mathematics

Article

PGA: A New Hybrid PSO and GA Method for Task Scheduling with Deadline Constraints in Distributed Computing

Kaili Shao [1], Ying Song [2,3] and Bo Wang [4,*]

[1] Faculty of Engineering, Huanghe Science and Technology University, Zhengzhou 450003, China; sklemail@hhstu.edu.cn
[2] Beijing Advanced Innovation Center for Materials Genome Engineering, Beijing Information Science and Technology University, Beijing 100083, China; songying@bistu.edu.cn
[3] Beijing Key Laboratory of Internet Culture and Digital Dissemination, Beijing Information Science and Technology University, Beijing 100029, China
[4] Software Engineering College, Zhengzhou University of Light Industry, Zhengzhou 450001, China
* Correspondence: wangb@zzuli.edu.cn

Abstract: Distributed computing, e.g., cluster and cloud computing, has been applied in almost all areas for data processing, while high resource efficiency and user satisfaction are still the ambition of distributed computing. Task scheduling is indispensable for achieving the goal. As the task scheduling problem is NP-hard, heuristics and meta-heuristics are frequently applied. Every method has its own advantages and limitations. Thus, in this paper, we designed a hybrid heuristic task scheduling problem by exploiting the high global search ability of the Genetic Algorithm (GA) and the fast convergence of Particle Swarm Optimization (PSO). Different from existing hybrid heuristic approaches that simply sequentially perform two or more algorithms, the PGA applies the evolutionary method of a GA and integrates self- and social cognitions into the evolution. We conduct extensive simulated environments for the performance evaluation, where simulation parameters are set referring to some recent related works. Experimental results show that the PGA has 27.9–65.4% and 33.8–69.6% better performance than several recent works, on average, in user satisfaction and resource efficiency, respectively.

Keywords: distributed computing; genetic algorithm; swarm intelligence; task scheduling

MSC: 68M20

check for updates

Citation: Shao, K.; Song, Y.; Wang, B. PGA: A New Hybrid PSO and GA Method for Task Scheduling with Deadline Constraints in Distributed Computing. *Mathematics* **2023**, *11*, 1548. https://doi.org/10.3390/math11061548

Academic Editors: Pavel Lyakhov and Maxim Deryabin

Received: 15 February 2023
Revised: 12 March 2023
Accepted: 21 March 2023
Published: 22 March 2023

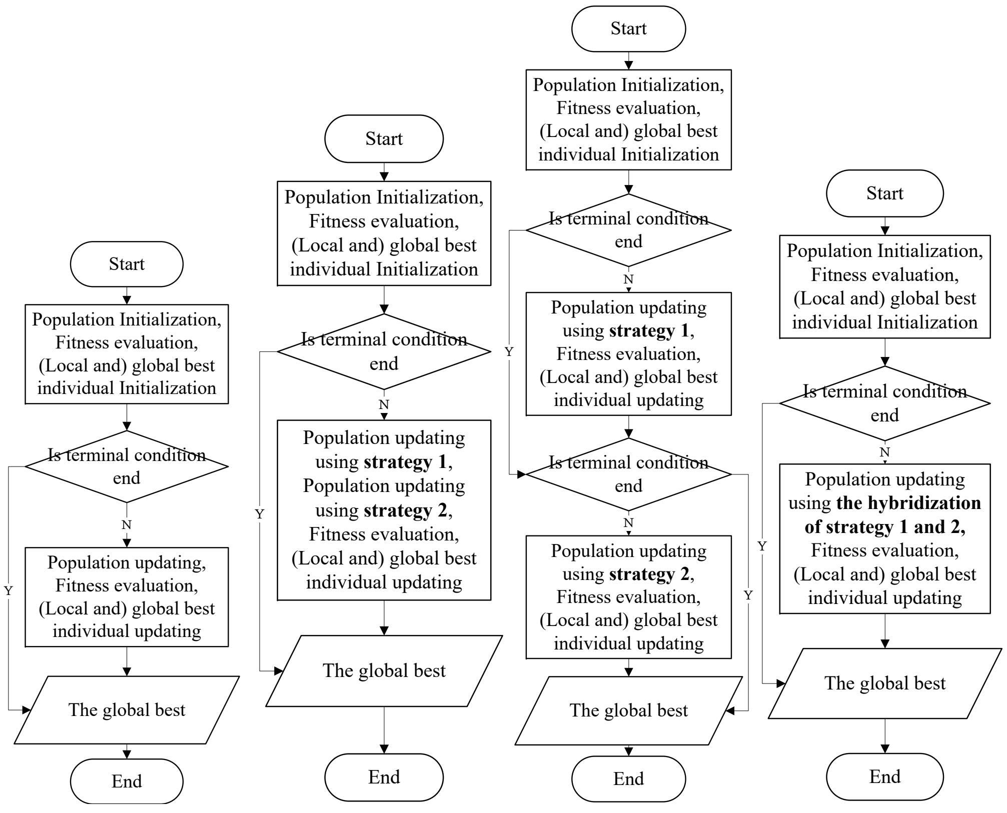
1. Introduction

Distributed computing systems, especially cloud computing, have been applied widely in production and science research. Task scheduling is one of the most concerning open issues for various distributed systems [1] for optimizing the resource efficiency [2], user satisfaction [3], energy efficiency [4,5], etc. In a distributed computing system, there are multiple multi-core servers with various capacities for processing computing tasks. The task scheduling is to decide which computing unit (e.g., computing cores) will be where each task is processed and the execution order of tasks assigned on each computing unit.

The task scheduling problem has been proven NP-hard when there are more than two computing units [6]. This means that there is no exact method to solve the scheduling problem within a reasonable time for large-scale computing systems unless NP = P because the time complexity of an exact method is exponentially increased with the system scale. Thus, related works mainly focus on designing heuristic or meta-heuristic search approaches to provide an accepted solution with polynomial time for the problem.

A heuristic method exploits the local search approach designed for a specific problem [7,8], to find a local optimal solution with very little time consumption. Inspired by natural laws or social phenomena, meta-heuristics employ general search strategies implemented with random methods [9,10]. Benefits from global search abilities, meta-heuristics can achieve better performance than heuristics with a few time overheads in many cases and are widely applied in various decision-making situations. Thus, in this paper, we are concerned with the design of meta-heuristics for solving the task scheduling problem.

There are numerous meta-heuristic methods, and each has its own advantages and limitations. The hybrid of two or more methods can achieve complementary advantages of these methods and thus can provide better performance than each of them. There are three hybridization methods for combining two or more heuristic or meta-heuristic methods, as shown in Figure 1. The first one is sequentially exploiting the population updating strategies of two or more algorithms in each iterative or evolutionary process (as shown in Figure 1b). The second one is first performing the population strategies of one algorithm and then another algorithm (as shown in Figure 1c). Both of these two hybridization methods sequentially exploit rather than combine the benefits of multiple meta-heuristic algorithms and thus have limited or even worse performance. Therefore, we consider exploiting the third hybridization method, which integrates two or more updating strategies, to propose a hybrid heuristic task scheduling algorithm for distributed computing by integrating the swarm cognition into the evolutionary strategy.


