**	612.008	3233.779	995.0084
	MOPSO	13.692	2.496	0	**0**	1.744	0.289	18.011	3.32	26.7054	4.1966	36,486.46	388.5253	87,266.1	26,859.1298
	MOSSO	1.986	0.921	0	**0**	5.724	0.038	9.331	2.551	10.4472	1.2398	29,494.15	**43.4111**	1,466,864	18,034.0917
	NSGA-II	23.45	3.732	0	**0**	2.109	0.579	22.251	6.213	0.0855	**0.0743**	36,643.47	992.1199	120,913.5	60,590.6631

Table 4. Result for medium-size problem.

N_{sol}	Alg	Avg (N_n)	Std (N_n)	Avg (N_p)	Std (N_p)	Avg (GD)	Std (GD)	Avg (SP)	Std (SP)	Avg (T)	Std (T)	Avg (F_1)	Std (F_1)	Avg (F_2)	Std (F_2)
50	$tSSO_0$	24.472	4.177	0	**0**	20.001	2.216	109.497	7.657	4.5751	0.7871	32,537.48	456.8619	196,678.1	37,821.6436
	$tSSO_1$	25.954	4.105	0	**0**	18.074	2.375	103.29	9.431	4.403	0.7533	32,405.09	566.3836	167,400.6	37,900.9380
	$tSSO_2$	29.672	4.648	0	**0**	14.629	2.778	89.001	13.521	4.3079	0.7238	32,177.28	716.8466	120,611.9	39,137.8242
	$tSSO_3$	26.74	4.27	0	**0**	17.701	2.308	102.104	9.346	4.5363	0.765	32,413.09	581.7217	161,630.7	37,035.2625
	$tSSO_4$	29.87	4.493	0	**0**	14.258	2.518	87.075	12.468	4.397	0.7395	32,032.87	750.0289	118,041.1	36,646.8146
	$tSSO_5$	35.19	5.142	0.006	0.077	9.059	2.676	58.733	16.013	4.0844	0.6768	31,237.8	919.8058	65,527.05	32,220.9459
	$tSSO_6$	32.81	4.649	0	**0**	12.138	2.64	76.712	14.884	4.417	0.748	32,117.32	801.4451	90,607.97	32,073.7435
	$tSSO_7$	36.364	4.946	0.002	0.045	8.258	2.656	54.108	16.584	4.2589	0.7155	31,191.78	946.228	56,523.23	28,417.2914
	$tSSO_8$	**44.082**	4.463	**0.064**	0.303	**0.976**	1.184	6.054	8.46	4.0538	0.6846	28,331.11	1073.477	**5418.824**	8430.7514
	MOPSO	5.278	1.58	0	**0**	18.821	1.434	85.456	4.305	**6.7045**	1.1244	31,253.08	456.2674	286,050.1	35,533.4621
	MOSSO	1	**0**	0	**0**	20.393	**0.106**	**5.148**	**3.944**	2.6425	0.3924	**28,024.24**	**44.3314**	499,880.3	**1086.7815**
	NSGA-II	8.528	2.42	0	**0**	23.357	3.072	109.155	10.135	0.013	**0.022**	32,648.83	926.4653	272,009.9	62,082.4709
100	$tSSO_0$	27.056	4.507	0	**0**	16.977	0.907	113.305	3.756	17.8248	2.9748	65,051.87	667.284	558,643.9	50,252.7832
	$tSSO_1$	28.476	4.388	0	**0**	15.963	1.018	111.674	4.176	17.1294	2.739	64,873.31	824.3303	509,758.2	53,979.6751
	$tSSO_2$	31.5	5.173	0	**0**	14.406	1.139	107.424	5.392	16.6702	2.6083	64,582.33	1120.178	438,233.4	60,357.1999
	$tSSO_3$	30.116	4.689	0	**0**	15.331	0.948	110.571	4.615	17.569	2.8203	64,877.63	892.6177	478,286.6	49,968.1420
	$tSSO_4$	32.442	4.55	0	**0**	13.563	1.04	104.043	5.577	16.9932	2.6475	64,347.25	1159.757	402,642.2	55,190.6790
	$tSSO_5$	37.524	5.4	0	**0**	10.642	1.067	87.882	6.868	15.7426	2.3916	62,982.56	1581.812	302,505.7	58,114.0986
	$tSSO_6$	37.188	4.95	0.002	0.045	11.909	1.066	96.473	6.802	17.1704	2.702	64,295.33	1245.893	334,714.5	50,535.6466
	$tSSO_7$	40.262	5.529	0.002	0.045	9.49	1.126	80.679	8.005	16.5428	2.4717	62,883.59	1635.266	260,267.7	55,235.1007
	$tSSO_8$	**57.302**	6.849	**0.152**	0.435	**2.329**	0.992	21.916	9.291	15.8048	2.3262	**55,830.02**	2048.334	**57,761.36**	42,030.1210
	MOPSO	6.848	1.843	0	**0**	13.321	0.697	85.525	3.172	**25.8886**	3.993	62,458.84	631.2151	573,135.1	48,380.8306
	MOSSO	1	**0**	0	**0**	14.418	**0.054**	**5.869**	**2.966**	10.1184	1.161	56,046.99	**62.3825**	999,720.8	**1646.6140**
	NSGA-II	10.444	2.663	0	**0**	17.906	1.378	109.495	7.892	0.0476	**0.05**	65,316.82	1320.111	621,792.5	85,797.1254
150	$tSSO_0$	29.234	4.409	0	**0**	14.51	0.548	111.228	3.186	39.0909	5.8462	97,500.52	813.8773	924,285.2	56,397.7725
	$tSSO_1$	30.044	4.752	0.004	0.063	13.987	0.624	111.209	3.446	37.5183	5.4441	97,367.75	1050.166	873,603.5	62,586.7269
	$tSSO_2$	32.602	4.976	0	**0**	12.975	0.721	109.937	3.887	36.3396	5.1288	97,031.77	1314.433	781,236.9	72,066.7149
	$tSSO_3$	32.566	4.636	0	**0**	13.238	0.588	110.849	3.482	38.4948	5.5112	97,373.33	1186.611	801,509.8	59,333.8052
	$tSSO_4$	34.552	4.915	0	**0**	12.02	0.672	106.822	4.391	37.3443	5.448	96,646.12	1497.259	702,948.2	67,389.4702
	$tSSO_5$	38.842	5.225	0	**0**	9.865	0.657	94.662	5.112	34.557	4.8437	94,766.85	2117.23	561,339.4	76,923.5292
	$tSSO_6$	39.38	4.876	0.002	0.045	10.63	0.684	100.761	5.231	37.7004	5.5096	96,480.24	1760.161	593,359.7	66,298.1134
	$tSSO_7$	43.326	5.156	0.002	0.045	8.652	0.66	86.662	5.569	36.3051	5.1832	94,508.79	2363.881	476,971	77,652.8961
	$tSSO_8$	**62.466**	7.329	**0.336**	0.713	**2.584**	0.702	29.27	7.644	34.4403	4.8116	**83,208.2**	2666.568	**142,370.8**	69,677.5283
	MOPSO	7.816	1.896	0	**0**	10.879	0.497	85.598	2.492	**56.5194**	8.1362	93,759.12	788.4589	857,348.1	63,773.8475
	MOSSO	1	**0**	0	**0**	11.777	**0.036**	**5.749**	**2.406**	22.2228	2.7068	84,077.84	**77.1704**	1,499,801	**1397.3776**
	NSGA-II	11.27	2.609	0	**0**	15.074	0.927	107.625	7.159	0.1059	**0.0684**	97,772.24	1688.187	994,780.5	101,431.9720

Table 5. Result for large-size problem.

