

Article

A Hybrid Multi-Objective Bat Algorithm for Solving Cloud Computing Resource Scheduling Problems

Jianguo Zheng  and Yilin Wang * 

Glorious Sun School of Business and Management, Donghua University, Shanghai 200051, China; zjg@dhu.edu.cn

* Correspondence: yilinwang0319@163.com

Abstract: To improve the service quality of cloud computing, and aiming at the characteristics of resource scheduling optimization problems, this paper proposes a hybrid multi-objective bat algorithm. To prevent the algorithm from falling into a local minimum, the bat population is classified. The back-propagation algorithm based on the mean square error and the conjugate gradient method is used to increase the loudness in the search direction and the pulse emission rate. In addition, the random walk based on lévy flight is also used to improve the optimal solution, thereby improving the algorithm's global search capability. The simulation results prove that the multi-objective bat algorithm proposed in this paper is superior to the multi-objective ant colony optimization algorithm, genetic algorithm, particle swarm algorithm, and cuckoo search algorithm in terms of makespan, degree of imbalance, and throughput. The cost is also slightly better than the multi-objective ant colony optimization algorithm and the multi-objective genetic algorithm.

Keywords: multi-objective; bat algorithm; resource scheduling problem; cloud computing; meta-heuristic algorithms



Citation: Zheng, J.; Wang, Y. A Hybrid Multi-Objective Bat Algorithm for Solving Cloud Computing Resource Scheduling Problems. *Sustainability* **2021**, *13*, 7933. <https://doi.org/10.3390/su13147933>

Academic Editor: Hyunchul Ahn

Received: 15 June 2021

Accepted: 12 July 2021

Published: 15 July 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

As a new type of computing model, cloud computing has fundamentally changed the delivery method of computing services and the convenience of resources provided through the Internet. The cloud computing resource scheduling problem is considered as difficult as the non-deterministic polynomial optimization problem, that is, an NP-hard problem. With the continuous development and complexity of cloud computing, this problem has become more challenging. Moreover, the scheduling optimization problem has become an indispensable and very important academic topic in academia. Therefore, one of the urgent problems to be solved is how to make cloud resource scheduling achieve a balance between availability and low cost, that is, to meet the availability of methods and technologies at a low cost. This requires optimizing load balance, improving scheduling performance, and improving resource utilization, while also reducing costs and increasing cost-effectiveness, so as to save energy and achieve good and fast sustainable development.

In the cloud computing environment, the objective function of resource scheduling is more focused on the goals of cloud providers and cloud users. The optimal resource scheduling generally considers the following goals, including service availability, computing cost, load balancing, makespan, and throughput. Therefore, the optimization problem can be transformed into mapping candidate solutions to fitness measurement functions, $R^n \rightarrow R$, and obtaining the optimal solution $z \in R^n$ from all feasible solutions, thereby satisfying the function minimization problem.

$$f(x) < f(y), \forall (x, y) \in R^n \quad (1)$$

The main contributions of this paper are as follows. First, a novel multi-objective optimization bat algorithm is proposed to solve the cloud-computing resource scheduling problem. The bat's population is divided into two types, namely the search-type population

and the captive population. In addition, the loudness and pulse emission rate of the original bat algorithm are improved and optimized using the Back Propagation based on the mean square error and Conjugate Gradient Method. Aiming at the optimization of feasible solutions, this paper uses lévy flight to improve the step length of the algorithm to achieve Pareto optimization. Second, we extend the objective functions to makespan, imbalance, cost, and throughput, and develop a resource scheduling model to meet the needs of cloud users and improve the resource utilization of cloud providers. Finally, these performances are compared and analyzed, and cloudlets or tasks are assigned to the virtual machine with the shortest time to prove the effectiveness of the proposed optimization model. It also verifies the superiority of the proposed algorithm compared with other meta-heuristic algorithms.

The rest of the paper is organized as follows: Section 2 shows the related works which deal with existing objective function of resource scheduling and scheduling techniques in the cloud environment. Section 3 deals with the proposed model. Section 4 discusses the proposed algorithm. Section 5 deals with performance evaluation and Section 6 consists of the conclusion and future work.

2. Related Works

With the vigorous development of big data, cloud users are increasingly demanding cloud computing platforms. How to schedule resources according to demand has become an important research topic.

In terms of parameters for measuring scheduling effectiveness, domestic and foreign scholars mainly focus on issues such as resource optimization, energy consumption, makespan, cost, and load balancing. To achieve optimal resource scheduling and reduce the energy consumption of cloud computing systems, Hao et al. [1] proposed a resource scheduling algorithm based on the execution of tasks in the cloud system. Further, Prakash [2] proposed a new dependency and time-based scheduling algorithm. Due to the types of clients and services provided in the cloud computing system, scheduling issues and energy consumption issues are the most required challenges for the system. Pirozmand et al. [3] proposed a two-step hybrid method for finding search and time tasks, genetic algorithm, and genetic algorithm-based neurological scheduling. Also, in order to ensure budget constraints in cloud computing while minimizing the completion time of workflows, Yao et al. [4] proposed a scheduling algorithm based on task repetition to optimize the completion time of budget-constrained workflows in cloud platforms. And Sardaraz [5] proposed a multi-objective scheduling algorithm for scheduling scientific workflows in cloud computing, which is based on a genetic algorithm for completion time, monetary cost, and load balancing. While Chen [6] proposed a real-time scheduling algorithm based on makespan and cost. It can be seen that most scholars use factors such as makespan, degree of imbalance, and cost as the objective function of cloud resource scheduling problems.

Currently, meta-heuristic algorithms used for cloud computing scheduling problems mostly focus on simulated annealing (SA), genetic algorithm (GA), particle swarm optimization algorithm (PSO), artificial bee colony algorithm (ABC), and ant colony algorithm (ACO). To fill the gap between resource dynamics and on-demand consumer application requirements, Attiya [7] proposed an improved Harris hawks optimization algorithm based on simulated annealing, while Muthulakshmi et al. [8] proposed a hybrid ABC-SA algorithm. Further, Velliangiri et al. [9] proposed a hybrid electronic search based on a genetic algorithm to improve task scheduling behavior by considering parameters such as completion time, load balance, resource utilization, and multi-cloud cost. Yu et al. [10] proposed a particle position weight calculation method based on particle fitness value classification, and applied the algorithm to the task scheduling of cloud computing platforms, and found the task scheduling with the shortest time-consuming and the most balanced computing resource load in the cloud platform Program. To minimize the completion time and balance the load, Kruekaew [11] also uses the artificial bee colony algorithm and

the virtual machine with the largest job priority to schedule and manage cloud resources. Further, Sanaj [12] proposed a chaotic squirrel search algorithm to optimize multi-task scheduling in an infrastructure-as-a-service cloud environment. Hence, to solve the VM integration problem, Xiao [13] proposed a multi-objective method based on dual thresholds and ant colony system. At the same time, Vinu [14] proposed a hybrid queue ant colony-artificial bee colony optimization algorithm.

3. Problem Formulation

Like other scheduling problems, resource scheduling in cloud computing is suitable for a valuable cloud resource to meet the cloud provider's needs for cloud users. It is mostly used to balance the load and ensure equal distribution of resources based on demand. And it gives some priority according to the set rules to ensure that cloud computing can meet the requests of all cloud users. Equation (2) for the resource scheduling problem is as follows:

$$RS = \sum_{x=1}^{m,n} (R_x + S_x + \dots + N_x) \times T_x \rightarrow U_x^z \quad (2)$$

where, it assign m cloudlets/task $T = (T_1, T_2, \dots, T_m)$ to convert n available material resources into cloud data center $R = (R_1, R_2, \dots, R_n)$, $S = (S_1, S_2, \dots, S_n)$, even $N = (N_1, N_2, \dots, N_n)$. Here, the target fitness function is set as $F = (F_1, F_2, \dots, F_z)$, and cloud users are represented by $U = (U_1, U_2, \dots, U_n)$. Assuming that there is a set of cloudlets/task $T_i = (T_1, T_2, \dots, T_n)$ required by cloud users, the agent is responsible for assigning the cloudlets/tasks to the necessary virtual resources $V_j = (V_1, V_2, \dots, V_m)$ as Virtual resources with the shortest completion time. The expected completion time (ETC) is described as the expected time for all cloudlets/tasks to be executed on certain virtual resources obtained through the use of the ETC matrix. Its elements are expressed as $ETC(T_i, V_j)$, and (3) is as follows:

$$ETC(T_i, V_j) = \begin{bmatrix} T_1 V_1 & T_1 V_2 & \dots & T_1 V_m \\ T_2 V_1 & T_2 V_2 & \dots & T_2 V_m \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ T_n V_1 & T_n V_2 & \dots & T_n V_m \end{bmatrix} \quad (3)$$

However, in reality, cloud resources cannot be fully utilized due to poor scheduling allocation. Therefore, the problem of determining the minimum cost resource utilization and constraining the completion time is a multi-objective optimization problem. Table 1 illustrates the complexity of the situation. Assuming that cloud users need IaaS cloud resources, these resources have the cheapest cost and the least execution time, which is impossible for cloud providers to fully realize.

Table 1. Options for multi-objective resource scheduling problems.