Start
Population Initialization, Fitness evaluation, (Local and) global best individual Initialization
Is terminal condition end
N
Y
Population updating, Fitness evaluation, (Local and) global best individual updating
Population updating using strategy 1, Population updating using strategy 2, Fitness evaluation, (Local and) global best individual updating
Population updating using strategy 1, Fitness evaluation, (Local and) global best individual updating
Population updating using strategy 2, Fitness evaluation, (Local and) global best individual updating
Population updating using the hybridization of strategy 1 and 2, Fitness evaluation, (Local and) global best individual updating
The global best
End

(**a**) (**b**) (**c**) (**d**)

Figure 1. The hybridization methods for combining two or more heuristic or meta-heuristic methods. (**a**) Basic framework. (**b**) Hybridization 1. (**c**) Hybridization 2. (d) Hybridization 3.

In this paper, we design a task scheduling method by integrating a Genetic Algorithm (GA) with Particle Swarm Optimization (PSO). A GA and PSO are both representative and classical meta-heuristics and have been widely used in in many areas, thanks to their good performance and easy implementations [11,12]. GA has a powerful global search ability due to the large population diversity produced by the crossover and mutation operators. However, a GA has a slow convergence velocity in the search process. Conversely, PSO quickly converges but is easily trapped into a local optimal position, where the position of each particle is updated based on its personal best position and the global best position of all particles in the evolution process. Therefore, in this paper, we combine both benefits of a GA and PSO by integrating the updating idea of PSO into the evolution process of GA, and propose a new hybrid heuristic method (PGA) for task scheduling in distributed computing systems.

The contributions of this paper can be briefly summarized as follows.

- We formulate the task scheduling problem with deadline constraints into a binary nonlinear programming model (BNLP) for distributed computing systems. There are two optimization objectives of BNLP. The major optimization objective is maximizing the number of tasks with deadline satisfaction, which is one of the metrics quantifying user satisfaction. The minor one is maximizing the overall resource utilization, which is a widely used quantitative metric for resource efficiency.
- We design a hybrid heuristic method for solving the task scheduling problem named PGA. In the PGA, each chromosome represents one solution mapping tasks onto computing units, and the fitness function is the objective of formulated BNLP. In the evolution process of the GA, the PGA adds two crossover operations to speed up the evolutionary rate, inspired by PSO, which crosses the current chromosome with its own historical best chromosome and the global best chromosome, respectively, for each individual.
- To evaluate the performance of the PGA, we conduct extensive simulated experiments, where the system parameters are set referring to related works. Experiment results confirm the performance superiority of the PGA in both user satisfaction and resource efficiency, compared with seven heuristic/meta-heuristic/hybrid heuristic task scheduling methods proposed recently.

In the following of this paper, Sections 2 and 3 illustrate our formulated model and proposed hybrid heuristic task scheduling method, respectively. In Section 4, we evaluate the performance of our proposed method. Section 5 gives recent works related to solving the task scheduling problem, and Section 6 concludes this paper.

2. Problem Formulation

In this section, we present the formulation of the task scheduling problem concerned in this paper. The notations used in the formulation are outlined in Table 1.

In a distributed computing system, there are S computing servers, $s_1, s_2, \ldots, s_S$. The server s_j has n_j cores, each with g_j computing capacity. During a period of time, there are T tasks requesting processing on the system represented by $t_1, t_2, \ldots, t_T$. The task t_i needs c_i computing resource for its processing and requires d_i deadline, i.e., the system finishes the task before d_i or just rejects it (In this paper, we consider tasks with hard deadline constraints, and leave the concern of soft deadline requirements as one of our future works). Without loss of generality, we assume that $d_1 \leq d_2 \leq \ldots \leq d_T$. Then, we t_i is assigned to one core of s_j; its execution time is c_i/g_j.

In this paper, we consider independent computing-intensive tasks, i.e., there is no data or logical dependency between the two tasks. This is because independent tasks, such as bag-of-tasks (BoT) applications, are very common in distributed systems, and it is worth designing a scheduling method for independent tasks [13]. In the future, we will study the extension of our work and the new approach for the scheduling of workflow tasks with dependencies on each other. In fact, our model and method (PGA) can be applied for scheduling interdependent tasks by just adding their dependency constraints.

For the formulation of the task scheduling problem, we define binary variables $x_{i,j,k}$ to represent whether t_i is assigned to kth core of s_j, as shown in Equation (1).

$$x_{i,j,k} = \begin{cases} 1, & \text{if } t_i \text{ is assigned to the } k\text{th core in } s_j \\ 0, & \text{otherwise} \end{cases}$$
$$1 \leq i \leq T, 1 \leq j \leq S, 1 \leq k \leq n_j. \tag{1}$$

Then, the tasks assigned to kth core of s_j include $\{t_i | x_{i,j,k} = 1\}$. In this paper, we do not use the execution redundancy that can improve the makespan of tasks but reduce the resource efficiency. Thus, a task cannot be assigned to more than one core for its execution, i.e.,

$$\sum_{j=1}^{S}\sum_{k=1}^{n_j} x_{i,j,k} \le 1, 1 \le i \le T. \tag{2}$$

Table 1. The notations used in our formulation.

Notation	Description
T	The number of tasks
t_i	The ith task
c_i	The computing resource amount required by t_i
d_i	The deadline required by t_i
f_i	The finish time of t_i
S	The number of computing servers
s_j	The jth server
n_j	The number of cores equipped in s_j
g_j	The computing capacity of each core in s_j
o_j^S	The occupied time of s_j for task executions
$o_{j,k}$	The use time of kth core in s_j
u_j^S	The resource utilization of s_j
U	The overall resource utilization of the system
N	The number of tasks with deadline satisfactions

For tasks assigned to one core, the number of tasks whose deadlines are satisfied is optimal when applying the earliest deadline first (EDF) execution order [14]. With an EDF execution order, the finish time of each task can be calculated by Equation (3).

$$f_i = \sum_{j=1}^{S}\sum_{k=1}^{n_j}(x_{i,j,k} \cdot \sum_{i'=1}^{i}(x_{i',j,k} \cdot \frac{c_{i'}}{g_j})), 1 \le i \le T \tag{3}$$

where f_i represents the finish time of t_i when the task is assigned to a core for its execution, and $f_i = 0$ if t_i is not assigned by any one core when the task is rejected. $\sum_{i'=1}^{i-1}(x_{i',j,k} \cdot c_{i'}/g_j)$ is the cumulative execution time of tasks that have earlier deadlines than t_i and are assigned to kth core of s_j. Thus, $\sum_{i'=1}^{i}(x_{i',j,k} \cdot c_{i'}/g_j)$ is the finish time when t_i is assigned to the kth core of s_j. Then, the deadline constraints required by tasks can be formulated as Equation (4). Equation (4) also holds for rejected tasks.

$$f_i \le d_i, 1 \le i \le T. \tag{4}$$

For each core, the use time is the cumulative time consumed by executing tasks assigned to the core, which can be formulated as Equation (5), where $o_{j,k}$ is the use time of kth core in s_j. The occupied time of a server is the maximum use time of its cores and thus can be calculated by Equation (6), where o_j^S is the occupied time of s_j for task executions.