N_{sol}	Alg	Avg (N_n)	Std (N_n)	Avg (N_p)	Std (N_p)	Avg (GD)	Std (GD)	Avg (SP)	Std (SP)	Avg (T)	Std (T)	Avg (F_1)	Std (F_1)	Avg (F_2)	Std (F_2)
50	$tSSO_0$	27.174	4.155	0	**0**	748.62	96.656	4480.39	365.757	8.4167	1.5329	98,628.83	771.7182	143,665.5	35,722.9236
	$tSSO_1$	29.798	4.686	0	**0**	639.303	109.316	4019.28	526.113	8.0533	1.343	98,279.76	960.7157	106,329.4	34,186.9693
	$tSSO_2$	34.79	5.074	0	**0**	478.211	136.159	3161.886	809.014	7.7758	1.2132	97,705.9	1259.995	62,884.07	30,649.8855
	$tSSO_3$	30.616	4.385	0	**0**	633.698	108.007	3994.477	524.318	8.218	1.3142	98,354.38	893.9128	104,469.4	33,485.2153
	$tSSO_4$	34.058	4.867	0	**0**	484.791	118.408	3210.835	695.26	7.9161	1.2229	97,382.84	1207.66	63,353.57	27,636.0545
	$tSSO_5$	42.104	5.073	0	**0**	237.875	158.78	1641.271	1075.713	7.293	1.092	95,492.67	1530.155	21,440.25	19,011.2797
	$tSSO_6$	37.198	4.822	0	**0**	408.566	128.757	2754.903	808.218	7.904	1.2089	97,480.85	1267.524	46,923.7	24,637.2246
	$tSSO_7$	42.934	4.724	0.002	0.045	205.495	154.737	1424.205	1060.277	7.6309	1.108	95,610.53	1573.731	17,520.32	16,311.2192
	$tSSO_8$	**45.918**	2.998	**0.006**	0.1	**6.73**	**36.201**	**45.981**	255.548	6.8534	1.0066	**89,370.73**	2219.432	**1315.029**	**2029.9054**
	MOPSO	6.328	1.726	0	**0**	904.797	76.097	4909.066	**144.96**	**12.4131**	1.93	97,663.75	709.6203	207,628.7	34,209.3182
	MOSSO	4.948	**1.556**	0	**0**	1152.116	58.678	4687.853	261.903	4.7327	0.6193	98,649.82	**634.5005**	334,246.6	33,687.3944
	NSGA-II	10.368	2.799	0	**0**	856.652	145.453	4711.009	392.009	0.0201	**0.0245**	98,891.73	1471.68	190,118.4	61,317.1989
100	$tSSO_0$	30.448	4.494	0	**0**	657.704	35.537	4947.358	**69.806**	33.2646	5.1487	197,248.5	1061.073	436,547.4	46,564.5095
	$tSSO_1$	32.06	4.894	0	**0**	613.194	42.945	4832.571	135.141	31.8998	4.8814	196,881.6	1430.933	380,499.4	52,806.8613
	$tSSO_2$	36.274	5.131	0	**0**	539.157	47.963	4531.276	241.08	30.8796	4.6327	196,021.5	1827.626	295,533.3	51,535.3657
	$tSSO_3$	34.158	4.699	0	**0**	586.502	42.738	4742.246	167.804	32.6676	4.963	196,844.5	1456.553	348,421.8	49,963.8885
	$tSSO_4$	37.156	5.092	0	**0**	511.663	48.763	4388.164	277.232	31.3592	4.5905	195,426.1	2058.278	266,699.6	49,262.8425
	$tSSO_5$	44.196	5.801	0	**0**	400.691	56.641	3663.083	419.515	28.8562	4.0773	192,100.9	2664.295	166,154.7	45,483.4355
	$tSSO_6$	41.918	5.683	0	**0**	453.726	49.978	4037.566	329.065	31.453	4.5322	195,306.7	2181.89	210,798.2	45,526.3109
	$tSSO_7$	47.33	5.836	0	**0**	363.759	57.152	3381.729	449.962	30.127	4.1393	191,714.3	2919.089	137,930.9	41,874.8504
	$tSSO_8$	**72.528**	7.324	**0.116**	0.468	**38.469**	58.147	**382.143**	578.595	27.616	3.7534	**176,917.2**	4417.363	**6889.327**	**8665.4426**
	MOPSO	7.948	2.065	0	**0**	641.551	38.106	4910.514	90.919	**49.2814**	7.0987	195,334.2	995.1512	416,075.8	48,864.6986
	MOSSO	6.508	**1.72**	0	**0**	813.33	**28.137**	4705.793	166.873	19.1044	2.2699	197,323	**843.5352**	665,380.8	45,753.2918
	NSGA-II	12.482	2.856	0	**0**	686.503	64.443	4926.057	119.871	0.0794	**0.041**	197,624.3	2143.287	478,296.4	87,822.5684
150	$tSSO_0$	31.892	4.499	0	**0**	571.334	23.553	4986.075	**23.806**	74.9205	11.0373	295,934.7	1364.945	739,913.7	60,689.2869
	$tSSO_1$	33.948	4.799	0	**0**	541.511	25.246	4952.636	54.778	71.7156	10.4578	295,532.6	1663.517	665,347.3	61,362.5540
	$tSSO_2$	37.058	5.306	0	**0**	497.394	31.575	4816.909	128.206	69.2553	9.7601	294,401.4	2369.671	562,929.2	70,555.4639
	$tSSO_3$	36.176	5.035	0	**0**	515.033	26.975	4884.23	88.475	73.4922	10.7752	295,225.5	1901.874	602,559.3	62,546.5565
	$tSSO_4$	38.654	5.274	0	**0**	461.168	33.508	4648.141	179.439	70.7016	10.1622	293,575.6	2662.045	485,020.5	69,694.8734
	$tSSO_5$	46.074	6.107	0.002	0.045	375.99	38.045	4077.306	301.451	65.0655	9.094	289,048.3	3537.873	325,141.9	64,553.0441
	$tSSO_6$	44.548	5.498	0	**0**	407.532	32.447	4317.863	231.212	70.9386	10.2171	292,870.2	2835.374	379,900.7	59,375.2787
	$tSSO_7$	49.592	5.643	0.004	0.063	339.382	35.921	3774.411	317.912	68.0226	9.4273	288,517.2	3575.387	265,771.2	54,642.6111
	$tSSO_8$	**77.03**	7.705	**0.362**	1.141	**75.415**	53.721	**915.721**	647.916	62.04	8.258	**263,672.3**	6382.976	**22,508.07**	**20,508.0012**
	MOPSO	8.758	2.175	0	**0**	522.934	26.468	4906.145	78.318	**110.3736**	14.8236	293,040.9	1222.526	621,413.8	62,184.5910
	MOSSO	7.398	**2.044**	0	**0**	666.568	**18.525**	4689.758	137.576	43.2861	4.9709	295,948.8	**1105.613**	1,005,113	55,590.2798
	NSGA-II	13.36	3.007	0	**0**	590.492	41.685	4939.701	88.364	0.18	**0.0623**	296,189.8	2758.174	792,818.2	110,125.8133

In addition, the histogram graphs for the average and standard deviations of the results of two-objective values, i.e., the energy consumption values and the makspan values, for three different-sized benchmark problems by the proposed tSSO and the compared MOPSO, MOSSO, and NSGA-II algorithms are further drawn, as shown in the following Figures 1–12, in order to obtain more results and discussion to validate the performance of the proposed model and to have a comparison at a glance. The best results among $tSSO_0$ to $tSSO_8$ are taken as the solution of tSSO and plotted in Figures 1–12.