Type	Cost (RMB)	Time (min)
A	100	570
B	90	700
C	80	600
D	70	850
E	60	820

Table 1 shows the options and decisions that take the least time and cost and means that the objective function can maximize the use of the least completion time and consume the least cost. But in real life, it is difficult to balance and perfect the time and cost. Comparing A and B, it can be found that A has a lower cost but takes longer, while B has

a higher cost but takes less time. Similarly, if A and C are compared, the time is too long or the cost is high, both are non-dominant. In the multi-objective scheduling problem, the main goal is to find a non-dominated curve or solution [15]. These parameters are based on calculation, network, and storage, including availability, bandwidth/speed, cost, degree of imbalance, execution time, memory, performance, priority, reliability, response time, SLA, temperature, throughput, time, and utilization [16].

Therefore, the following part takes makespan, the degree of imbalance, cost, throughput, and performance as the objective function of cloud computing optimal resource scheduling, and calculates the corresponding values by using formulas (4)–(8). The minimum makespan, the minimum throughput, cost, and the smooth and stable imbalance degree can show that the multi-objective bat algorithm in this paper is more efficient. The goal is to reduce the makespan of specific cloudlets/tasks on all VMs during resource scheduling and obtain the least makespan, throughput, cost, and the most stable degree of imbalance.

3.1. Makespan Model

The makespan is defined as the longest completion time used to determine the execution of the task. If the makespan of cloudlets or task exceeds the needs of cloud users, these requirements will not be completed on time [17]. Therefore, it is necessary to reduce the makespan of all task mappings on the VM defined in (4).

$$f(x) = \max \bigcup_{i=1}^m T_i, \quad \forall i \in N, \quad i = 1, 2, \dots, m \quad (4)$$

3.2. Degree of Imbalance Model

To check the unbalanced load on the VM, the degree of imbalance is introduced [15]. Its role is to describe the amount of load distribution among VMs regarding their execution capabilities. Therefore, it is necessary to consider a stable average degree of imbalance to obtain better resource scheduling performance. Equation (5) is as follows:

$$f(x) = \frac{\bigcup_{i=1}^m \max T_i - \bigcup_{i=1}^m \min T_i}{\text{avg} T_i} \quad (5)$$

$$\forall i \in N, \quad i = 1, \dots, m$$

3.3. Cost Model

As reducing the cost of cloud users, the cost is defined as the total amount of resource usage or resource utilization paid by cloud users to cloud providers. In this way, it also effectively enhances the profit and revenue of cloud providers [5,18]. Assume that the real-time time identified by the cloud provider and the feedback description of the VM constitute the cost of the VM, and the result fluctuates accordingly. Therefore, (6) represents the computational cost of completing a specific VM task.

$$f(x) = \sum_{i=1}^m \text{resource}_i (C_i \times T_i) \quad (6)$$

$$\forall i \in N, \quad i = 1, \dots, m$$

where C_i represents the resource cost per unit time and T_i represents the time of resource utilization.

3.4. Throughput Model

In cloud computing, throughput is defined as the total number of tasks that have been successfully executed. This is equivalent to the number of certain tasks completed in a certain period. Furthermore, it can be known that task scheduling in cloud computing requires minimum throughput [15]. Equation (7) expresses the throughput that defines the total number of tasks mapped on the VM.

$$f(x) = \sum_{i=1}^m T_i, \quad \forall i \in N, \quad i = 1, 2, \dots, m \quad (7)$$

3.5. Performance Improvement Rate

In the cloud computing resource scheduling problem, the definition of performance improvement rate (PIR) is used to estimate the performance improvement percentage of the i -th algorithm compared to the k -th algorithm [15]. Usually, it is expressed as a percentage (%).

$$PIR(\%) = \frac{\sum_k PM - \sum_i PM}{\sum_i PM} * 100 \quad (8)$$

4. Materials and Methods

Bat Algorithm is a new meta-heuristic swarm intelligence algorithm proposed by Yang in 2010 inspired by bats' search behavior [19]. This algorithm has unique advantages in solving multi-objective optimization and nonlinear global optimization problems. However, when looking for a feasible solution to a specific issue, the bat algorithm is prone to fall into a local optimum, leading to premature convergence. Under ideal conditions, the bat algorithm is improved mainly by adjusting the pulse frequency f , loudness A_i , and pulse emission rate r_i . Because solving the resource scheduling problem of cloud computing involves four indicators, it belongs to the multi-objective problem. To make it easily distinguished from the original bat algorithm and other meta-heuristic algorithms, the algorithm proposed in this paper is named the hybrid multi-objective bat algorithm.

4.1. The Idealized Rules of Original Bat Algorithm

Under ideal circumstances, a new type of swarm intelligence algorithm is designed to simulate the changes in the frequency, loudness, and pulse emission rate of the pulses emitted by bats when foraging.

When proposing the bat algorithm, Yang gave the following idealized rules for the echolocation characteristics of simulated bats:

- IR1: All bats use echolocation to perceive distance, and can also "know" the difference between prey or things and obstacles in a way that we do not;
- IR2: The bat flies at will at a fixed frequency at a position and at a speed, looking for food with different wavelengths and loudness. They can adjust the wavelength or frequency of the pulses they emit and adjust the rate of pulse emission according to their proximity to their targets;
- IR3: Although loudness is unpredictable in many ways, we assume that loudness decreases from a positive maximum value to a minimum constant value.

4.2. A Hybrid Bat Algorithm Based on Multi-Objective

4.2.1. Habitat Initialization

To solve the resource allocation problem, the bat algorithm generates initial candidate habitats in the form of a matrix of size $N_{pop} \times n$, where N_{pop} and n are the size and the number of the initial population, respectively. Assuming that the bat sets a frequency f in the solution, each frequency f in the solution represents a solution, and the frequency f is a virtual resource or virtual machine reserved for a mapping task in a cloud environment. For bat $i = 1, 2, \dots, n$, assuming that the position of the bat is x_i and the velocity is v_i . In the D-dimensional space, the position of the bat i at time is x_i^t , the speed is v_i^t , and there is

$$f_i = f_{min} + (f_{max} - f_{min})\beta \quad (9)$$

$$v_i^t = v_i^{t-1} + (x_i^t - x_*)f_i \quad (10)$$

$$x_i^t = x_i^{t-1} + v_i^t \quad (11)$$

4.2.2. Population Classification

The optimization process is divided into two stages. The first stage is that when the individual bat is in a better search position and is close to the optimal solution, its loudness and pulse emission rate reach the best state. This process is called the search phase, its population is called the search-type population. And the second stage is the population of bats in a disadvantaged search position, that is, the capture population, so two populations with different functions are obtained. After each iteration, by updating the search direction and step length, the loudness and pulse emission rate of the bat algorithm are improved to find the optimal solution. By adjusting the weights and deviations of the network, the difference between the average value and the standard deviation of the bat algorithm can be minimized.

Search-Type Population

The combination of Back Propagation (BP) algorithm based on Mean Square Error (MSE) and gradient descent method minimizes the mean square error [20]. The formula based on the mean square error measurement is shown in (12):

$$\text{MSE}(d, y) = \frac{1}{N} \sum_{n=1}^N (d(n) - y(n))^2 \quad (12)$$

where $d(n)$ represents the n -th element of the required signal, and $y(n)$ represents the n th actual output. In the training process, the weight vector in the $(t + 1)$ iteration is updated by (14). Where α is the learning rate or step length, W_t is the weight vector of the previous iteration and g_t is the gradient vector, which can be calculated by (13) and (14).

$$w_{t+1} = w_t - \alpha g_t \quad (13)$$

$$g_t = \frac{\partial e}{\partial w} \Big|_{w=w_t} = \left[\frac{\partial e}{\partial w_{11}} \cdots \frac{\partial e}{\partial w_{ij}} \cdots \frac{\partial e}{\partial w_{nn}} \right] \quad (14)$$

In (14), e is the MSE error output in the t -th step of the training process and $\partial e / \partial w$ is derived from the MSE error on each element of the w vector. The weight is expressed as $w_{t+1} = w_t - \alpha_t g_t$, α_t will be adjusted to an appropriate value to achieve better convergence. At this time, suppose the probability pops of the search-type population, its loudness and pulse emission rate are updated to:

$$A_i^{t+1} = \alpha_t \times A_i^t \quad (15)$$

$$r_i^{t+1} = r_i^0 (1 - e^{-w^t}) \quad (16)$$

Captive Population

As a branch of the gradient descent method, the Conjugate Gradient method (CG) is applied to nonlinear unconstrained optimization problems. This method has strong local and global convergence [21]. According to the cloud computing resource scheduling problem, the CG method uses (17) to generate a weight sequence:

$$w'_{t+1} = w'_t + \alpha'_t d \quad (17)$$

where α'_t is the result of the line search method, which can be an exact line search or an inaccurate line search, which is expressed as a step size here. In (17), d_t is in the descending direction, which is expressed as the search direction here, and its formula is shown in (18):

$$d_{t+1} = -g_{t+1} + \beta_t d_t \quad (18)$$

In (18), β_t is the conjugate parameter, g_{t+1} is the gradient of the objective function concerning the weight at step $t + 1$, and represents the direction of the last step, and the first step is $d_0 = -g_0$.