$$o_{j,k} = \sum_{i=1}^{T}(x_{i,j,k} \cdot \frac{c_i}{g_j}), 1 \le j \le S, 1 \le k \le n_j. \tag{5}$$

$$o_j^S = \max_{k=1}^{n_j}\{o_{j,k}\}, 1 \le j \le S. \tag{6}$$

Then the resource utilization of each server can be obtained from Equation (7), and the overall resource utilization of the distributed system can be achieved using Equation (8), where u_j^S is the resource utilization of s_j. U is the overall resource utilization. $\sum_{k=1}^{n_j} o_{j,k} \cdot g_j$ is the amount of resources consumed by task executions in s_j, and $o_j^S \cdot n_j \cdot g_j$ is the occupied resource amount of s_j.

$$u_j^S = \frac{\sum_{k=1}^{n_j} o_{j,k} \cdot g_j}{o_j^S \cdot n_j \cdot g_j} = \frac{\sum_{k=1}^{n_j} o_{j,k}}{o_j^S \cdot n_j}, 1 \le j \le S. \tag{7}$$

$$U = \frac{\sum_{j=1}^{S} (\sum_{k=1}^{n_j} o_{j,k} \cdot g_j)}{\sum_{j=1}^{S} (o_j^S \cdot n_j \cdot g_j)}. \tag{8}$$

Based on the above formulations, the task scheduling problem can be modeled as the following optimization problem.

$$\text{Maximizing } N + U, \tag{9}$$

subject to

$$N = \sum_{i=1}^{T} \sum_{j=1}^{S} \sum_{k=1}^{n_j} x_{i,j,k}, \tag{10}$$

$$\text{and Equations (1)–(8)}$$

where $x_{i,j,k}$, $1 \le i \le T, 1 \le j \le S, 1 \le k \le n_j$, are the decision variables. The objective is maximizing the number of tasks with deadline meets (N) plus the overall resource utilization (U), where N is calculated by Equation (10). Noticing that the utilization U is no more than 1, the number of tasks with a deadline meeting N is the major optimization objective, and the utilization U is the minor one. Because the decision variables are binary and constraints (3), (6) and (8) are non-linear, the optimization problem belongs to a constrained binary non-linear programming problem (BNLP). This problem can be solved by existing tools, e.g., lp_solve [15] and the optimization toolbox of MathWorks [16]. However, these tools take exponential time to solve a BNLP, on average, and thus cannot be applied for middle to large-scale problems.

3. PGA: The Hybrid Heuristic Scheduling Method

In this section, we present a hybrid heuristic method for solving the task scheduling problem formulated in the previous section, based on the ideas of a GA and PSO. Our proposed method, the PGA, is outlined in Algorithm 1, which integrates the idea of self-cognition and social cognition of PSO into GA. We need to design the encoding/decoding method for the map between chromosomes and scheduling solutions first when applying GA. The encoding/decoding method used in our method is as follows. In our method, similar to PSO instead of a GA, individual and chromosome are two different things. Chromosomes are to individuals what positions are to particles. An individual has only one chromosome, and its chromosome can be changed with population evolutions.

Based on the encoding/decoding method, the PGA first initializes a population with multiple individuals (line 1), where each gene value of the chromosome for every individual is set between 1 and the core number, randomly. Moreover, the PGA evaluates the fitness of each individual (line 2). The fitness function is the objective of the optimization problem formulated in Section 2, i.e., $N + U$, where N and U can be get from the task scheduling solution decoded by the chromosome. Then, the PGA records the current chromosome as the personal best chromosome for each individual (line 3), and the chromosome with the best fitness as the global best chromosome (line 4).

After the initialization, the PGA evolves the population iteratively with crossover, mutation, and selection operators. In each evolution, the PGA first invokes the crossover operator on each individual with a probability (the crossover probability) three times (line 7), which crosses its chromosome with the chromosome of another individual, its personal best chromosome, and the global best chromosome, respectively. Each time the crossover operator is invoked, two new chromosomes (offspring) are produced, and there are six new chromosomes are produced by the crossover operator for each chromosome. For each new chromosome, the PGA evaluates its fitness and updates the personal best chromosome and the global best chromosome as it if it had better fitness (lines 8–10).

For each individual, the PGA performs the mutation operator on its chromosome with a probability (the crossover probability), which can produce a new chromosome (line 11). Moreover, the PGA evaluates the fitness of the new chromosome and updates the personal and the global best chromosomes as the new one when the new one has better fitness for the individual (line 12).

At the end of each evolution, for each individual, the PGA uses the selection operator to select a chromosome as its chromosome for the next round of evolution (line 14). For an individual, the candidate chromosomes of the selection include its current chromosome, six new chromosomes produced by the crossover operator, and the new chromosome by the mutation operator.

After the evolution phase finishes, the PGA returns the task scheduling solution decoded by the global best chromosome (lines 15 and 16). The maximum evolution generation is set for the terminal condition in this paper.

In the remainder of this section, we will illustrate the encoding/decoding method and the operators used in the evolution, respectively.

Algorithm 1 PGA: the hybrid heuristic scheduling method

```
Require: The information of tasks and computing servers, the parameters of GA;
Ensure: the solution of task scheduling;
 1: Initializing a population (individuals), i.e., setting the chromosome of each individual randomly;
 2: Evaluating the fitness of each individual;
 3: Recording the personal best chromosome ($pb$) as the initialized one for each individual;
 4: Setting the global best chromosome ($gb$) as the chromosome with the best fitness in the population;
 5: while The terminal condition is not met do
 6:     for Each individual do
 7:         Crossing its chromosome with the chromosome of another individual that is randomly
            selected, $pb$, and $gb$, with a certain probability, respectively, and producing six new chromosomes
            (each crossover operator produces two new chromosomes);
 8:         Evaluating fitnesses of six new chromosomes, and getting the best chromosome ($\wp$) that
            has the best fitness from these six chromosomes;
 9:         If the fitness of $\wp$ is better than that of $pb$, then $pb$ is updated as $\wp$;
10:         If the fitness of $\wp$ is better than that of $gb$, then $gb$ is updated as $\wp$;
11:         Mutating its chromosome with a certain probability, which produces one new chromosome;
12:         Evaluating the fitness of the new chromosome and updating $pb$ and $gb$ if the new chromo-
            some has a better fitness, respectively.
13:     for Each individual do
14:         Selecting one chromosome from its current chromosome and seven new chromosomes
            produced by it by crossover and mutation operations (lines 7 and 11), which is set as the new
            chromosome of the individual for the next evolution;
15: Decoding $gb$ into a task scheduling solution;
16: return the task scheduling solution;
```

3.1. The Encoding or Decoding Method

In a PGA, a chromosome is expressed by a vector, and its genes are the values of all dimensions. For each chromosome, genes have a one-to-one correspondence with tasks. The possible value of every gene is between 1 and the number of computing cores in the system, which identifies the core where the corresponding task is assigned. Then, given a chromosome, we can decode an assignment of tasks to cores. Moreover, with the EDF scheme for executing tasks in each core, we can get a task-scheduling solution from a chromosome. Next, we give an example to help readers understand the encoding/decoding method.

Assuming a distributed system consisting of 2 computing cores, there are 4 tasks that need to be processed. Then there are 4 genes in each chromosome, which respectively correspond to these 4 tasks. The possible value of each gene includes 1 and 2, which represent the core where the corresponding task is assigned. The chromosome with the code (1, 1, 2, 2) represents that the first and the last two tasks are assigned to the first and the second cores, respectively. For two tasks assigned to a core, the task with the earlier deadline will be executed first.