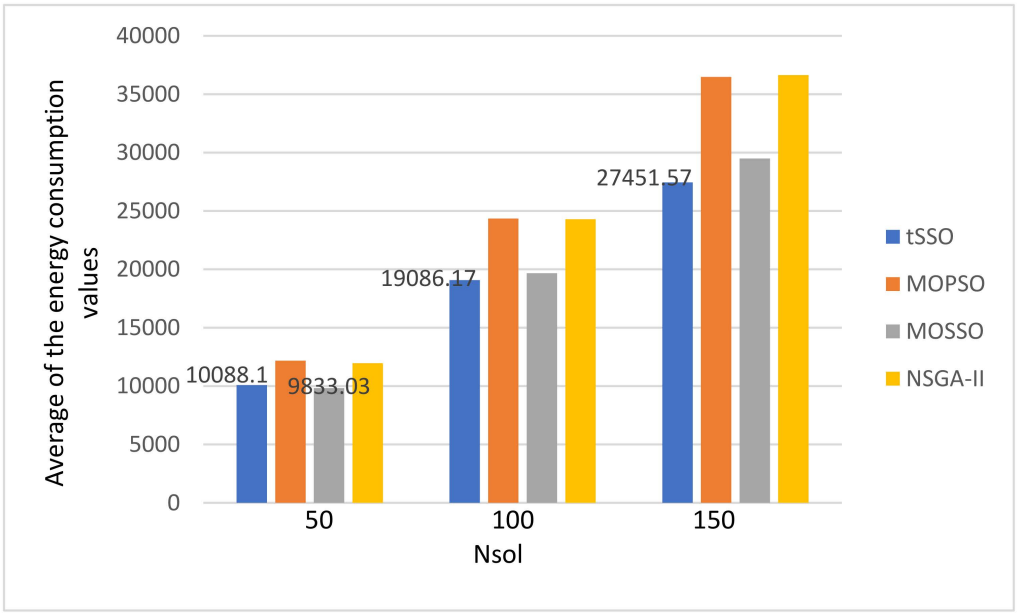

Figure 1. Average of the energy consumption for small-size benchmark.

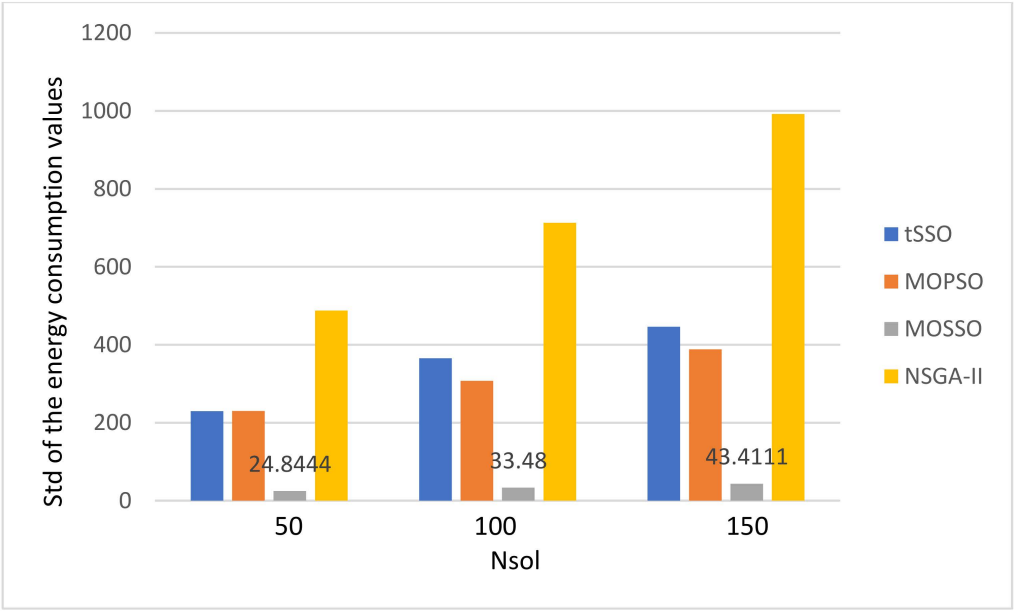

Figure 2. Std of the energy consumption for small-size benchmark.

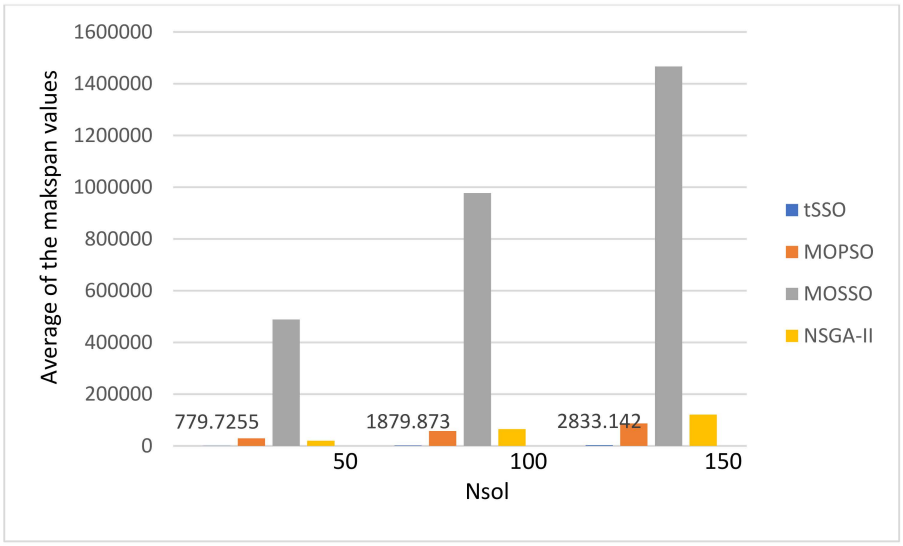

Figure 3. Average of the makspan for small-size benchmark.