In the iterative process, the loudness A_i and the pulse emission rate r_i will also change. As the bat gets closer to its prey, its loudness will decrease, and the pulse emission rate will increase. At this time, the probability of the catching population is pop_h and $pop_s + pop_h = 1$. In summary, the loudness and pulse emission rate of the bat algorithm are updated to

$$A_i^{t+1} = \alpha'_t \times A_i^t \quad (19)$$

$$r_i^{t+1} = r_i^o (1 - e^{-w't}) \quad (20)$$

4.3. Bat Algorithm Based on Lévy Flight

In the resource scheduling optimization problem of cloud computing, feasible solutions have some unavoidable limitations, such as lower accuracy and slower speed. Moreover, the reference [22] confirms that for the situation where the target position of the “no priority direction” is unknown and the target cannot be supplemented, and lévy flight is the best search mode. To improve the solution, compared with the standard Gaussian distribution algorithm, the bat algorithm is combined with lévy flight. Here x_i^t is the current solution, x_i^{t+1} is the new solution, then

$$x_i^{t+1} = x_i^t + \text{round}(\alpha \otimes L(\lambda)) \quad (21)$$

where α represents the step vector, $\otimes$ represents the Hadamard multiplication operator, $\text{round}(o)$ represents the rounding operation, $L(\lambda) = [l_1(\lambda), \dots, l_t(\lambda)]^T$ is the length of the random step, and $l_i(\lambda)$ follows the lévy flight, $l_i(\lambda) \sim t^{-\lambda}$, $1 < \lambda \leq 3$. Here, λ is the parameter that affects the random step size.

4.4. The Steps of Multi-Objective Bat Algorithm

In the cloud computing resource scheduling problem, the individual position in the bat algorithm can be mapped to the solution of the problem, that is, the optimal path. The task can be decomposed into several subtasks according to the requirements of resource allocation, and each subtask can be determined by the candidate solution. Or execute the corresponding service selected centrally in the feasible solution. The optimal individual position can be mapped to the optimal solution of the optimization problem. The fitness value of the habitat can be mapped to the quality of the solution to the problem. The dimension of habitat can be mapped to the number of subtasks of the problem. In summary, the steps of the hybrid bat algorithm are as follows. The flow chart of the proposed algorithm is shown in Figure 1.

- Step 1: Determine the objective function
Set the bat population size (N_{pop}), maximum number of iterations (gen), frequency range ($[f \min, f \max]$), step vector (α) and stop condition (tol).
- Step 2: Initialize the habitat
In the feasible region, randomly generate the initial habitat ($habitat$), frequency (f_i), initial pulse emission rate (r_i) and loudness (A_i).
- Step 3: Calculate the fitness value
Calculate the fitness value of the objective function RS and update the optimal position (p_i) and the optimal global position (p_g).
- Step 4: Set the initial iteration loop
Set $t = 1$, $tt = 1$.
- Step 5: Update the positions of all individuals
If $t < GEN$, update the positions of all individuals, the current optimal position $p_{current}$ and the global optimal position p_{global} ; otherwise, go to step 8.
- Step 6: Update iteration loop

- Update $t = t + 1$, $tt = tt + 1$.
- Step 7: Calculate the new population fitness value
If $tt < tol$, return to step 5; otherwise, calculate the current optimal position of the individual and generate a new population, and sort according to the individual fitness value. The new population is divided into two parts. One part includes a better search-type population with a number of $pop_s * n$, and the other part is a search-type population with a number of $(1 - pop_s) * n$. According to Section 4.2, new and better individuals are generated through local search.
- Step 8: Calculate the fitness value of the new search-type population
Set $tt = 1$; otherwise, return to step 5.
- Step 9: Output the optimal solution.

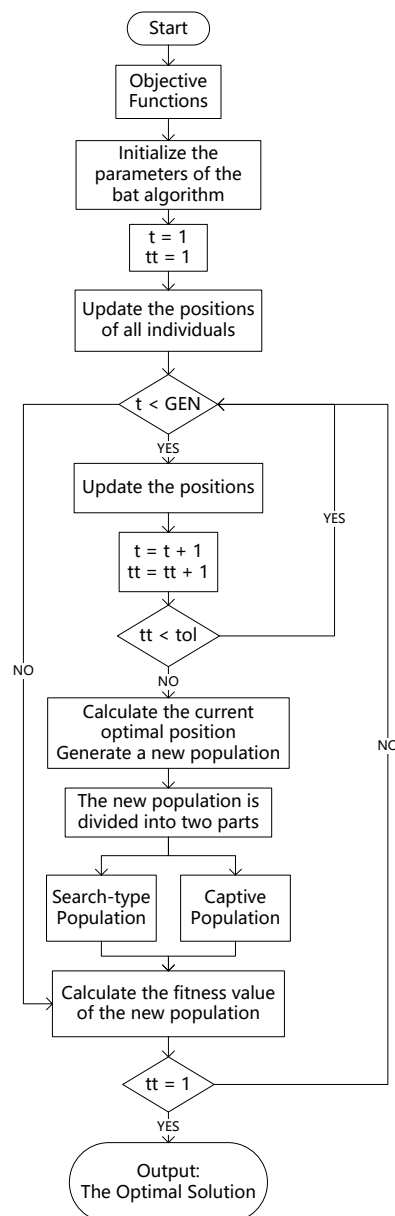


Figure 1. The flow chart of the MOBA algorithm.

5. Results and Discussion

This part describes the parameter settings, the implementation of the multi-objective bat algorithm(MOBA), and other meta-heuristic algorithms for optimizing IaaS cloud com-

puting resource scheduling. The test comes from the CUTer database [23]. The numerical experiments were performed using MATLAB r2020a software on a computer with 12.0 GB RAM, running professional Microsoft Windows 10. And creating a data set based on the uniform distribution and the normal distribution. The former includes an equal number of small, medium, and large tasks, and the latter includes fewer small, fewer large tasks, and more medium tasks, denoted by S01 and S02, respectively. In IaaS cloud computing, the MOBA algorithm is compared with other meta-heuristic algorithms, and a set of parameters is selected, including makespan, throughput, imbalance, cost, and performance improvement rate. In simulation analysis, after 50 simulation runs, the average value, standard deviation, and optimal value are obtained to compare performance indicators.

5.1. Parameter Setting

Based on the literature [24–27], Table 2 shows this problem's parameter settings.

The parameter settings of the metaheuristic algorithm are shown in Table 3. The parameter values of the ant colony optimization algorithm (MOACO) are based on [26,28]. The MOACO has strong robustness and the ability to search for better solutions in solving performance, and is easy to implement in parallel. However, it has the shortcomings of slow convergence speed, easy to fall into local optimum, and lack of initial pheromone. Moreover, the search time of the algorithms is too long, and it is prone to stagnation. The parameter values of the genetic algorithm (MOGA) are based on [29]. The Multi-objective genetic algorithms have good global search ability and can quickly search out all solutions in the solution space without falling into the trap of rapid decline of locally optimal solutions. By using its inherent parallelism, distributed calculations can be carried out conveniently and the solution speed can be accelerated. However, the local search ability of genetic algorithms is poor, which leads to the greater time consumption of pure genetic algorithms. The search efficiency is low in the later stage of evolution, and the problem of premature convergence is prone to occur. The parameter values of particle swarm optimization (MOPSO) are based on [30,31]. MOPSO has a very fast approach to the optimal solution, and can effectively optimize the parameters of the system. Its advantage lies in solving some optimization problems of continuous functions. However, it is prone to premature convergence, especially when dealing with complex multi-peak search problems, and the local optimization ability is poor. The Cuckoo Algorithm (MOCOS) are [32,33]. There are few parameters, the convergence speed is not sensitive to parameter changes, it is not easy to fall into the local optimum, and it is easy to couple with other algorithms. It also has good versatility and strong robustness, but slow convergence speed and lack of vitality.

Table 2. Simulation parameter settings in cloud computing for resource scheduling.

Number	Entities	Parameters	Values
1	User	Number of users	25
		Number of brokers	5
2	Cloudlet	Number of cloudlets	100–2000
		Length	60,000
		File size	400
3	Host	RAM	2048
		Storage	10,000
		Bandwidth	10,000

Table 2. *Cont.*

Number	Entities	Parameters	Values
4	VM	Number of VMs	25
		Type of policy	Time shared
		RAM	512
		Bandwidth	10,000
		MIPS	1000
		Size	10,000
		Operating system	Windows
		Number of CPUs	1 on each
5	Data center	Number of data centers	2

Table 3. In resource scheduling, the parameter setting of meta-heuristic algorithm.

Algorithms	Parameters	Values
MOACO	Number of ants	10
	Vaporization factor	0.4
	Pheromone tracking weight	0.3
	Heuristic information weight	1
	Pheromone updating constant	100
MOGA	Population size	1000
	Max iteration	1000
	Crossover rate	0.5
	Mutation rate	0.1
MOPSO	Population size	1000
	Max iteration	1000
	Crossover rate	0.5
	Mutation rate	0.1
	Particle size	100
	Self-recognition coefficients	2
	Uniform random number	[0, 1]
	Max iteration	1000
MOCSSO	Inertia weight	[0.4, 0.9]
	Population size	1000
	Abandon probability	0.25
	Max iteration	1000
MOBA	Step size	[0.1, 1]
	Population size	1000
	Max iteration	1000
	Frequency	[0, 1]
	Loudness	[1, 2]
	Pulse emission rate	[0, 1]
	Step vector	[0, 1]

5.2. Simulation Results

This part elaborates and discusses the simulation results of statistical analysis, and evaluates the performance of the proposed MOBA in cloud computing resource scheduling. Here, the *mean* means the average, *std* is the standard deviation and *best* is the optimal value.