3.2. The Crossover Operator

In this paper, we use the uniform crossover operator to produce two offspring from two chromosomes, which helps to enhance the exploitation ability of the PGA [17]. The uniform crossover operator swaps two genes between two chromosomes with a probability in each dimension (gene position). In the implementation, for each gene position, a random number between 0 and 1 is generated with a uniform distribution, and two genes of two chromosomes are swapped in the position if the random number is less than the pre-defined probability. In the task scheduling problem, the swap between two genes of two chromosomes means that the cores that the corresponding task is assigned to are swapped in corresponding scheduling solutions.

As an example in Figure 2, the uniform crossover operator is performed on two chromosomes (4, 5, 9, 2, 5, 10, 10, 8, 3) and (7, 6, 8, 1, 7, 2, 1, 8, 4), and the genes of chromosomes are swapped in the second, fourth, and seventh dimensions. Two offspring (new chromosomes) are produced, which are (4, 6, 9, 1, 5, 10, 1, 8, 3) and (7, 5, 8, 2, 7, 2, 10, 8, 4).

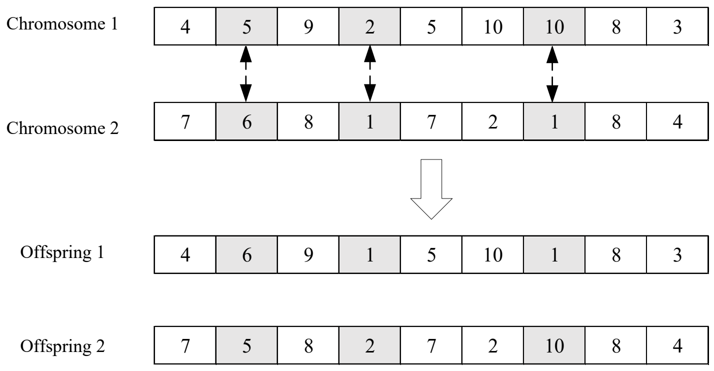


Figure 2. An example for illustrating the uniform crossover operator.

There are two parameters (probabilities) that need to be set for the crossover operator. One for deciding whether to perform the crossover operator for every individual and another for each dimension. Both parameters have an impact on the population diversity and convergence of the PGA. When the probability is great, there will be more individuals or genes crossed, and more new chromosomes will be generated, which can increase the population diversity. However, the increase of the diversity is limited, as the crossover operator cannot generate new genes in a dimension. However, this helps to pass good genes to the next generations and thus speeds the convergence. Therefore, these parameters should be adaptive to population diversity and convergence, which is one of our future works. In this paper, we set both probabilities as the same value.

3.3. The Mutation Operator

For each individual, a mutation operator is performed on its chromosome with a probability. In this paper, we use the uniform mutation operator as it helps to maintain population diversity. Similar to the uniform crossover operator, for a chromosome mutating, the uniform mutation operator generates a random number in the range of [0, 1] for each gene position and mutates the gene of the chromosome in the position if the random number is less than pre-set probability. The mutation of a gene is to set its value as another possible one that can be randomly generated. In the task scheduling problem, the mutation of a gene changes the core that the corresponding task is assigned to.

For example, as shown in Figure 3, the uniform mutation operator is performed on the chromosome (10, 9, 1, 6, 10, 6, 5, 1, 7), and mutates the second and fifth genes to 4 and 2, respectively. Then we can get a new chromosome (10, 4, 1, 6, 2, 6, 5, 1, 7).

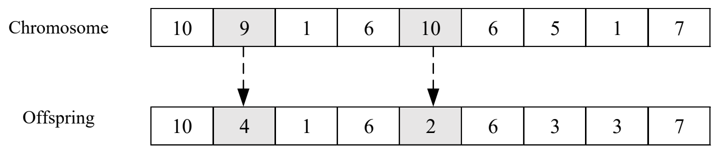


Figure 3. An example for illustrating the uniform mutation operator.

The mutation operator can largely increase the population diversity, as it can generate new genes in each dimension, but this decreases the convergence rate. Thus, the mutation probability also should be set carefully. Intuitively, all of the crossover, mutation, and selection operators affect diversity and convergence, and they all should be adaptively configured. In addition, it is very probable that these operators are interconnected in their performance, which is seldom studied.

3.4. The Selection Operator

At the end of each evolution, the PGA constructs a new population from the current population, and new chromosomes are produced using the crossover and mutation operators, by the selection operator, for the next evolution. To be specific, the PGA performs the roulette wheel selection on each individual. For each individual, there is one current chromosome and seven produced chromosomes at the end of each evolution. Then the PGA uses the roulette wheel selection to select one chromosome from these eight chromosomes as the new chromosome of the individual, where the probability that a chromosome is selected is proportional to its fitness.

For the selection operator, there is a tradeoff between exploration and exploitation that needs to be considered. For a chromosome with high fitness, it has a great probability of being selected for the next evolution and vice versa. This helps to preserve the good gene and make the convergence fast but decreases the population diversity to a certain extent. One feasible idea is to guarantee diversity in the early stage and to ensure strong convergence in the late stage. One of our future works is studying adaptive selection according to diversity and convergence.

3.5. Time Complexity Analysis

As illustrated in the previous subsections, the crossover, mutation, and selection operators traverse all dimensions, and thus, each operator has $O(T)$ time complexity. From Algorithm 1, we can see that, in each iteration of the population evolution, the crossover operator is performed three times (line 7) and the mutation operator one time (line 11), one for each individual. Thus, each iteration has $O(T \cdot D)$ time complexity, where D is the number of individuals. Then, the time complexity of the PGA is $O(T \cdot D \cdot ITE)$, where ITE is the number of iterations. Compared with a GA, the PGA performs the crossover operator two more times and, thus, has the same time complexity as a GA and PSO.

4. Performance Evaluation

In this section, we evaluate the performance of our method with extensive simulated experiments. We first present the experiment environment used for the performance evaluation and then analyze the experiment results in the following.

4.1. Experiment Environment

To evaluate the performance of our PGA, we simulate a distributed system referring to some related works [18–20]. The parameters of our simulated experiments are shown in Table 2. In the simulated system, there are 50 servers; each has 2–64 cores. The computing capacity of each core in a server is set in the range of [1000, 4000] Million Instructions Per Second (MIPS) randomly. One thousand tasks are generated to be processed by the distributed system. For each task, the amount of its required computing resource is [1000, 100,000] Million Instructions (MI), and its deadline is set as [1, 100] seconds, randomly. To evaluate the energy consumed by servers, we use the most popular linear model, where the consumed energy (E_j) can be estimated with the resource utilization for each server using Equation (11). Where P_j^{idle} and P_j^{full} are the consumed power when the server runs at idle and full loads, respectively. $u(\tau)_j$ is the resource utilization changed with time τ.

$$E_j = \int_\tau (P_j^{idle} + (P_j^{full} - P_j^{idle}) \cdot u(\tau)_j) d\tau, 1 \le j \le S. \tag{11}$$

The power consumed by each server is shown in Table 3, which is set according to the core number, referring to [21–23].

Table 2. The parameters of our simulated experiments.