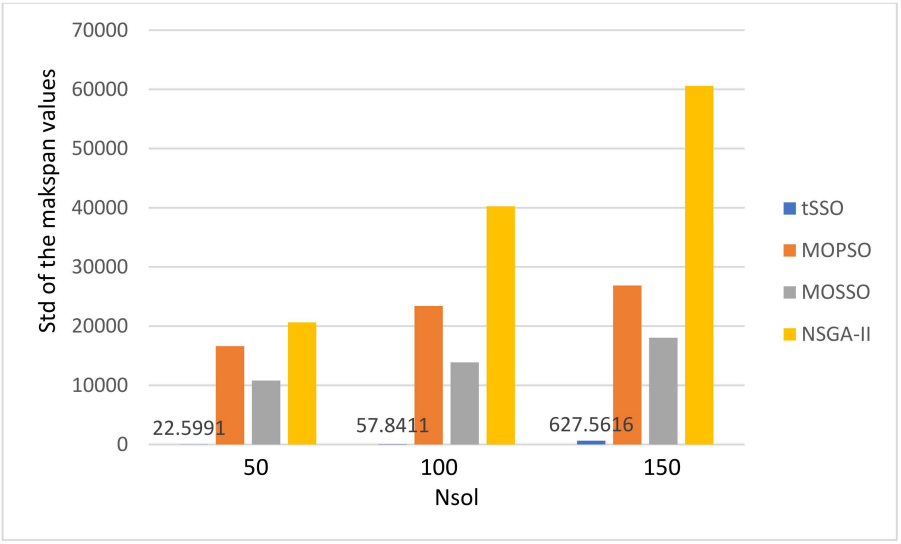

Figure 4. Std of the makspan for small-size benchmark.

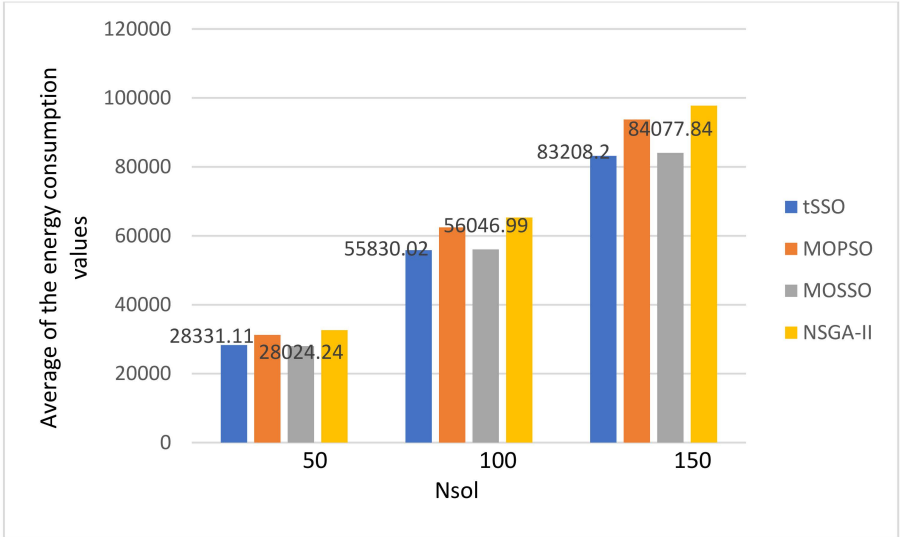

Figure 5. Average of the energy consumption for medium-size benchmark.

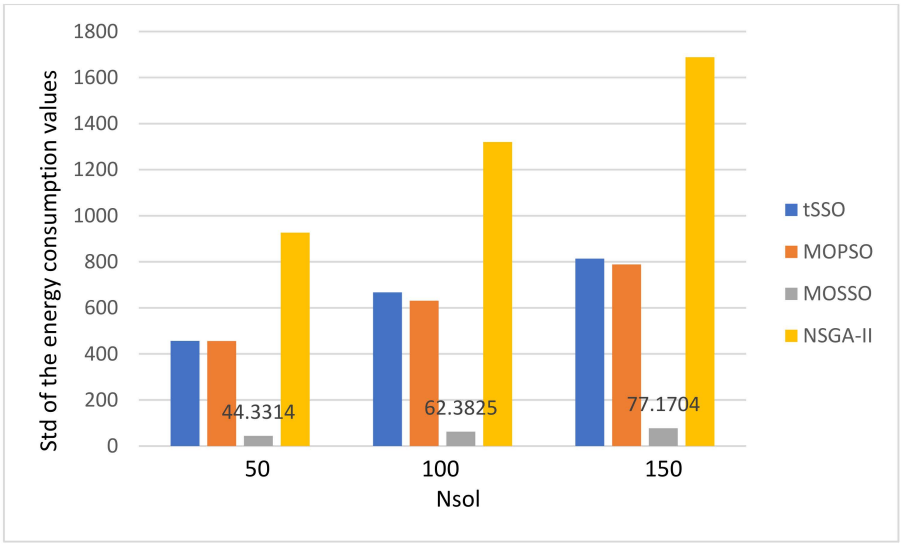

Figure 6. Std of the energy consumption for medium-size benchmark.

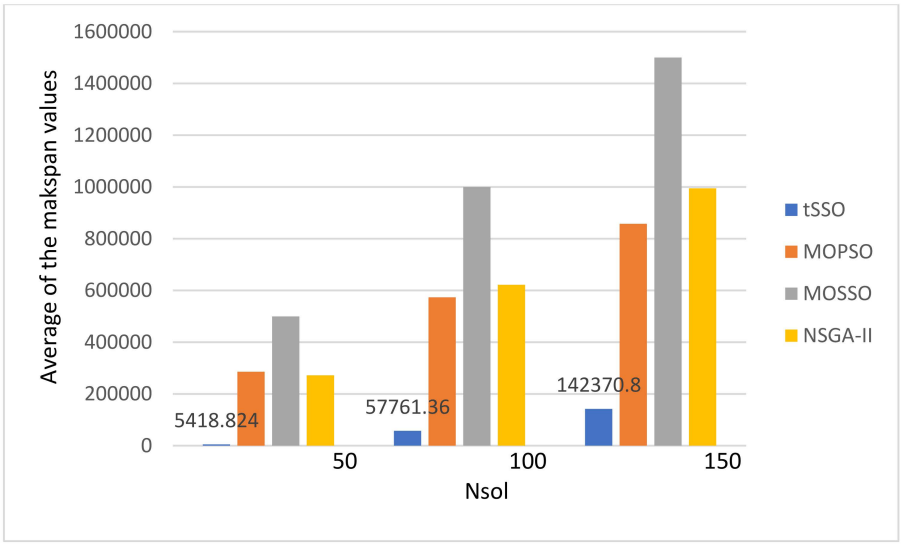

Figure 7. Average of the makspan for medium-size benchmark.