5.2.1. Optimization Test Problems

Four standard functions are selected in Test Functions and Datasets as shown in Table 4. The algorithm proposed in this paper and the original bat algorithm (BA) are used to solve the four standard functions to test the optimization performance of the algorithm. The parameters are set as follows, the population size is 100, the maximum number of iterations is 50, the maximum frequency is 1, the minimum frequency is 0, the loudness is 0.9, and the pulse emission rate is 0.9. In addition, the step vector of MOBA takes a value of 0.9. According to the above parameter settings, each algorithm runs 50 times. Table 5 shows the optimization results of the two algorithms for the four standard functions.

Table 4. Four standard functions.

Function	Type	Dimension	Search Scope	The Optimal Value
Ackley	Local Minima	100	[−32.768, 32.768]	0
Griewank	Local Minima	100	[−600, 600]	0
Sphere	Bowl-Shaped	100	[−5.12, 5.12]	0
Rosenbrock	Valley-Shaped	100	[−5, 10]	0

Table 5. Optimization results of algorithms for four standard functions.

Function	Algorithm	The Optimal Value	Worst Value	Mean	std
Ackley	BA	5.81	7.12	6.19	0.99
	MOBA	5.25	6.64	5.97	1.23
Griewank	BA	1.03	1.05	1.05	0.0029
	MOBA	0.91	1.054	0.9537	0.095
Sphere	BA	126.74	213.16	161.32	45.19
	MOBA	120	270.23	157.96	86.56
Rosenbrock	BA	0.6×10^5	1.52×10^5	1.16×10^5	0.52×10^5
	MOBA	0.55×10^5	2.01×10^5	0.68×10^5	0.78×10^5

It can be seen from the table that compared with the original bat algorithm, the algorithm proposed in this paper has better optimal value, worst value, average value, and standard deviation for these four benchmark functions. The original bat algorithm only has a higher optimization result for the simple test function Griewank.

To fully illustrate the performance of the algorithm in this paper, Figure 2 shows the optimal convergence curves of the two algorithms for the four benchmark functions. It can be seen from the figure that the overall optimization performance of the original bat algorithm (BA) and the algorithm proposed in this paper (MOBA) are similar in the four benchmark functions, and the optimization effect on high-dimensional complex spaces is acceptable. However, in the optimization test of the multimodal function Ackley, as the number of iterations increases, the objective function value of the original bat algorithm decreases too slowly. Moreover, its optimization accuracy is not good, and the convergence speed is too slow.

The improved bat algorithm in this paper shows good performance in the entire range of iterations, regardless of the unimodal function Sphere, Rosenbrock, or the multimodal

function Ackley and Griewank functions. It can be clearly seen that the curve of the objective function value drops fast and the optimization accuracy is high. With the increase in the number of iterations, the optimal value can be obtained in the optimization.

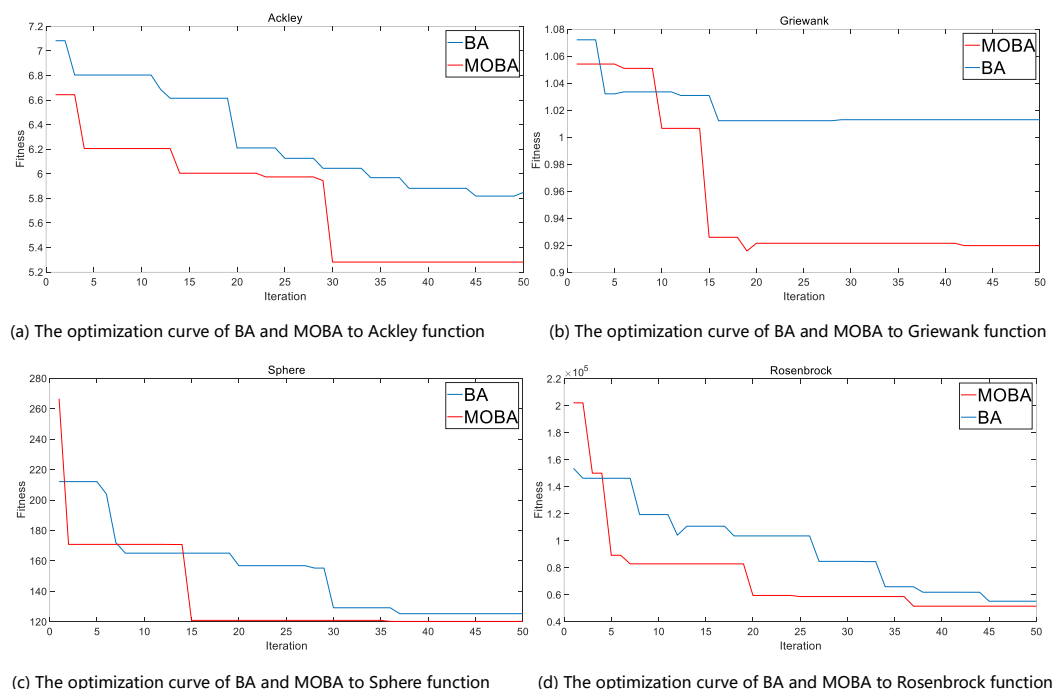


Figure 2. The optimization curve of the algorithms on the function.

5.2.2. Makespan

Table 6 shows that after running 50 times, when the number of the cloudlets is 100, 500, 1000, and 2000, the optimal value of the makespan of all algorithms is very close to the average value. It can also be clearly seen that as the number of the cloudlets increases, the standard deviation of MOBA becomes smaller and smaller, indicating that the larger the data scale, the greater the advantage of the proposed algorithm and the better its superiority. Compared with other algorithms, the standard deviation of MOBA is the smallest. When the cloudlet is 2000, its value is zero, indicating that the divergence and deviation of the data are the smallest, and the time consumed is the smallest.

Figure 3 shows that as in a uniform distributed environment, it can be seen that the green color represents different values of MOBA under different conditions of cloudlets' values, and its makespan is always lower than the other several meta-heuristic algorithms. Especially when the cloudlet is 1000, the value is much lower than MOGA represented by orange. And under the normal distribution, when the cloudlet's values are 100 and 2000, the degree of change of all algorithms is small. While the values are 500 and 1000, the makespan of MOGA is the longest, the makespan of MOBA proposed in this paper is the best. It shows that in these two distributions, MOBA has better effectiveness and superiority.

Table 6. After 50 runs, the makespan statistics table.

Algorithm	Cloudlets	S01			
		100	500	1000	2000
MOACO	<i>mean</i>	1562.73	2002.73	3213.22	5846.9
	<i>std</i>	4.18	21.64	33.81	21.23
	<i>best</i>	1558.9	1974.87	3153.32	5781.03
MOGA	<i>mean</i>	1533.18	3011.76	4768.28	5865.64
	<i>std</i>	7.57	149.28	8.84	24.9
	<i>best</i>	1518.29	2953.06	4755.65	5759.9
MOPSO	<i>mean</i>	1528.49	1993.4	2582.26	5828.15
	<i>std</i>	8.09	17.03	19.02	22.02
	<i>best</i>	1337.81	1973.25	2562.36	5761.02
MOCSSO	<i>mean</i>	1296.12	1721.55	2255.55	5687.5
	<i>std</i>	2.51	1.6	0.91	14.41
	<i>best</i>	1290.43	1719.07	2253.74	5641.2
MOBA	<i>mean</i>	1290.6	1490.94	1940.56	5324.02
	<i>std</i>	2.22	1.41	0.8	0
	<i>best</i>	1124.56	1488.74	1938.96	5632.02
Algorithm	Cloudlets	S02			
		100	500	1000	2000
MOACO	<i>mean</i>	719.02	954.25	1457.44	1349.2
	<i>std</i>	1.92	10.88	16.85	32.72
	<i>best</i>	717.26	941.73	1428.03	1307.8
MOGA	<i>mean</i>	718.26	1374.29	2174.43	1292.34
	<i>std</i>	3.53	67.96	4.05	31.35
	<i>best</i>	711.33	1339.27	2168.42	1239.34
MOPSO	<i>mean</i>	711.97	1036.11	1346.02	1406.06
	<i>std</i>	8.55	11.25	16.05	31.52
	<i>best</i>	624.55	1022.8	1327.28	1357.46
MOCSSO	<i>mean</i>	704.35	935.51	1219.35	1069
	<i>std</i>	1.36	0.87	0.49	17.15
	<i>best</i>	701.27	934.17	1218.37	1013.9
MOBA	<i>mean</i>	565.08	745.84	970.76	1004
	<i>std</i>	1.11	0.7	0.4	0
	<i>best</i>	562.56	744.74	969.96	1004