Parameter	Value
server number	50
core number of each server	[2, 64]
computing capacity of each core	[1000, 4000] MIPS
computing resource requirement of each task	[1000, 100,000] MI
deadline of each task	[1, 100] seconds
mutation probability	0.1
crossover probability	0.5
acceleration coefficients of PSO	2.0
(linearly decreased) inertia weight of PSO	[0.5, 1.2]

Table 3. The power of a server.

The Core Number	P^{idle}	P^{full}
$[1, 8)$	110	175
$[8, 16)$	125	210
$[16, 24)$	210	300
$[24, +\infty)$	350	500

The compared methods used in our experiments include the following.

HC The hill climbing method (HC) is one of the commonly used meta-heuristics for task scheduling [24,25]. Given a starting position, HC searches for a neighbor of the current position until there is no neighbor that has better fitness, with the scheme of depth-first search.

GA The basic idea of [26], which exploits the principle of a Genetic Algorithm (GA) for scheduling tasks.

GAHC The method used by Hussain and Al-Turjman [20] uses the idea of HC to improve the quality of each chromosome at the beginning of each evolution for the GA. Instead of the selection operator of a GA, GAHC replaces the chromosome with its offspring with better fitness after each of the crossover and mutation operators.

PSO Particle swarm optimization (PSO) is also one of the most popular approaches for the task scheduling problem, e.g., [27].

PSO_M This is the work proposed by Hafsi et al. [28], which integrates the mutation operator into PSO. PSO_M adds the mutation operator for each particle at the end of each iteration of PSO.

GA+PSO The method used in [29] exploits a GA and PSO in the first half and the second half of the evolution phase, respectively.

GA_PSO The method proposed by Wang et al. [30] uses a GA and PSO in each iteration of the population evolution.

We compare our method with the above methods in the following three aspects.

User Satisfaction The satisfaction strongly affects the income and reputation of operating a distributed system [31], which usually is quantified by the completion rates of tasks. In this paper, we use the following two metrics for the quantification of the satisfaction, the number (N) and the computing size ($\sum_{i=1}^{T}\sum_{j=1}^{S}\sum_{k=1}^{n_j}(x_{i,j,k}\cdot c_i)$) of tasks with deadlines met, where the computing size of a task is the number of computing resources required by the task.

Resource Efficiency In general, the task load finished by consuming per unit of resources can be applied to quantifying the resource efficiency in distributed systems. Two metrics are used in our experiments for evaluating the resource efficiency, which is the overall resource utilization (U) and energy efficiency ($\frac{\sum_{i=1}^{T}\sum_{j=1}^{S}\sum_{k=1}^{n_j}(x_{i,j,k}\cdot c_i)}{\sum_{j=1}^{S}E_j}$). Energy efficiency is the processed computing size per unit of energy, which is the ratio of processed computing size to the consumed energy.

Processing Efficiency The processing efficiency can reflect the speed-up of the distributed processing system. In this paper, the processing efficiency is quantified by dividing the processed computing size by the makespan of processed tasks, i.e., the processing speed, which is calculated by $\frac{\sum_{i=1}^{T}\sum_{j=1}^{S}\sum_{k=1}^{n_j}(x_{i,j,k}\cdot c_i)}{\max_{i=1}^{T}\{f_i\}}$.

In our experiment, we repeat the following procedures 100 times: generating a distributed system, measuring each method in every metric, and normalizing each metric value of each method by dividing it by that of HC to highlight the relative performance of different methods. In the following, we report the normalized (relative) performance in each metric.

4.2. Experiment Results

4.2.1. User Satisfaction

Figures 4 and 5 give the relative performance of various methods in user satisfaction. As shown in these figures, we can see that the PGA completes 27.9–55.8% more tasks and 32.8–65.4% larger computing size in distributed systems, on average. This verifies that the PGA can achieve better user satisfaction than other methods. The main benefit of the PGA is exploiting the self-cognition and social cognition idea of PSO in a GA, which not only ensures the powerful exploration ability of a GA but also makes up for the weak exploitation ability with PSO. Figures 4 and 5 show the PGA has a much better performance than a GA and PSO, which indicates the high integration efficiency of our method.

The distributed systems process about 50% more tasks and about 60% larger computing size, on average, when applying the PGA, compared with PSO_M, GA+PSO, and GA_PSO. These four methods are all designed by combining PSO and GA. Thus, this can prove that our method has much higher efficiency in the integration of a GA and PSO for designing hybrid heuristic algorithms.

The experimental results show that HC has the worst user satisfaction with these methods. This is mainly because HC is a search strategy with a single individual, where its performance is sensitive to the searching starting point, and it is easy to fall into a local optimal location. Other methods are all searching with a population, which can exploit the cooperation among multiple individuals to improve the global search ability. Individual search strategies like HC can be used to improve the quality of each individual for population meta-heuristic algorithms for better performance. For example, GAHC has 17.9% and 21.1% better performance than a GA by applying HC for each individual in the number and computing size of processed tasks, respectively, as shown in Figures 4 and 5. However, local search strategies may decrease the exploration ability of each individual and thus increase the probability of trapping into local optimums of other meta-heuristic methods. One of our future works is to study such impacts of individual search strategies on the performance of population meta-heuristic algorithms for integrating them.

From Figures 4 and 5, we can also see that PSO_M, GA + PSO, and GA_PSO have comparable performance. The main reasons are as follows. The performance of a GA and PSO are similar. PSO_M, GA + PSO, and GA_PSO all combine a GA and PSO. GA+PSO and GA_PSO separately use a GA and PSO, which cannot achieve a better performance than either of them. PSO_M operates a mutation on each particle in every iteration for PSO. This can help a particle to escape from a local optimal position but may decrease the convergence rate. Therefore, the integration approach of multiple meta-heuristics should be carefully designed for good performance.

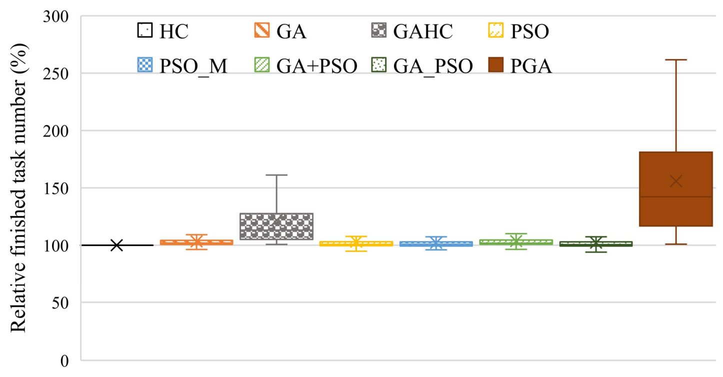


Figure 4. The relative numbers of tasks with deadline meets when applying various task scheduling methods.

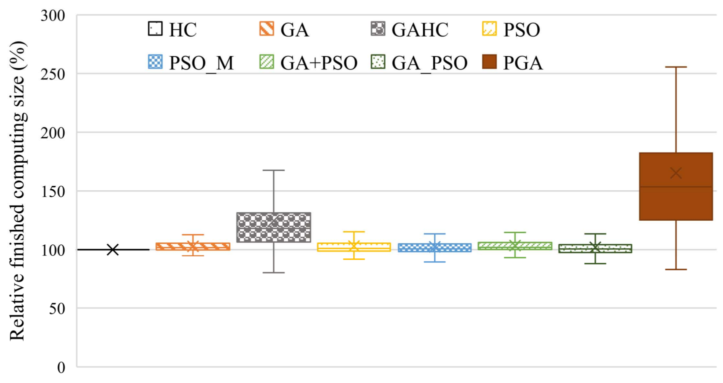


Figure 5. The relative computing sizes of tasks with deadline meets when applying various task scheduling methods.