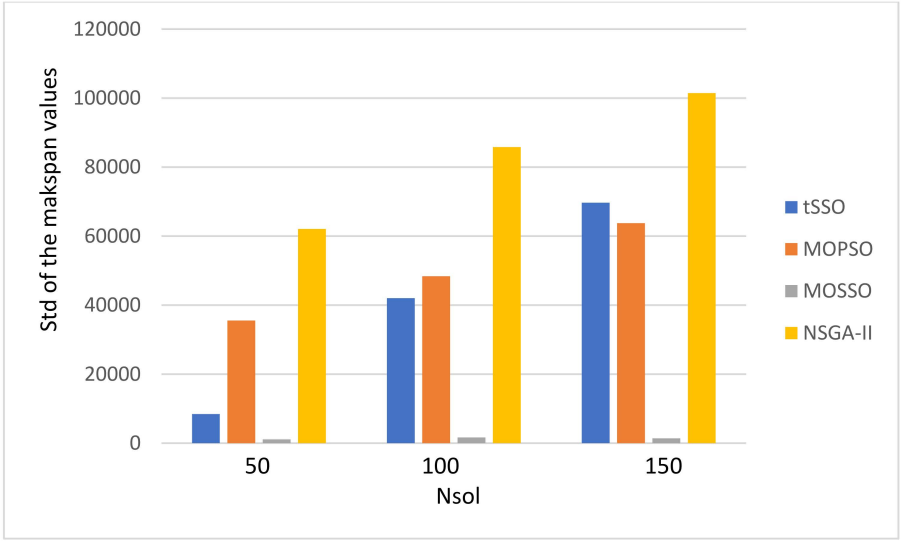

Figure 8. Std of the makspan for medium-size benchmark.

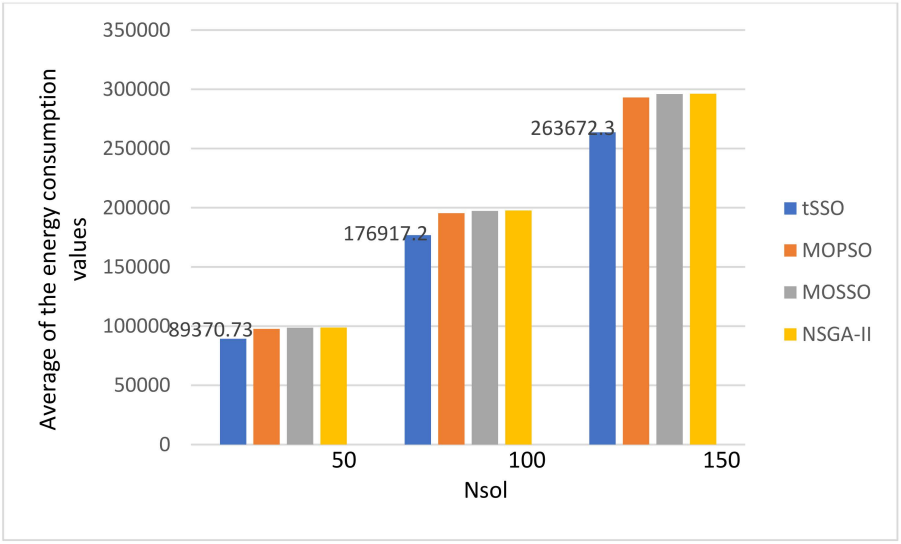

Figure 9. Average of the energy consumption for large-size benchmark.

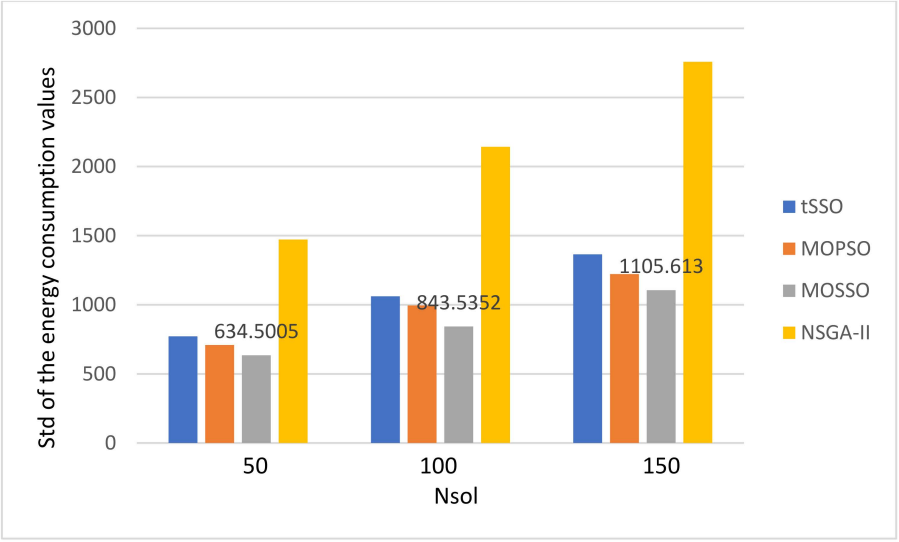

Figure 10. Std of the energy consumption for large-size benchmark.

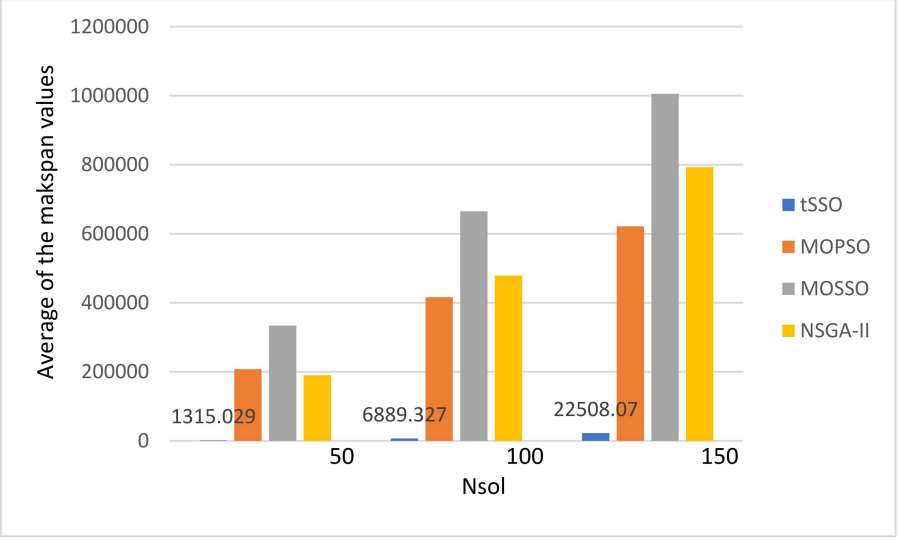

Figure 11. Average of the makspan for large-size benchmark.