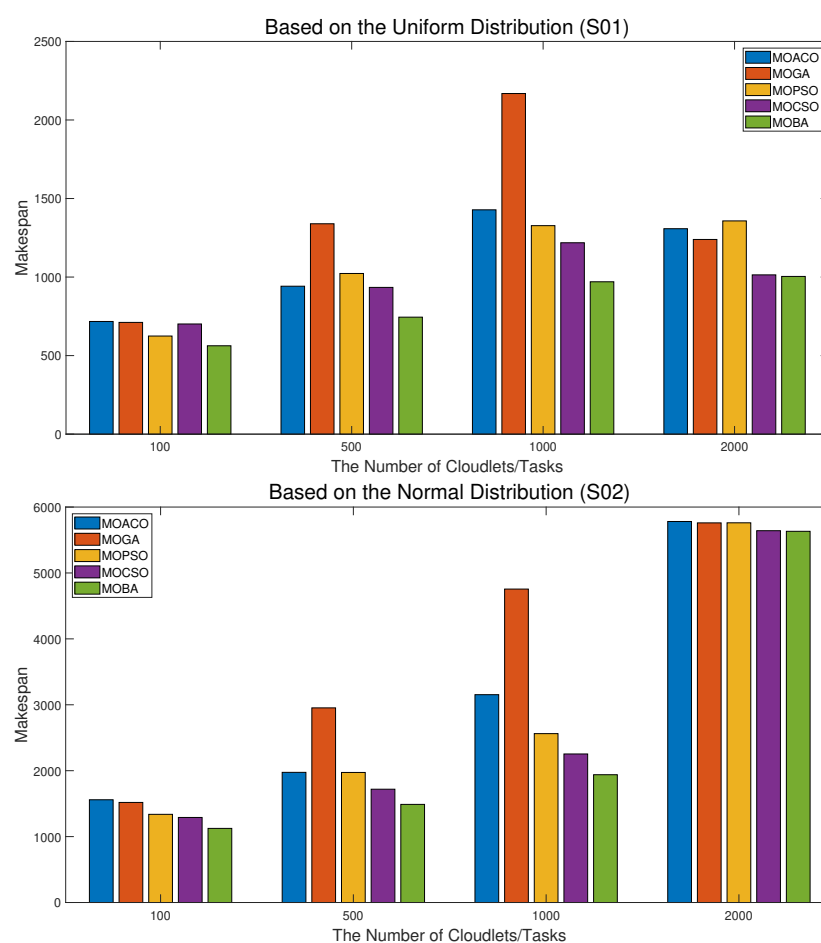


Figure 3. In the case of uniform distribution and normal distribution, the minimum makespan of each algorithm.

5.2.3. Degree of Imbalance

Table 7 shows that after running 50 times, in the case of uniform distribution and normal distribution, when the cloudlet is 1000, the standard deviation of all algorithms reaches the maximum value. At the same time, it can also be seen that the average and optimal value of MOBA's imbalance degree are almost the same, and the standard deviation is almost the same. Compared with other algorithms, the standard deviation of MOBA still reaches the minimum which shows that the MOBA algorithm has significant characteristics and is more superior to other meta-heuristic algorithms when considering the degree of imbalance.

Figure 4 shows that in the case of uniform distribution, similar to Table 5, the value of its cloudlet fluctuates very little, which also reflects the performance of examining the degree of imbalance. The change under uniform distribution is not as strong as the normal distribution. In a normal distribution environment, it can be analyzed that MOPSO has the worst imbalance. Moreover, the degree of imbalance of MOBA is the best, and the results of MOCSO are comparable to those of MOBA.

Table 7. Statistics of the degree of imbalance after running 50 times.

Algorithm	Cloudlets	S01			
		100	500	1000	2000
MOACO	<i>mean</i>	0.5026	0.5529	0.5552	0.5541
	<i>std</i>	0.0726	0.0828	0.0878	0.0854
	<i>best</i>	0.5092	0.5404	0.5192	0.5289
MOGA	<i>mean</i>	0.5006	0.5229	0.5334	0.5282
	<i>std</i>	0.0723	0.0755	0.077	0.0763
	<i>best</i>	0.505	0.5308	0.5428	0.5369
MOPSO	<i>mean</i>	0.5132	0.5511	0.5682	0.5597
	<i>std</i>	0.0763	0.0811	0.0873	0.0853
	<i>best</i>	0.5128	0.5447	0.5405	0.5437
MOCSSO	<i>mean</i>	0.4944	0.5048	0.5144	0.5107
	<i>std</i>	0.0714	0.0729	0.0742	0.0736
	<i>best</i>	0.4998	0.5135	0.5237	0.5197
MOBA	<i>mean</i>	0.4942	0.5048	0.5147	0.5096
	<i>std</i>	0.0714	0.0728	0.0743	0.0734
	<i>best</i>	0.4987	0.4987	0.4987	0.5004
Algorithm	Cloudlets	S02			
		100	500	1000	2000
MOACO	<i>mean</i>	0.5023	0.5593	0.5628	0.5611
	<i>std</i>	0.0725	0.0837	0.0887	0.0873
	<i>best</i>	0.5089	0.5471	0.5272	0.5373
MOGA	<i>mean</i>	0.498	0.5081	0.5186	0.5136
	<i>std</i>	0.072	0.0734	0.0749	0.0742
	<i>best</i>	0.5023	0.5157	0.5277	0.5217
MOPSO	<i>mean</i>	0.5076	0.5891	0.6056	0.5974
	<i>std</i>	0.0771	0.0864	0.092	0.0892
	<i>best</i>	0.5044	0.5839	0.5798	0.5819
MOCSSO	<i>mean</i>	0.4935	0.5037	0.5109	0.5084
	<i>std</i>	0.0713	0.0727	0.0737	0.0743
	<i>best</i>	0.4996	0.5124	0.5201	0.5163
MOBA	<i>mean</i>	0.4942	0.5048	0.5147	0.5098
	<i>std</i>	0.0714	0.0727	0.0743	0.0746
	<i>best</i>	0.4897	0.4897	0.5143	0.5002

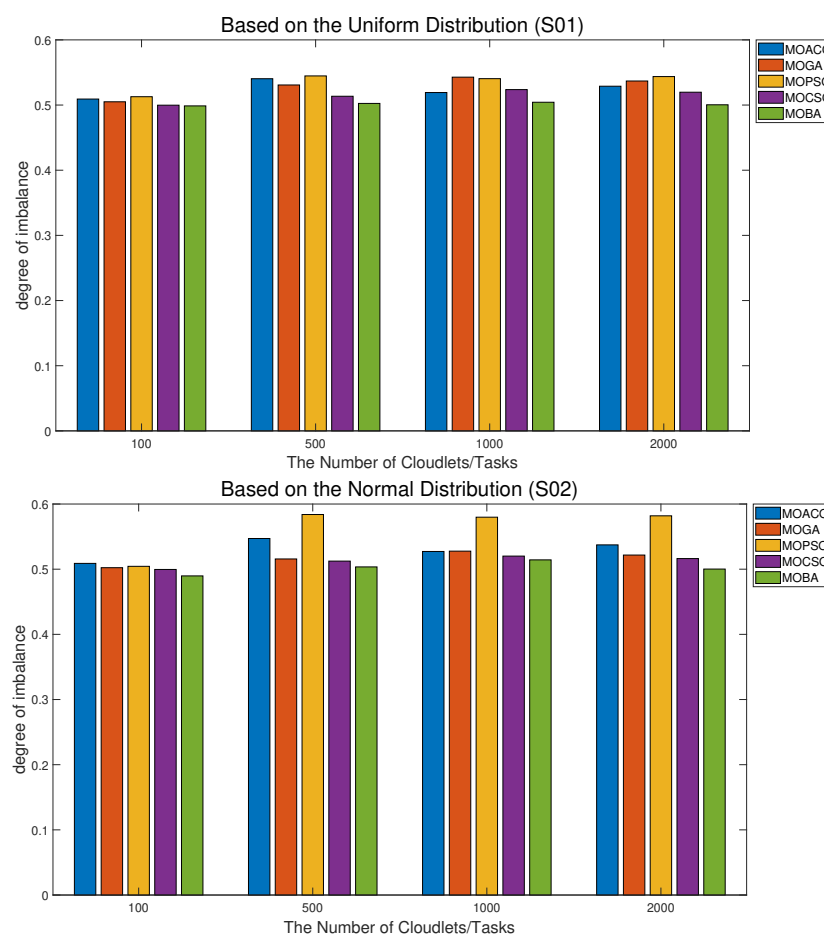


Figure 4. In the case of uniform distribution and normal distribution, the degree of imbalance of each algorithm.

5.2.4. Cost

As it is a cost calculation, the smaller the result, the better. It can be observed from Table 8 that in uniform distribution and normal distribution, as the number and scale of the cloudlets increase, the cost of all algorithms also increases. It can be seen from the standard deviation that MOBA is generally smaller than the standard deviation of MOACO, MOGA, and MOPSO, and larger than the standard deviation of MOCSSO. It can also be obtained that the cost of MOACO is the highest, while the cost of MOCSSO is the lowest.

It can be clearly seen from Figure 5 that regardless of the uniform distribution or the normal distribution, MOBA is better than MOACO and MOGA, and worse than MOPSO and MOCSSO. We can also see that as the number of the cloudlets changes, MOACO fluctuates the most, indicating that its cost investment is the largest, and the changes and fluctuations are large. It means that the consumption in terms of energy-saving is also the largest, and it is not conducive to the improvement of resource utilization.

Table 8. After running 50 times, the cost statistics table.