4.2.2. Resource Efficiency

Figures 6 and 7 show the relative resource utilizations and energy efficiencies achieved using various task scheduling methods. The PGA achieves 33.8–69.6% higher utilization and 25.7–57.3% better energy efficiency than the compared methods. This proves that our method has a high resource efficiency for scheduling tasks in distributed systems by combining a GA and PSO. The reason that our method has higher resource utilization and energy efficiency than other methods is as follows. Compared with other methods, the PGA processes more computing size, which means more resources are consumed by processing tasks. However, the PGA consumes comparable time for finishing these tasks with other methods, which implies that comparable resources are occupied for processing these tasks. Thus, the resource utilization achieved by the PGA is higher than by other methods. A high resource utilization often means a good load balance among computing cores and, thus, a low energy waste.

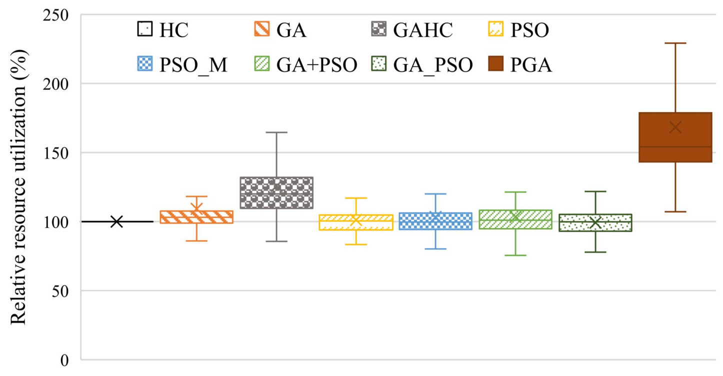


Figure 6. The relative resource utilizations achieved by various task scheduling methods.

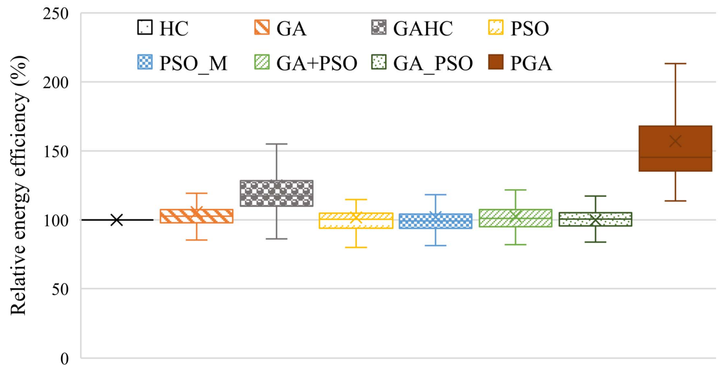


Figure 7. The relative energy efficiencies achieved by various task scheduling methods.

4.2.3. Processing Efficiency

Figure 8 presents the processing efficiencies of distributed systems when applying different task scheduling methods. As shown in this figure, the PGA has 53.1–112.3% higher processing efficiency than compared methods. This is because the PGA processes more task loads with a similar time compared with other methods. This phenomenon indicates that the PGA provides a higher speed-up than others and thus can be applied for distributed systems very well.

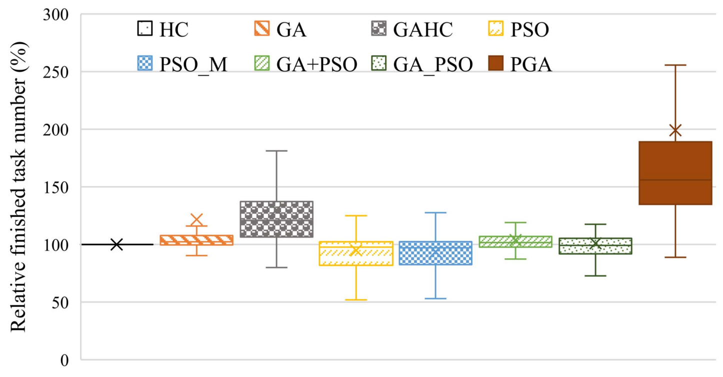


Figure 8. The relative processing efficiencies achieved by various task scheduling methods.

5. Related Works

Distributed systems, e.g., cloud computing and cluster computing, have been applied in all walks of life, which benefit from the rapid development of information and communication technologies. Moreover, there are plenty of research works on task scheduling for improving user satisfaction and/or resource efficiency in various distributed systems. The characteristics of related works are outlined in Table 4.

Table 4. The characteristics of related works.

Works	Used Algorithm	Hybrid
Wang et al. [32]	designed heuristic	no
Ma et al. [33]	designed heuristic	no
Mangalampalli et al. [34]	CSO	no
Otair et al. [35]	designed population optimization	no
Teraiya and Shah [27]	PSO	no
Chandrashekar et al. [36]	ACO	no
Yeh et al. [37]	SSO	no
Nwogbaga et al. [29]	GA and PSO	H2
Sharma et al. [38]	PSO and ACO	H2
Wang et al. [30]	GA and PSO	H1
Kumar and Karri [39]	EOA and EFO	H1
Cheikh and Bougara [40]	PSO and extremal optimization	H2
Hussain and Al-Turjman [20]	GA and HC	H1
Hafsi et al. [28]	PSO and the mutation operator of GA	H1
Chhabra et al. [41]	OBL, PSO, and WOA	H1 + H2
Ours	GA and PSO	H3

"hybrid" column represents the hybridization strategy used by the related works. "no" means the work used a single heuristic or meta-heuristic algorithm. "H1/2/3" represents hybridization 1/2/3 as shown in Figure 1.

Due to the NP-hard nature of the task scheduling problem, existing works mainly exploited heuristics and meta-heuristics to solve the problem. For example, Wang et al. [32] presented a heuristic approach to improve user satisfaction and the resource cost for a hybrid cloud which consists of private and public clouds. This work assigned tasks whose requirements could not be satisfied by the public cloud to the private cloud at first, considering the scarcity of private resources. The heuristic method proposed by Ma et al. [33] iteratively assigned the next task to the server with the most available resources for load balancing. Mangalampalli et al. [34] used Cat Swarm Optimization (CSO) algorithm to map tasks and virtual machines in cloud computing for minimizing makespan, energy consumption, and total power cost at data centers. IMOMVO [35] designed an improved multi-verse optimizer, a novel population optimization technique, for task scheduling in cloud computing to improve task execution performance. Teraiya and Shah [27] used PSO to design the new scheduling approach, which considered each task as a particle and exploited the PSO technique to identify the most critical task for a prior execution. Chandrashekar et al. [36] exploited Ant Colony Optimization (ACO) for task scheduling in cloud computing to improve the makespan and the resource cost. Yeh et al. [37] applied Simplified Swarm Optimization (SSO) to reduce the energy consumption and task computing time for cloud computing.