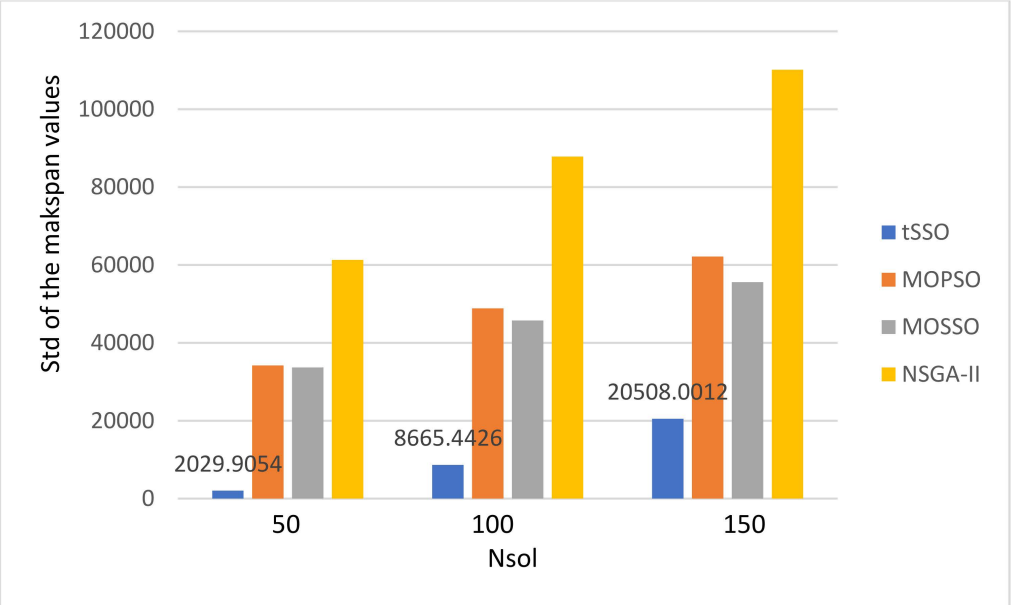

Figure 12. Std of the makspan for large-size benchmark.

On average, the average of the energy consumption for the small-size benchmark obtained by the proposed tSSO is better than those obtained by the compared algorithms, including MOPSO, MOSSO, and NSGA-II, though it is slightly worse than the performance of MOSSO when Nsol equals 50, as shown in Figure 1.

The standard deviation of the energy consumption for the small-size benchmark obtained by the MOSSO is the best, as shown in Figure 2.

The average of the makspan for the small-size benchmark obtained by the proposed tSSO is the best and is superior to MOSSO, as shown in Figure 3.

The standard deviation of the makspan for the small-size benchmark obtained by the proposed tSSO is the best and is superior to the compared algorithms, including MOPSO, MOSSO, and NSGA-II, as shown in Figure 4.

On average, the average of the energy consumption for the medium-size benchmark obtained by the proposed tSSO is better than those obtained by the compared algorithms, including MOPSO, MOSSO, and NSGA-II, though it is slightly worse than the performance of MOSSO when Nsol equals 50, as shown in Figure 5.

The standard deviation of the energy consumption for the medium-size benchmark obtained by the MOSSO is the best, as shown in Figure 6.

The average of the makspan for the medium-size benchmark obtained by the proposed tSSO is the best and is superior to the compared algorithms, including MOPSO, MOSSO, and NSGA-II, as shown in Figure 7.

The standard deviation of the makspan for the medium-size benchmark obtained by the MOSSO is the best and is superior to the compared algorithms, including the proposed tSSO, MOPSO, and NSGA-II, as shown in Figure 8.

The average of the energy consumption for the large-size benchmark obtained by the proposed tSSO is better than those obtained by the compared algorithms, including MOPSO, MOSSO, and NSGA-II, as shown in Figure 9.

The standard deviation of the energy consumption for the large-size benchmark obtained by the MOSSO is the best, as shown in Figure 10.

The average of the makspan for the large-size benchmark obtained by the proposed tSSO is the best and is superior to the compared algorithms, including MOPSO, MOSSO, and NSGA-II, as shown in Figure 11.

The standard deviation of the makspan for the large-size benchmark obtained by the proposed tSSO is the best and is superior to the compared algorithms, including MOPSO, MOSSO, and NSGA-II, as shown in Figure 12.

Overall, the performance of the proposed tSSO outperforms the compared algorithms, including MOPSO, MOSSO, and NSGA-II in all aspects of measures.

7. Conclusions

This study sheds light on a nascent two-objective time-constrained job scheduling problem focusing on energy consumption and service quality in terms of the makspan to find non-dominated solutions for the purpose of ameliorating the service quality and addressing the environmental issues of cloud computing services. In response to this two-objective problem, we proposed a new two-objective simplified swarm optimization (tSSO) algorithm to revise and improve the errors in the previous MOSSO algorithm [17], which, in the multi-objective problem, ignores the fact that the number of temporary nondominated solutions is not always only one, and some temporary nondominated solutions may not be temporary nondominated solutions in the next generation, based on SSO to deliver the job scheduling in cloud computing.

To ensure better solution quality, the tSSO algorithm integrates the crowding distance, a hybrid elite selection, and a new stepwise update mechanism, i.e., the proposed tSSO is a population-based, all-variable update, and stepwise function-based method. From the experiments conducted on three different-sized problems [17], regardless of the parameter setting, each of the proposed tSSOs outperformed the MOPSO [13], MOSSO [17], and NSGA-II [19,20], in convergency, diversity, the number of obtained temporary nondominated solutions, and the number of obtained real nondominated solutions. Among nine different parameter settings, we concluded that the tSSO algorithm with $c_p = c_w = 0.5$ is the best one. The results prove that the proposed tSSO can successfully achieve the aim of this work.

In order for the readers to understand the real-life cloud problems, the assumptions in Section 3.2 are simplified. Hence, future work will relax the assumptions to further meet and solve real-life cloud problems. In addition, a sequel work that considers real-life assumptions with changes in the algorithms is needed in the near future.

Moreover, in future works, the proposed model will be tested with more data from different domains, and the results will be compared with recently published studies from top journals and conferences.

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

Funding: This research was supported in part by the National Natural Science Foundation of China (Grant No. 621060482), Research and Development Projects in Key Areas of Guangdong Province (Grant No. 2021B0101410002), and the National Science and Technology Council, R.O.C. (MOST 107-2221-E-007-072-MY3, MOST 110-2221-E-007-107-MY3, MOST 109-2221-E-424-002 and MOST 110-2511-H-130-002).

Acknowledgments: We wish to thank the anonymous editor and the referees for their constructive comments and recommendations, which significantly improved this paper. This article was once submitted to arXiv as a temporary submission that was only for reference and did not provide the copyright.

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

Notations

The following notations are used:

$\lvert \bullet \rvert$	number of elements in $\bullet$
N_{var}	number of jobs used in the test problem
N_{cpu}	number of processors contained in the given data center
N_{run}	number of runs for the algorithms
N_{gen}	number of generations in each run
N_{sol}	number of solutions in each generation
N_{non}	number of selected temporary nondominated solutions
X_i	ith solution
$x_{i,j}$	jth variable in X_i
P_i	the best solution among all solutions updated based on X_i in SSO
$p_{i,j}$	jth variable in P_i
$gBest$	index of the best solution among all solutions in SSO, i.e., $F(P_{gBest})$ is better than or equal to $F(P_i)$ for $i = 1, 2, \ldots, N_{sol}$
ρ_I	random number generated uniformly within interval I
c_g, c_p, c_w, c_r	positive parameters used in SSO with $c_g + c_p + c_w + c_r = 1$
C_g, C_p, C_w	$C_g = c_g$, $C_p = C_g + c_p$, and $C_w = C_p + c_w$
$F_l(\bullet)$	lth fitness function value of solution $\bullet$
$Max(\bullet)$	maximal value of $\bullet$, i.e., $Max(F_l)$ is the maximal value of the lth objective function
$Min(\bullet)$	minimal value of $\bullet$, i.e., $Min(F_l)$ is the minimal value of the lth objective function
S_t	set of selected solutions from Π_t to generate new solutions in the $(t+1)^{th}$ generation. Note that $S_1 = \Pi_1$ and $\lvert S_t \rvert = N_{sol}$ for $i = 1, 2, \ldots, N_{gen}$
$size_i$	size of the job i for $i = 1, 2, \ldots, N_{var}$
$start_i$	start time of the job i for $i = 1, 2, \ldots, N_{var}$
$speed_j$	execution speed of the processor j for $j = 1, 2, \ldots, N_{cpu}$
e_j	energy consumption per unit time of the processor j for $j = 1, 2, \ldots, N_{cpu}$
T_{ub}	deadline constraint of job scheduling
$t_{i,j}$	processing time $t_{i,j}$ where $t_{i,j} = \begin{cases} size_i/speed_j & \text{if task } i \text{ is processed on processor } j \\ 0 & \text{otherwise} \end{cases}$ for $i = 1, 2, \ldots, N_{var}$ and $j = 1, 2, \ldots, N_{cpu}$