Algorithm	Cloudlets	S01			
		100	500	1000	2000
MOACO	<i>mean</i>	99.66	157.16	194.66	271.01
	<i>std</i>	2.02	2.83	3.4	7.8
	<i>best</i>	95.3	150.53	185.31	249.65
MOGA	<i>mean</i>	42.76	69.54	76.53	118.9
	<i>std</i>	0.98	1.73	1.76	4.08
	<i>best</i>	40.47	65.56	70.66	107.03
MOPSO	<i>mean</i>	28.53	54.98	61.25	95.71
	<i>std</i>	0.94	1.554	1.95	3.28
	<i>best</i>	27.44	43.15	55.19	85.15
MOCSSO	<i>mean</i>	22.51	34.09	40.47	62.55
	<i>std</i>	0.38	0.8	1.02	2.6
	<i>best</i>	21.43	25.42	27.61	54.68
MOBA	<i>mean</i>	35.54	59.38	67.87	103.55
	<i>std</i>	0.84	1.45	1.69	3.6
	<i>best</i>	33.87	51.32	59.54	92.75
Algorithm	Cloudlets	S02			
		100	500	1000	2000
MOACO	<i>mean</i>	1.55	35.1	66.47	132.88
	<i>std</i>	0.24	4.89	9.73	14.93
	<i>best</i>	1.13	28.98	54.65	101.7
MOGA	<i>mean</i>	0.55	12.09	23.59	47.09
	<i>std</i>	0.3	2.88	5.47	8.91
	<i>best</i>	0.29	9.36	18.97	36.39
MOPSO	<i>mean</i>	0.22	9.09	17.96	35.97
	<i>std</i>	4.26	4.27	4.28	7.61
	<i>best</i>	0.32	6.61	12.89	29.06
MOCSSO	<i>mean</i>	0.31	5.16	10.02	26.97
	<i>std</i>	0.14	0.62	1.1	3.1
	<i>best</i>	0.18	4.96	9.74	23.6
MOBA	<i>mean</i>	0.43	10.43	20.27	42.69
	<i>std</i>	1.48	2.73	4	7.06
	<i>best</i>	0.32	8.35	16.42	34.18

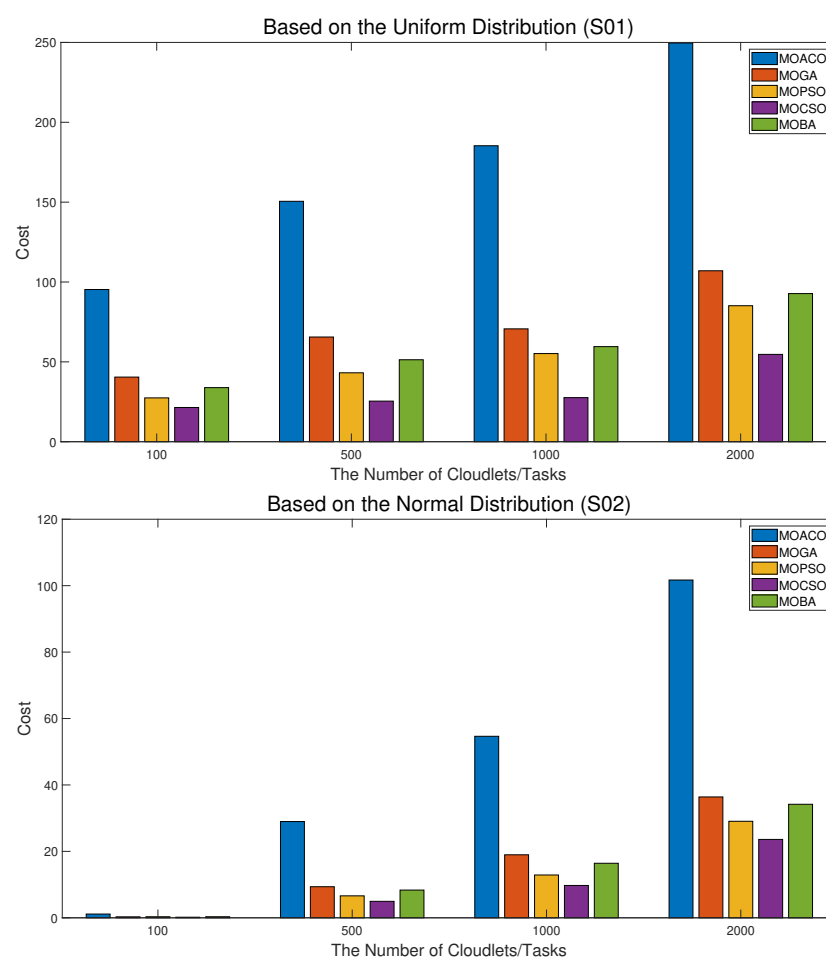


Figure 5. In the case of uniform distribution and normal distribution, the minimum cost of each algorithm.

5.2.5. Throughput

For non-concurrent application systems, throughput is strictly inversely proportional to response time. Therefore, in a concurrent system, throughput is usually used as a performance indicator. It shows that the resource allocation is reasonable, and the throughput seen by each user does not increase linearly with the increase of the cloudlets. Table 9 shows the mean, standard deviation, and optimal of the throughput. After 50 runs, the number of cloudlets is 100, 500, 1000, and 2000. It can be seen from Table 9 that in the throughput analysis of the MOBA algorithm, the standard deviation value is not as prominent as the makespan and cost data. However, compared with the other four algorithms, the standard deviation of MOBA has a significant advantage. In particular, when the cloudlet is 100 and 200, the value of MOBA is one-third of MOACO.

It can be analyzed from Figure 6 that in the case of uniform distribution, as the number of the cloudlets increases, the throughput also increases. This shows that throughput will affect the efficiency and utilization of resource scheduling, thereby further affecting the sustainable development of cloud computing and the issues of energy conservation and emission reduction. In addition, we can see from the figure that the throughput of MOBA has always been close to MOCSSO in a uniform distribution, but has been roughly smaller than MOCSSO in a normal distribution. In the large-scale case of the cloudlets of 1000 and 2000, the throughput of MOBA is much smaller than the three meta-heuristic algorithms of MOACO, MOGA, and MOPSO. This also verifies the conclusions in Table 9. It has to be said that in terms of throughput, MOBA still has strong advantages and good performance.

Table 9. After 50 runs, the throughput statistics table.

Algorithm	Cloudlets	S01			
		100	500	1000	2000
MOACO	<i>mean</i>	6371.82	9847.10	13,957.38	15,894.51
	<i>std</i>	64.81	35.06	144.45	144.53
	<i>best</i>	5793.41	9798.30	13,685.52	15,680.74
MOGA	<i>mean</i>	5184.74	12,163	20,442.70	21,761.99
	<i>std</i>	69.34	904.29	202.46	762.69
	<i>best</i>	5132.40	11,270.98	20,243.09	21,034.24
MOPSO	<i>mean</i>	5277.80	16,289.42	22,394.57	25,814.19
	<i>std</i>	72.07	162.10	119.43	212.55
	<i>best</i>	4120.10	15,968.72	22,231.03	25,491.36
MOCSSO	<i>mean</i>	4565.69	5490.20	7428.87	8637.573
	<i>std</i>	33.25	125.61	265.67	285.71
	<i>best</i>	4482.67	5247.46	6991.34	8184.06
MOBA	<i>mean</i>	4029.61	4912.45	6141.06	7393.87
	<i>std</i>	29.49	111.40	235.63	256.21
	<i>best</i>	3955.97	4912.45	6141.06	7393.87
Algorithm	Cloudlets	S02			
		100	500	1000	2000
MOACO	<i>mean</i>	2830.04	7296.57	9938.45	11,517.43
	<i>std</i>	87.72	125.98	202.85	246.64
	<i>best</i>	2573.14	7260.42	9744.86	11,364.27
MOGA	<i>mean</i>	2300.5	5396.79	9070.54	9672.307
	<i>std</i>	30.77	401.24	389.83	554.80
	<i>best</i>	2277.28	5001	8981.97	9349.40
MOPSO	<i>mean</i>	2431.91	7249.37	9966.38	11,504.59
	<i>std</i>	28.67	72.14	253.15	244.28
	<i>best</i>	1845.91	7106.65	9893.60	11,360.92
MOCSSO	<i>mean</i>	2481.16	18.04	2436.12	1663.53
	<i>std</i>	2983.80	68.34	2851.83	1974.20
	<i>best</i>	4037.46	144.61	3799.56	2656.87
MOBA	<i>mean</i>	2015.81	2457.45	3072.06	3713.76
	<i>std</i>	14.75	55.73	117.87	143.15
	<i>best</i>	1978.97	2457.45	3072.06	3713.76