Each of the above works used one heuristic or meta-heuristic algorithm and did not exploit the complementary advantages of multiple algorithms for better task-scheduling solutions. Some studies considered combining a meta-heuristic algorithm with another heuristic or meta-heuristic algorithm for task scheduling. The method used in [29] exploited a GA in the first half of the evolution phase and PSO in the second half. With the same hybridization strategy to [29], Sharma et al. [38] used PSO and ACO to reduce the latency in fog computing. Wang et al. [30] continuously used a GA and PSO in each iteration of the evolution, and Kumar and Karri [39] combined the earthworm optimization algorithm (EOA) and the electric fish optimization algorithm (EFO) in the same way. Cheikh and Bougara [40] first used PSO to get a solution for task scheduling. Then, with the initialization of this solution, their method applied extremal optimization for a new solution. These above works exploited two meta-heuristic algorithms separately and thus didn't take full advantage of the combination of these two algorithms. The method used by Hussain and Al-Turjman [20] performed the HC operation on each parent to generate parents with better fitnesses for a GA at the beginning of each evolution and replaced the selection operator by the replacement operator that the parent is replaced with its offspring with better fitness. Hafsi et al. [28] performed the mutation operator on each particle at the end of each iteration of PSO to overcome the drawback of easily trapping into local minima. Chhabra et al. [41] proposed a hybrid heuristic method by incorporating opposition-based learning (OBL) and PSO into Whale optimization algorithm (WOA), for scheduling BoT applications on physical cloud resources to minimize the makespan and energy consumption. This work used OBL to generate the initial population and applied OBL and the velocity update mechanism of PSO on whale exploration solutions, respectively, in the exploration phase of WOA.

In this paper, different from existing works, we proposed a hybrid heuristic method by integrating the principle idea of PSO into a GA, which combines both advantages of a GA and PSO very well, to optimize user satisfaction and resource efficiency with deadline constraints with task scheduling.

6. Conclusions

In this paper, we focus on the task scheduling problem with deadline constraints in various distributed computing systems. To address the problem, we formulate it as BNLP and propose a hybrid heuristic method, PGA, for solving the problem. The PGA combines both advantages of a GA and PSO by integrating the self-cognition and social cognition idea of PSO into the evolution of GA. Simulated experiments are conducted, and the results verify the superior performance of the PGA in user satisfaction, resource efficiency, and processing efficiency. The task scheduling is an instance of discrete optimization problems, and the PGA performs well in its solving. We believe that the PGA can be also applied to solving other discrete optimization problems as well as solving task scheduling, which is one of our future works.

Even though the integration of multiple meta-heuristic methods has the opportunity to provide a hybrid heuristic with good performance, some meta-heuristics are not complementary, which may not improve or even degrade the performance by hybridizing them. Furthermore, the integration strategy has an impact on performance. Therefore, in the future, we will study the complementarity of different meta-heuristics and design an efficient integration strategy for the hybrid of multiple meta-heuristics to improve the performance of the distributed systems in various aspects.

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

Funding: The research was supported by the key scientific and technological projects of Henan Province (Grant No. 202102210174), the National Natural Science Foundation of China (Grant No. 61872043), Qin Xin Talents Cultivation Program, Beijing Information Science & Technology University (No. QXTCP B201904), the fund of the Beijing Key Laboratory of Internet Culture and Digital Dissemination Research (Grant No. ICDDXN004).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

Abbreviations

The following abbreviations are used in this manuscript:

ACO	Ant Colony Optimization
BNLP	Binary non-linear programming
BoT	Bag-of-tasks
CSO	Cat Swarm Optimization
EFO	Electric Fish Optimization
EOA	Earthworm Optimization Algorithm
GA	Genetic algorithm
GAHC	GA with hill climbing
HC	Hill climbing
MINLP	Mixed-integer non-linear programming
MI	Million instructions
MIPS	Million instructions per second
OBL	Opposition-based learning
PGA	Hybrid GA and PSO
PSO	Particle swarm optimization
PSO_M	PSO with mutation operator
SSO	Simplified Swarm Optimization
WOA	Whale optimization algorithm

References

1. Jamil, B.; Ijaz, H.; Shojafar, M.; Munir, K.; Buyya, R. Resource Allocation and Task Scheduling in Fog Computing and Internet of Everything Environments: A Taxonomy, Review, and Future Directions. *ACM Comput. Surv.* **2022**, *54*, 1–38. [CrossRef]
2. Dai, H.; Wu, J.; Wang, Y.; Xu, C. Towards scalable and efficient Deep-RL in edge computing: A game-based partition approach. *J. Parallel Distrib. Comput.* **2022**, *168*, 108–119. [CrossRef]
3. Sang, Y.; Cheng, J.; Wang, B.; Chen, M. A three-stage heuristic task scheduling for optimizing the service level agreement satisfaction in device-edge-cloud cooperative computing. *PeerJ Comput. Sci.* **2022**, *8*, e851. [CrossRef]
4. Peng, J.; Li, K.; Chen, J.; Li, K. HEA-PAS: A hybrid energy allocation strategy for parallel applications scheduling on heterogeneous computing systems. *J. Syst. Archit.* **2022**, *122*, 102329. [CrossRef]
5. Ghafari, R.; Kabutarkhani, F.H.; Mansouri, N. Task scheduling algorithms for energy optimization in cloud environment: A comprehensive review. *Clust. Comput.* **2022**, *25*, 1035–1093. [CrossRef]
6. Du, J.; Leung, J.Y.T. Complexity of Scheduling Parallel Task Systems. *SIAM J. Discret. Math.* **1989**, *2*, 473–487. [CrossRef]
7. Martí, R.; Reinelt, G. Heuristic Methods. In *Exact and Heuristic Methods in Combinatorial Optimization: A Study on the Linear Ordering and the Maximum Diversity Problem*; Springer: Berlin/Heidelberg, Germany, 2022; Chapter 2, pp. 27–57. [CrossRef]
8. Durasević, M.; Jakobović, D. Heuristic and metaheuristic methods for the parallel unrelated machines scheduling problem: A survey. *Artif. Intell. Rev.* **2022**, *56*, 3181–3289. [CrossRef]
9. Vinod Chandra, S.S.; Anand, H.S. Nature inspired meta heuristic algorithms for optimization problems. *Computing* **2022**, *104*, 251–269. [CrossRef]
10. Abualigah, L.; Elaziz, M.A.; Khasawneh, A.M.; Alshinwan, M.; Ibrahim, R.A.; Al-Qaness, M.A.A.; Mirjalili, S.; Sumari, P.; Gandomi, A.H. Meta-heuristic optimization algorithms for solving real-world mechanical engineering design problems: A comprehensive survey, applications, comparative analysis, and results. *Neural Comput. Appl.* **2022**, *34*, 4081–4110. [CrossRef]
11. Abo-Alsabeh, R.R.; Salhi, A. The Genetic Algorithm: A study survey. *Iraqi J. Sci.* **2022**, *63*, 1215–1231. [CrossRef]
12. Shami, T.M.; El-Saleh, A.A.; Alswaitti, M.; Al-Tashi, Q.; Summakieh, M.A.; Mirjalili, S. Particle Swarm Optimization: A Comprehensive Survey. *IEEE Access* **2022**, *10*, 10031–10061. [CrossRef]
13. Abohamama, A.S.; El-Ghamry, A.; Hamouda, E. Real-Time Task Scheduling Algorithm for IoT-Based Applications in the Cloud–Fog Environment. *J. Netw. Syst. Manag.* **2022**, *30*, 54. [CrossRef]
14. Pinedo, M.L. *Scheduling: Theory, Algorithms, and Systems*; Springer: Berlin/Heidelberg, Germany, 2016; Chapter 2, pp. 13–32.
15. lpsolve: Mixed Integer Linear Programming (MILP) Solver. 2021. Available online: https://sourceforge.net/projects/lpsolve/ (accessed on 20 March 2023).
16. MathWorks, I. Optimization Toolbox: Solve Linear, Quadratic, Conic, Integer, and Nonlinear Optimization Problems. 2022. Availabe online: https://ww2.mathworks.cn/en/products/optimization.html (accessed on 20 March 2023).
17. Jong, K.A.D.; Spears, W.M. A formal analysis of the role of multi-point crossover in genetic algorithms. *Ann. Math. Artif. Intell.* **1992**, *5*, 1–26. [CrossRef]
18. Nabi, S.; Ahmed, M. OG-RADL: Overall performance-based resource-aware dynamic load-balancer for deadline constrained Cloud tasks. *J. Supercomput.* **2021**, *77*, 7476–7508. [CrossRef]
19. Nabi, S.; Ahmed, M. PSO-RDAL: Particle swarm optimization-based resource- and deadline-aware dynamic load balancer for deadline constrained cloud tasks. *J. Supercomput.* **2022**, *78*, 4624–4654. [CrossRef]