References

1. Wang, F.; Xu, J.; Cui, S. Optimal Energy Allocation and Task Offloading Policy for Wireless Powered Mobile Edge Computing Systems. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 2443–2459. [CrossRef]
2. Wei, S.C.; Yeh, W.C. Resource allocation decision model for dependable and cost-effective grid applications based on Grid Bank. *Future Gener. Comput. Syst.* **2017**, *77*, 12–28. [CrossRef]
3. Yeh, W.C.; Wei, S.C. Economic-based resource allocation for reliable Grid-computing service based on Grid Bank. *Future Gener. Comput. Syst.* **2012**, *28*, 989–1002. [CrossRef]
4. Manikandan, N.; Gobalakrishnan, N.; Pradeep, K. Bee optimization based random double adaptive whale optimization model for task scheduling in cloud computing environment. *Comput. Commun.* **2022**, *187*, 35–44. [CrossRef]
5. Guo, W.; Li, J.; Chen, G.; Niu, Y.; Chen, C. A PSO-Optimized Real-Time Fault-Tolerant Task Allocation Algorithm in Wireless Sensor Networks. *IEEE Trans. Parallel Distrib. Syst.* **2015**, *26*, 3236–3249. [CrossRef]
6. Afifi, H.; Horbach, K.; Karl, H. A Genetic Algorithm Framework for Solving Wireless Virtual Network Embedding. In Proceedings of the 2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), Barcelona, Spain, 21–23 October 2019.
7. Santos, J.; Hempel, M.; Sharif, H. Compression Distortion-Rate Analysis of Biomedical Signals in Machine Learning Tasks in Biomedical Wireless Sensor Network Applications. In Proceedings of the 2020 International Wireless Communications and Mobile Computing (IWCMC), Limassol, Cyprus, 15–19 June 2020.
8. Sun, Z.; Liu, Y.; Tao, L. Attack Localization Task Allocation in Wireless Sensor Networks Based on Multi-Objective Binary Particle Swarm Optimization. *J. Netw. Comput. Appl.* **2018**, *112*, 29–40. [CrossRef]
9. Lu, Y.; Zhou, J.; Xu, M. Wireless Sensor Networks for Task Allocation using Clone Chaotic Artificial Bee Colony Algorithm. In Proceedings of the 2019 IEEE International Conference of Intelligent Applied Systems on Engineering (ICIASE), Fuzhou, China, 26–29 April 2019.
10. Khan, M.S.A.; Santhosh, R. Task scheduling in cloud computing using hybrid optimization algorithm. *Soft Comput.* **2022**, *26*, 3069–13079. [CrossRef]

11. Malawski, M.; Juve, G.; Deelman, E.; Nabrzyski, J. Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. *Future Gener. Comput. Syst.* **2015**, *48*, 1–18. [CrossRef]
12. Chen, H.; Zhu, X.; Guo, H.; Zhu, J.; Qin, X.; Wu, J. Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment. *J. Syst. Softw.* **2015**, *99*, 20–35. [CrossRef]
13. Coello, C.A.C.; Pulido, G.T.; Lechuga, M.S. Handling multiple objectives with particle swarm optimization. *IEEE Trans. Evol. Comput.* **2004**, *8*, 256–279. [CrossRef]
14. Guo, X. Multi-objective task scheduling optimization in cloud computing based on fuzzy self-defense algorithm. *Alex. Eng. J.* **2021**, *60*, 5603–5609. [CrossRef]
15. Liu, J.X.; Luo, G.; Zhang, X.M.; Zhang, F.; Li, B.N. Job scheduling model for cloud computing based on multi-objective genetic algorithm. *Int. J. Comput. Sci. Issues* **2013**, *10*, 134–139.
16. Jena, R.K. Multi objective task scheduling in cloud environment using nested PSO framework. *Procedia Comput. Sci.* **2015**, *57*, 1219–1227. [CrossRef]
17. Huang, C.L.; Jiang, Y.Z.; Yin, Y.; Yeh, W.C.; Chung, V.Y.Y.; Lai, C.M. Multi Objective Scheduling in Cloud Computing Using MOSSO. In Proceedings of the 2018 IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, Brazil, 8–13 July 2018. [CrossRef]
18. Yeh, W.C. A two-stage discrete particle swarm optimization for the problem of multiple multi-level redundancy allocation in series systems. *Expert Syst. Appl.* **2009**, *36*, 9192–9200. [CrossRef]
19. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197. [CrossRef]
20. Yeh, W.C.; Zhu, W.; Yin, Y.; Huang, C.L. Cloud computing task scheduling problem by Nondominated Sorting Genetic Algorithm II (NSGA-II). In Proceedings of the First Australian Conference on Industrial Engineering and Operations Management, Sydney, Australia, 20–22 December 2022.
21. Houssein, E.H.; Gad, A.G.; Wazery, Y.M.; Suganthan, P.N. Task Scheduling in Cloud Computing based on Meta-heuristics: Review, Taxonomy, Open Challenges, and Future Trends. *Swarm Evol. Comput.* **2021**, *62*, 100841. [CrossRef]
22. Arunarani, A.R.; Manjul, D.; Sugumaran, V. Task scheduling techniques in cloud computing: A literature survey. *Future Gener. Comput. Syst.* **2019**, *91*, 407–415. [CrossRef]
23. Kumar, M.; Sharma, S.C.; Goel, A.; Singh, S.P. A comprehensive survey for scheduling techniques in cloud computing. *J. Netw. Comput. Appl.* **2019**, *143*, 1–33. [CrossRef]
24. Chen, X.; Cheng, L.; Liu, C.; Liu, Q.; Liu, J.; Mao, Y.; Murphy, J. A WOA-Based Optimization Approach for Task Scheduling in Cloud Computing Systems. *IEEE Syst. J.* **2020**, *14*, 3117–3128. [CrossRef]
25. Attiya, I.; Elaziz, M.A.; Xiong, S. Job Scheduling in Cloud Computing Using a Modified Harris Hawks Optimization and Simulated Annealing Algorithm. *Comput. Intell. Neurosci.* **2020**, *2020*, 3504642. [CrossRef]
26. Gąsior, J.; Seredyński, F. Security-Aware Distributed Job Scheduling in Cloud Computing Systems: A Game-Theoretic Cellular Automata-Based Approach. In Proceedings of the International Conference on Computational Science ICCS 2019: Computational Science—ICCS, 2019; pp. 449–462.
27. Mansouri, N.; Javidi, M.M. Cost-based job scheduling strategy in cloud computing environments. *Distrib. Parallel Databases* **2020**, *38*, 365–400. [CrossRef]
28. Cheng, F.; Huang, Y.; Tanpure, B.; Sawalani, P.; Cheng, L.; Liu, C. Cost-aware job scheduling for cloud instances using deep reinforcement learning. *Clust. Comput.* **2022**, *25*, 619–631. [CrossRef]
29. Shukri, S.E.; Al-Sayyed, R.; Hudaib, A.; Mirjalili, S. Enhanced multi-verse optimizer for task scheduling in cloud computing environments. *Expert Syst. Appl.* **2021**, *168*, 114230. [CrossRef]
30. Jacob, T.P.; Pradeep, K. A Multi-objective Optimal Task Scheduling in Cloud Environment Using Cuckoo Particle Swarm Optimization. *Wirel. Pers. Commun.* **2019**, *109*, 315–331. [CrossRef]
31. Abualigah, L.; Diabat, A. A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments. *Clust. Comput.* **2021**, *24*, 205–223. [CrossRef]
32. Sanaj, M.S.; Prathap, P.M.J. Nature inspired chaotic squirrel search algorithm (CSSA) for multi objective task scheduling in an IAAS cloud computing atmosphere. *Eng. Sci. Technol. Int. J.* **2020**, *23*, 891–902. [CrossRef]
33. Abualigah, L.; Alkhrabsheh, M. Amended hybrid multi-verse optimizer with genetic algorithm for solving task scheduling problem in cloud computing. *J. Supercomput.* **2022**, *78*, 740–765. [CrossRef]
34. Mitchell, M. *An Introduction to Genetic Algorithms*; MIT Press: London, UK, 1998.
35. Yeh, W.C. Orthogonal simplified swarm optimization for the series–parallel redundancy allocation problem with a mix of components. *Knowl.-Based Syst.* **2014**, *64*, 1–12. [CrossRef]
36. Yeh, W.C. A New Exact Solution Algorithm for a Novel Generalized Redundancy Allocation Problem. *Inf. Sci.* **2017**, *408*, 182–197. [CrossRef]
37. Yeh, W.C.; Hsieh, Y.H.; Hsu, K.Y.; Huang, C.L. ANN and SSO Algorithms for a Newly Developed Flexible Grid Trading Model. *Electronics* **2022**, *11*, 11193259. [CrossRef]
38. Yeh, W.C. Simplified swarm optimization in disassembly sequencing problems with learning effects. *Comput. Oper. Res.* **2012**, *39*, 2168–2177. [CrossRef]