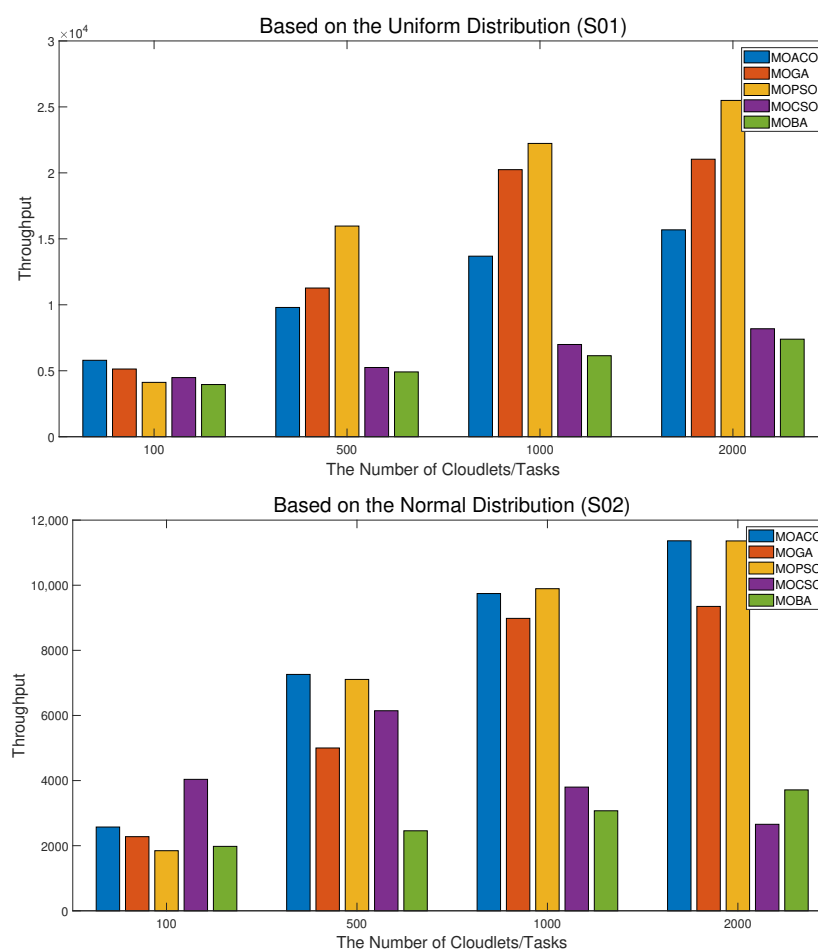


Figure 6. In the case of uniform distribution and normal distribution, the maximum throughput of each algorithm.

5.3. Performance Comparison

In summary, firstly, after 50 runs, the average value, standard deviation, and optimal value of MOBA are all very significant in terms of makespan, degree of imbalance, cost, and throughput. The saliency analysis proves that the MOBA algorithm is more robust and has a stronger Pareto optimal solution ability. Secondly, the MOBA algorithm has good robustness under the conditions of uniform distribution and normal distribution, and the ability to reach close to the optimal value in almost all operations. Therefore, this paper discusses the calculation results of simulations formulated using statistical significance analysis to evaluate the performance of the proposed MOBA algorithm for multi-objective resource scheduling in an IaaS cloud computing environment.

Tables 10–13 show that the relative performance improvement rate of MOBA algorithm based on makespan, the imbalance degree, cost and throughput compared with MOACO, MOGA, MOCSO, and MOPSO algorithm. The conclusions are as follows:

- In terms of total makespan, the value of MOBA is 13,465.54, which is the smallest among these algorithms. It can be seen from Table 10 that the performance comparison analysis with MOACO, MOGA, MOPSO, and MOCSO shows that the results are 20.15%, 34.14%, 15.66%, and 8.85%, respectively. Moreover, the performance comparison result between MOGA and MOACO is −21.24%, which also verifies the conclusion of Figure 3. The conclusion is that MOGA has the longest makespan and the worst performance. Furthermore, this also fully shows that MOBA has strong advantages and superiority in solving the problem of makespan. In terms of makespan, the MOBA algorithm was compared and analyzed with the MOACO,

MOGA, MOPSO, and MOCSO algorithms, and the results obtained were shortened by 10.07%, 25.82%, 5.02% and -2.66% , respectively.

- For the degree of imbalance, the values of these algorithms do not differ greatly. It can be seen from Table 11 that the total imbalance is in the interval of $[4, 5]$, and the result of MOBA is 4.01%, which is the smallest among these meta-heuristic algorithms. Even so, it can be accurately seen that the comparison results of MOBA with other meta-heuristic algorithms are 4.85%, 4.05%, 8.61%, and 2.23%, and the comparison results of MOPSO with MOACO and MOGA are -4.11% and -4.99% , respectively. From this, it can be clearly affirmed that the performance of MOBA in the degree of imbalance is the most stable, and the most unstable algorithm is MOPSO. This also verifies the conclusion of Figure 4.
- In terms of cost, it can be clearly seen that the total cost of MOBA is 296.75, which is much lower than MOCAO and MOGA, but higher than MOPSO and MOCSO. Moreover, in the performance comparison, it can also be seen that the results of comparison between MOBA and other meta-heuristic algorithms in terms of cost are 65.78%, 14.91%, -14.22% and -77.04% , respectively. This also coincides with the conclusion of Figure 5. This proves that MOBA is unsatisfactory in terms of cost and is not very prominent.
- As we all know, throughput is related to the rationality of resource allocation and resource utilization. Overall, the throughput result of MOBA is 33,625.59, compared with other meta-heuristic algorithms, the results are 55.70, 59.63, 65.69, 19.06, respectively. In addition, it can also be analyzed to compare MOGA with MOACO, and compare MOPSO with MOACO and MOGA, and the results are all negative. This shows that in terms of throughput comparison, MOGA and MOPSO are both unsatisfactory results, especially MOPSO is the worst. In comparison, MOBA is the best performance.

Table 10. Comparison of performance in terms of Makespan.

Algorithm	MOACO	MOGA	MOPSO	MOCSO	MOBA
Total Makespan	16,862.94	20,445.26	15,966.53	14,772.15	13,465.54
PIR over MOACO		-21.24	5.32	12.40	20.15
PIR over MOGA			21.91	27.75	34.14
PIR over MOPSO				7.48	15.66
PIR over MOCSO					8.85

Table 11. Comparison of performance in terms of DI.

Algorithm	MOACO	MOGA	MOPSO	MOCSO	MOBA
Degree of imbalance	4.22	4.18	4.39	4.11	4.01
PIR over MOACO		0.84	-4.11	2.68	4.85
PIR over MOGA			-4.99	1.86	4.05
PIR over MOPSO				6.53	8.61
PIR over MOCSO					2.23

Table 12. Comparison of performance in terms of Cost.

Algorithm	MOACO	MOGA	MOPSO	MOCSSO	MOBA
Total Cost	867.25	348.73	259.81	167.62	296.75
PIR over MOACO		59.79	70.04	80.67	65.78
PIR over MOGA			25.50	51.93	14.91
PIR over MOPSO				35.48	−14.22
PIR over MOCSSO					−77.04

Table 13. Comparison of performance in terms of Total Throughput.

Algorithm	MOACO	MOGA	MOPSO	MOCSSO	MOBA
Total Throughput	75,900.66	83,290.36	98,018.29	41,544.03	33,625.59
PIR over MOACO		−9.74	−29.14	45.27	55.70
PIR over MOGA			−17.68	50.12	59.63
PIR over MOPSO				57.62	65.69
PIR over MOCSSO					19.06

It can be seen that the MOBA algorithm accelerates the convergence speed and enhances its ability to handle optimization of resource scheduling problems in IaaS cloud computing, which also proves the effectiveness and robustness of the MOBA algorithm.

6. Conclusions

Resource scheduling is an NP-hard problem in the cloud computing environment. This paper outlines the problem of resource scheduling, proposes the MOBA algorithm, and discusses and analyzes its convergence and performance. It also compares and analyzes the performance of other meta-heuristic algorithms, including MOACO, MOGA, MOPSO, and MOCSSO algorithms for solving and optimizing resource scheduling problems in cloud computing. The research motivation is to discuss the problem of multi-objective resource scheduling by maximizing the utilization of resources in the cloud computing environment. Therefore, the MOBA proposed in this paper effectively utilizes resources through makespan, degree of imbalance, cost, and throughput. The simulation results are obtained with the help of graphs and statistical analysis, indicating that the proposed MOBA algorithm is more effective and has better performance than the existing meta-heuristic algorithm in optimizing resource scheduling in the cloud computing environment.

(1) Through the classification of MOBA populations, the search range of the algorithm is expanded, and the ability to solve feasible solutions is increased. Therefore, the search for better target areas can be achieved more effectively, and the completion time of the cloudlet in cloud computing can be reduced. It can also maintain the load balance of the nodes.

(2) The loudness and pulse emission rate of the original bat algorithm were improved and optimized using the back-propagation based on the mean square error and conjugate gradient methods. MOBA effectively overcomes the phenomenon of local optima and premature convergence, improves the solution accuracy and convergence speed, makes the resource utilization of cloud computing more balanced and reasonable, and achieves a good and fast sustainable development effect.

(3) Through the use of Lévy flight to optimize and improve the feasible solution, the Pareto optimization of the bat algorithm is realized, which effectively reduces the cost and energy consumption required for cloud computing, and greatly reduces the cost of the cloudlet. The degree of balance makes it more stable and balanced, and the resource scheduling ability of cloud computing has been greatly improved, which is good at optimizing resources and rationally scheduling resources, thereby improving the efficiency of resource allocation.

It is worth mentioning that this algorithm can also be used to solve other optimization problems in cloud computing environments and other discrete optimization problems in different fields. Therefore, in subsequent research, we will first pay more attention to solving other optimization problems of cloud computing, such as task scheduling or green cloud computing environment. Secondly, we will also combine the bat algorithm with other heuristic algorithms, and it may also prove to be a very effective optimization algorithm in the cloud computing environment. In the future research process, we will pay more attention to the research direction of environmental protection and sustainable economic development, such as the dual-carbon economy and green management.

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

Funding: This research was funded by the Fundamental Research Funds for the Central Universities under Grant No. CUSF-DH-D-2018050 (18D310804).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data of benchmark functions can be found here: <http://www.sfu.ca/~ssurjano/optimization.html>, accessed on 12 July 2021.