20. Hussain, A.A.; Al-Turjman, F. Hybrid Genetic Algorithm for IOMT-Cloud Task Scheduling. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 6604286. [CrossRef]
21. Barroso, L.A.; Hölzle, U. The Case for Energy-Proportional Computing. *Computer* **2007**, *40*, 33–37. [CrossRef]
22. Baliga, J.; Ayre, R.W.A.; Hinton, K.; Tucker, R.S. Green Cloud Computing: Balancing Energy in Processing, Storage, and Transport. *Proc. IEEE* **2011**, *99*, 149–167. [CrossRef]
23. Tian, W.; He, M.; Guo, W.; Huang, W.; Shi, X.; Shang, M.; Toosi, A.N.; Buyya, R. On minimizing total energy consumption in the scheduling of virtual machine reservations. *J. Netw. Comput. Appl.* **2018**, *113*, 64–74. [CrossRef]
24. Aghdashi, A.; Mirtaheri, S.L. Novel dynamic load balancing algorithm for cloud-based big data analytics. *J. Supercomput.* **2022**, *78*, 4131–4156. [CrossRef]
25. Athmani, M.E.; Arbaoui, T.; Mimene, Y.; Yalaoui, F. Efficient Heuristics and Metaheuristics for the Unrelated Parallel Machine Scheduling Problem with Release Dates and Setup Times. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'22), New York, NY, USA, 9–13 July 2022; Association for Computing Machinery: New York, NY, USA, 2022; pp. 177–185. [CrossRef]
26. Pradhan, R.; Satapathy, S.C. Energy Aware Genetic Algorithm for Independent Task Scheduling in Heterogeneous Multi-Cloud Environment. *J. Sci. Ind. Res.* **2022**, *81*, 776–784. [CrossRef]
27. Teraiya, J.; Shah, A. Optimized scheduling algorithm for soft Real-Time System using particle swarm optimization technique. *Evol. Intell.* **2022**, *15*, 1935–1945. [CrossRef]
28. Hafsi, H.; Gharsellaoui, H.; Bouamama, S. Genetically-modified Multi-objective Particle Swarm Optimization approach for high-performance computing workflow scheduling. *Appl. Soft Comput.* **2022**, *122*, 108791. [CrossRef]
29. Nwogbaga, N.E.; Latip, R.; Affendey, L.S.; Rahiman, A.R.A. Attribute reduction based scheduling algorithm with enhanced hybrid genetic algorithm and particle swarm optimization for optimal device selection. *J. Cloud Comput.* **2022**, *11*, 15. [CrossRef]
30. Wang, B.; Wu, P.; Arefzaeh, M. A new method for task scheduling in fog-based medical healthcare systems using a hybrid nature-inspired algorithm. *Concurr. Comput. Pract. Exp.* **2022**, *34*, e7155. [CrossRef]
31. Wang, B.; Cheng, J.; Cao, J.; Wang, C.; Huang, W. Integer particle swarm optimization based task scheduling for device-edge-cloud cooperative computing to improve SLA satisfaction. *PeerJ Comput. Sci.* **2022**, *8*, e893. [CrossRef]
32. Wang, B.; Wang, C.; Huang, W.; Song, Y.; Qin, X. Security-aware task scheduling with deadline constraints on heterogeneous hybrid clouds. *J. Parallel Distrib. Comput.* **2021**, *153*, 15–28. [CrossRef]
33. Ma, Z.; Zhang, S.; Chen, Z.; Han, T.; Qian, Z.; Xiao, M.; Chen, N.; Wu, J.; Lu, S. Towards Revenue-Driven Multi-User Online Task Offloading in Edge Computing. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 1185–1198. [CrossRef]
34. Mangalampalli, S.; Swain, S.K.; Mangalampalli, V.K. Multi Objective Task Scheduling in Cloud Computing Using Cat Swarm Optimization Algorithm. *Arab. J. Sci. Eng.* **2022**, *47*, 1821–1830. [CrossRef]
35. Otair, M.; Alhmoud, A.; Jia, H.; Altalhi, M.; Hussein, A.M.; Abualigah, L. Optimized task scheduling in cloud computing using improved multi-verse optimizer. *Clust. Comput.* **2022**, *25*, 4221–4232. [CrossRef]
36. Chandrashekar, C.; Krishnadoss, P.; Kedalu Poornachary, V.; Ananthakrishnan, B.; Rangasamy, K. HWACOA Scheduler: Hybrid Weighted Ant Colony Optimization Algorithm for Task Scheduling in Cloud Computing. *Appl. Sci.* **2023**, *13*, 3433. [CrossRef]
37. 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. [CrossRef]
38. Sharma, O.; Rathee, G.; Kerrache, C.A.; Herrera-Tapia, J. Two-Stage Optimal Task Scheduling for Smart Home Environment Using Fog Computing Infrastructures. *Appl. Sci.* **2023**, *13*, 2939. [CrossRef]
39. Kumar, M.S.; Karri, G.R. EEOA: Cost and Energy Efficient Task Scheduling in a Cloud-Fog Framework. *Sensors* **2023**, *23*, 2445. [CrossRef]
40. Cheikh, S.; Walker, J.J. Solving Task Scheduling Problem in the Cloud Using a Hybrid Particle Swarm Optimization Approach. *Int. J. Appl. Metaheuristic Comput.* **2022**, *13*, 1–25. [CrossRef]
41. Chhabra, A.; Huang, K.; Bacanin, N.; Rashid, T.A. Optimizing bag-of-tasks scheduling on cloud data centers using hybrid swarm-intelligence meta-heuristic. *J. Supercomput.* **2022**, *78*, 9121–9183. [CrossRef]