39. Yeh, W.C. New parameter-free simplified swarm optimization for artificial neural network training and its application in the prediction of time series. *IEEE Trans. Neural Netw. Learn. Syst.* **2013**, *24*, 661–665. [PubMed]
40. Yeh, W.C.; Zhu, W.; Peng, Y.F.; Huang, C.L. A Hybrid Algorithm Based on Simplified Swarm Optimization for Multi-Objective Optimizing on Combined Cooling, Heating and Power System. *Appl. Sci.* **2022**, *12*, 10595. [CrossRef]
41. Yeh, W.C.; Huang, C.L.; Lin, P.; Chen, Z.; Jiang, Y.; Sun, B. Simplex Simplified Swarm Optimization for the Efficient Optimization of Parameter Identification for Solar Cell Models. *IET Renew. Power Gener.* **2018**, *12*, 45–51. [CrossRef]
42. Yeh, W.C.; Ke, Y.C.; Chang, P.C.; Yeh, Y.M.; Chung, V. Forecasting Wind Power in the Mai Liao Wind Farm based on the Multi-Layer Perceptron Artificial Neural Network Model with Improved Simplified Swarm Optimization. *Int. J. Electr. Power Energy Syst.* **2014**, *55*, 741–748. [CrossRef]
43. Yeh, W.C.; Liu, Z.; Yang, Y.C.; Tan, S.Y. Solving Dual-Channel Supply Chain Pricing Strategy Problem with Multi-Level Programming Based on Improved Simplified Swarm Optimization. *Technologies* **2022**, *2022*, 10030073. [CrossRef]
44. Lin, H.C.S.; Huang, C.L.; Yeh, W.C. A Novel Constraints Model of Credibility-Fuzzy for Reliability Redundancy Allocation Problem by Simplified Swarm Optimization. *Appl. Sci.* **2021**, *11*, 10765. [CrossRef]
45. Tan, S.Y.; Yeh, W.C. The Vehicle Routing Problem: State-of-the-Art Classification and Review. *Appl. Sci.* **2021**, *11*, 10295. [CrossRef]
46. Zhu, W.; Huang, C.L.; Yeh, W.C.; Jiang, Y.; Tan, S.Y. A Novel Bi-Tuning SSO Algorithm for Optimizing the Budget-Limited Sensing Coverage Problem in Wireless Sensor Networks. *Appl. Sci.* **2021**, *11*, 10197. [CrossRef]
47. Yeh, W.C.; Jiang, Y.; Tan, S.Y.; Yeh, C.Y. A New Support Vector Machine Based on Convolution Product. *Complexity* **2021**, *2021*, 9932292. [CrossRef]
48. Wu, T.Y.; Jiang, Y.Z.; Su, Y.Z.; Yeh, W.C. Using Simplified Swarm Optimization on Multiloop Fuzzy PID Controller Tuning Design for Flow and Temperature Control System. *Appl. Sci.* **2020**, *10*, 8472. [CrossRef]
49. Yeh, W.C.; Jiang, Y.; Huang, C.L.; Xiong, N.N.; Hu, C.F.; Yeh, Y.H. Improve Energy Consumption and Signal Transmission Quality of Routings in Wireless Sensor Networks. *IEEE Access* **2020**, *8*, 198254–198264. [CrossRef]
50. Yeh, W.C. A new harmonic continuous simplified swarm optimization. *Appl. Soft Comput.* **2019**, *85*, 105544. [CrossRef]
51. Yeh, W.C.; Lai, C.M.; Tseng, K.C. Fog computing task scheduling optimization based on multi-objective simplified swarm optimization. *J. Phys. Conf. Ser.* **2019**, *1411*, 012007. [CrossRef]
52. Yeh, W.C. Solving cold-standby reliability redundancy allocation problems using a new swarm intelligence algorithm. *Appl. Soft Comput.* **2019**, *83*, 105582. [CrossRef]
53. Veldhuizen, V.D.A.; Lamont, G.B. Multiobjective evolutionary algorithm research: A history and analysis. *Evol. Comput.* **1999**, *8*, 125–147. [CrossRef]
54. Schott, J.R. Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization. 1995. Available online: http://hdl.handle.net/1721.1/11582 (accessed on 11 January 2022).