Acknowledgments: We gratefully acknowledge the anonymous reviewers for their insightful comments on the manuscript.

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

References

1. Hao, L.; Li, B.; Li, K.; Jin, Y. Research for Energy Optimized Resource Scheduling Algorithm in Cloud Computing Base on Task Endurance Value. In Proceedings of the 2019 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), Dalian, China, 29–31 March 2019.
2. Prakash, V.; Bawa, S.; Garg, L. Multi-Dependency and Time Based Resource Scheduling Algorithm for Scientific Applications in Cloud Computing. *Electronics* **2021**, *10*, 1320. [CrossRef]
3. Pirozmand, P.; Hosseinabadi, A.A.R.; Farrokhzad, M.; Sadeghilalimi, M.; Mirkamali, S.; Slowik, A. Multi-objective hybrid genetic algorithm for task scheduling problem in cloud computing. *Neural Comput. Appl.* **2021**, 1–14. [CrossRef]
4. Yao, F.; Pu, C.; Zhang, Z. Task Duplication-Based Scheduling Algorithm for Budget-Constrained Workflows in Cloud Computing. *IEEE Access* **2021**, *9*, 37262–37272. [CrossRef]
5. Sardaraz, M.; Tahir, M. A parallel multi-objective genetic algorithm for scheduling scientific workflows in cloud computing. *Int. J. Distrib. Sens. Netw.* **2020**, *16*, 1–13. [CrossRef]
6. Chen, H.; Wen, J.; Pedrycz, W.; Wu, G. Big Data Processing Workflows Oriented Real-Time Scheduling Algorithm using Task-Duplication in Geo-Distributed Clouds. *IEEE Trans. Big Data* **2020**, *6*, 131–144. [CrossRef]
7. Attiya, I.; Abd Elaziz, M.; Xiong, S.W. Job Scheduling in Cloud Computing Using a Modified Harris Hawks Optimization and Simulated Annealing Algorithm. *Eng. Sci. Technol. Int. J. JESTECH* **2020**, *2020*, 1–17. [CrossRef] [PubMed]
8. Muthulakshmi, B.; Somasundaram, K. A hybrid ABC-SA based optimized scheduling and resource allocation for cloud environment. *Clust. Comput.* **2019**, *22*, 10769–10777. [CrossRef]
9. Velliangiri, S.; Karthikeyan, P.; Xavier, V.M.A.; Baswaraj, D. Hybrid electro search with genetic algorithm for task scheduling in cloud computing. *Ain Shams Eng. J.* **2021**, *12*, 631–639. [CrossRef]
10. Yu, G.L.; Zhao, Y.; Cui, Z.W.; Yu, Z. A QPSO Algorithm Based on Hierarchical Weight and Its Application in Cloud Computing Task Scheduling. *Computer Sci. Inf. Syst.* **2021**, *18*, 189–212. [CrossRef]
11. Kruekaew, B.; Kimpan, W. Enhancing of Artificial Bee Colony Algorithm for Virtual Machine Scheduling and Load Balancing Problem in Cloud Computing. *Int. J. Comput. Intell. Syst.* **2020**, *13*, 496–510. [CrossRef]
12. 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. JESTECH* **2020**, *23*, 891–902. [CrossRef]
13. Xiao, H.; Hu, Z.; Li, K. Multi-Objective VM Consolidation Based on Thresholds and Ant Colony System in Cloud Computing. *IEEE Access* **2019**, *7*, 53441–53453. [CrossRef]
14. Vinu, S. Optimal Task Assignment in Mobile Cloud Computing by Queue Based Ant-Bee Algorithm. *Wirel. Pers. Commun.* **2019**, *104*, 173–197.

15. Madni, S.H.H.; Latiff, M.S.A.; Coulibaly, Y.; Abdulhamid, S.M. Recent advancements in resource allocation techniques for cloud computing environment: A systematic review. *Clust. Comput.* **2017**, *20*, 2489–2533. [\[CrossRef\]](#)
16. Bansal, M.; Sanjay Kumar Malik, S.K. A multi-faceted optimization scheduling framework based on the particle swarm optimization algorithm in cloud computing. *Sustain. Comput. Inform. Syst.* **2020**, *28*, 100429. [\[CrossRef\]](#)
17. Madni, S.; Latiff, M.; Abdulhamid, S. Optimal resource scheduling for IaaS cloud computing using cuckoo search algorithm. *Sains Humanika* **2017**, *9*, 71–76. [\[CrossRef\]](#)
18. Hosseini, L.; Tang, S.J.; Mookerjee, V.; Sriskandarajah, C. A Switch in Time Saves the Dime: A Model to Reduce Rental Cost in Cloud Computing. *Inf. Syst. Res.* **2020**, *31*, 753–775. [\[CrossRef\]](#)
19. Yang, X. A New Metaheuristic Bat-Inspired Algorithm. In *Studies in Computational Intelligence*; González, J.R., Pelta, D.A., Cruz, C., Terrazas, G., Krasnogor, N., Eds.; Springer: Berlin, Germany, 2010; pp. 65–74.
20. Luo, J.; Ren, R.; Guo, K. The deformation monitoring of foundation pit by back propagation neural network and genetic algorithm and its application in geotechnical engineering. *PLoS ONE* **2020**, *15*, e0233398. [\[CrossRef\]](#)
21. Cools, S.; Cornelis, J.; Vanroose, W. Numerically Stable Recurrence Relations for the Communication Hiding Pipelined Conjugate Gradient Method. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 2507–2522. [\[CrossRef\]](#)
22. Lu, Y.; Sun, Y.; Liu, X.; Gao, B. Control allocation for a class of morphing aircraft with integer constraints based on Lévy flight. *J. Syst. Eng. Electron.* **2020**, *31*, 826–840.
23. Gould, N.; Orban, D.; Toint, P. CUTER and SifDec: A constrained and unconstrained testing environment, revisited. *Acm Trans. Math. Softw.* **2003**, *29*, 373–394. [\[CrossRef\]](#)
24. Abdullahi, M.; Ngadi, M. Hybrid symbiotic organisms search optimization algorithm for scheduling of tasks on cloud computing environment. *PLoS ONE* **2016**, *11*, e0158229.
25. Abdulhamid, S.; Latiff, M.; Madni, S.; Abdullahi, M. Fault tolerance aware scheduling technique for cloud computing environment using dynamic clustering algorithm. *Neural Comput. Appl.* **2016**, *29*, 279–293. [\[CrossRef\]](#)
26. Abdulhamid, S.; Latiff, M.; Abdul-Salaam, G.; Madni, S. Secure scientific applications scheduling technique for cloud computing environment using global league championship algorithm. *PLoS ONE* **2016**, *11*, e0158102. [\[CrossRef\]](#)
27. Abdullahi, M.; Ngadi, M. Symbiotic Organism Search optimization-based task scheduling in cloud computing environment. *Future Gener. Comput. Syst.* **2016**, *56*, 640–650. [\[CrossRef\]](#)
28. Liu, L.; Luo, T.; Du, Y. A new task scheduling strategy based on improved ant colony algorithm in IaaS layer. In Proceedings of the 2019 International Conference on Computer, Information and Telecommunication Systems (CITS), Beijing, China, 28–31 August 2019.
29. Naithani, P. Genetic Algorithm Based Scheduling to Reduce Energy Consumption in Cloud. In Proceedings of the 2018 Fifth International Conference on Parallel, Distributed and Grid Computing (PDGC), Solan, India, 20–22 December 2018.
30. Wu, D. Cloud Computing Task Scheduling Policy Based on Improved Particle Swarm Optimization. In Proceedings of the 2018 International Conference on Virtual Reality and Intelligent Systems (ICVRIS), Hunan, China, 10–11 August 2018.
31. Yalcinoz, T.; Rudion, K. Economic Load Dispatch Using an Improved Particle Swarm Optimization based on functional constriction factor and functional inertia weight. In Proceedings of the 2019 IEEE International Conference on Environment and Electrical Engineering and 2019 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I&CPS Europe), Genova, Italy, 11–14 June 2019.
32. Marichelvam, M.; Prabakaran, T.; Yang, X. Improved cuckoo search algorithm for hybrid flow shop scheduling problems to minimize makespan. *Appl. Soft Comput.* **2014**, *19*, 93–101. [\[CrossRef\]](#)
33. Ouaraab, A.; Ahiod, B.; Yang, X. Discrete cuckoo search algorithm for the travelling salesman problem. *Neural Comput. Appl.* **2014**, *24*, 1659–1669. [\[CrossRef\]](#)

Short Biography of Authors



Jianguo Zheng received the B.S. degree in mathematics from Zhongshan University, the M.S. degree in system engineering from Xidian University, China, and the Ph.D. degree from the National Key Laboratory for Radar Signal Processing, Xidian University, in 2002. He was a Visiting Professor with the University of Ballarat, Ballarat, VIC, Australia. He is currently a Professor with the School of Business and Management, Donghua University, China. His research interests include data mining, intelligence decision systems, neural networks, techno-economic analysis, and operations management.



Yilin Wang received the B.S. degree in information management from Henan University of Science and Technology, and she continued to pursue a master's degree and doctorate degree at Donghua University. She is currently a PhD student in management science and engineering. Her research interests include bat algorithm, data mining and optimization.