Research on Sparrow Search Optimization Algorithm for Multi-Objective Task Scheduling in Cloud Computing Environment

Zhi-Yong Luo[*], Ya-Nan Chen and Xin-Tong Liu
School of Computer Science and Technology, Harbin University of Science and Technology, Harbin 150080, China

Abstract. In cloud computing, optimizing task scheduling is crucial for improving overall system performance and resource utilization. To minimize cloud service costs and prevent resource wastage, advanced techniques must be employed to efficiently allocate cloud resources for executing tasks. This research presents a novel multi-objective task scheduling method, BSSA, which combines the Backtracking Search Optimization Algorithm (BSA) and the Sparrow Search Algorithm (SSA). BSA enhances SSA's convergence accuracy and global optimization ability in later iterations, improving task scheduling results. The proposed BSSA is evaluated and compared against traditional SSA and other algorithms using a set of 8 benchmark test functions. Moreover, BSSA is tested for task scheduling in cloud environments and compared with various metaheuristic scheduling algorithms. Experimental results demonstrate the superiority of the proposed BSSA, validating its effectiveness and efficiency in cloud task scheduling.

Keywords: Cloud computing, task scheduling, multi-objective optimization, sparrow search algorithm

1. Introduction

Cloud computing has brought significant advancements to the fields of computing and storage through the rapid deployment of large-scale applications, providing efficient IT services for numerous jobs and tasks [1]. Cloud servers not only offer the required hardware resources to users but also provide software resources, ensuring the elasticity and efficient management of cloud resources. Among them, the resource management subsystem is used to execute scheduling tasks and map the tasks to be processed in a more efficient virtual machine manner [2]. Due to the uncertainty of task workflows, sequences, and characteristics, as well as the uncertain quantity of available resources, resources are randomly allocated to tasks [3]. Therefore, in the scheduling process, performance metrics, such as makespan, system utilization, response time, and network-based performance, such as traffic and network communication costs, are calculated to find the optimal virtual machine for each task [4].

Metaheuristic algorithms have demonstrated their effectiveness and robustness in solving a multitude of real-world problems [5]. Several multi-objective task scheduling algorithms based on metaheuristic algorithms, such as Clustering Sparrow Search Algorithm, Dynamic Decision Algorithm, Differential Evolution Algorithm, Whale Optimization Algorithm, and others have been widely researched and applied in cloud computing environments. These algorithms have exhibited favorable outcomes in task scheduling and resource allocation issues, which have effectively enhanced system performance and resource utiliza-

[*]Corresponding author. Zhi-Yong Luo, School of Computer Science and Technology, Harbin University of Science and Technology, Harbin 150080, China. E-mail: luozhiyongemail@sina.com.

tion. Therefore, multi-objective task scheduling algorithms have become a prominent research direction in cloud computing, providing significant support for the efficient operation and sustainable development of cloud computing applications.

The Sparrow Search Algorithm (SSA) is a population-based swarm intelligence optimization algorithm inspired by the foraging and anti-predator behavior of sparrows, and it was proposed as a metaheuristic algorithm in 2020 [6]. This algorithm effectively simulates the process of sparrows exploring and exploiting food sources using a mathematical model. Initially, a random population of sparrows is generated, and each sparrow's foraging position is evaluated using a fitness function. The Sparrow Search Algorithm is widely used to solve real-world optimization problems due to its simple concepts and ease of implementation. However, similar to other swarm intelligence algorithms, it is prone to get stuck in local optima during later iterations, which can affect the quality of the obtained solutions.

On the other hand, the Backtracking Search Optimization Algorithm (BSA) possesses robust global optimization capabilities. In a study by Zaman et al. [7], BSA was employed to optimize the Particle Swarm Optimization algorithm, enhancing convergence accuracy and avoiding local optima. In another study by Chen et al. [8], the Backtracking Search was used as a variation strategy in the Poplar Optimization Algorithm (POA), maintaining algorithm diversity to achieve better optimization performance. Therefore, in this paper, the BSA algorithm is utilized as a local search technique to improve the convergence accuracy and global optimization capability of SSA.

In this paper, we propose the BSSA algorithm as a multi-objective task scheduling method for addressing task scheduling issues in cloud computing environments. The primary objectives of the algorithm are to minimize the makespan and improve resource utilization while considering load balancing within the system, aiming to optimize job scheduling. The main contributions of this paper are summarized as follows:

1) An optimized version of the Sparrow Search Algorithm (SSA) is proposed by introducing the Backtracking Search Algorithm (BSA) as a local search technique to enhance SSA's global search capability.
2) The BSSA algorithm is subjected to benchmark function testing and analysis to validate its optimization performance.
3) The proposed method is applied to the task scheduling problem in cloud computing environments, considering factors such as minimum makespan, resource utilization, and throughput in the cloud environment.

To achieve these objectives, the performance of the BSSA algorithm is evaluated through benchmark function tests to assess the algorithm's performance. Additionally, the proposed method is compared with other heuristic algorithms for task scheduling performance in CloudSim.

In this article, the following sections are organized as follows: Section 2 reviews related work on task scheduling in cloud computing. Section 3 presents the problem of task scheduling in the cloud and describes the proposed BSSA algorithm in detail. Section 4 describes the experimental environment used to analyze the performance of the BSSA algorithm. Finally, Section 5 concludes the paper and summarizes its main contributions.

2. Related Work

In cloud computing environments, the task scheduling problem is classified as an NP-hard problem, which requires the use of heuristic or metaheuristic algorithms to improve the ability of algorithms to find optimal solutions. As a result, hybrid heuristic algorithms are increasingly being employed to address task scheduling and resource allocation issues. Many scholars have proposed a variety of algorithms for task scheduling in cloud computing. In [9], a two-stage hybrid metaheuristic algorithm called CSSA-DE (Clustering Sparrow Search Algorithm with Differential Evolution) was proposed for task scheduling in cloud computing environments. The algorithm first evenly distributes incoming tasks among clusters and stores them in the queue of the MCH (Master Cluster Head). Then, the algorithm uses SSA and DE methods to search for the most suitable task server pair, to minimize resource fragmentation and energy costs while balancing the workload assigned to virtual machines. In [10], a dynamic decision-based method was proposed to tackle issues related to resource consumption and time execution. The proposed method employs an empirical algorithm to adapt quickly to cloud computing tasks, reducing the resource and time computation of mobile devices significantly. In [11], an advanced method named IWC (Improved WOA-based Cloud Task Scheduling) was proposed to enhance the search capability of the WOA-based method for optimal solutions. The proposed IWC exhibits better convergence speed and accuracy in searching for the optimal task scheduling plan, achieving better system resource utilization performance. In [12], a fuzzy self-defense algorithm-based multi-objective task scheduling optimization method for cloud computing was proposed. The method aims to optimize the shortest time, resource load balance degree, and multi-objective task completion cost for cloud computing multi-objective task scheduling. The experimental results show that this method can improve the maximum completion time, deadline violation rate, and virtual machine resource utilization rate. In [13], a modified moth search algorithm (MSDE) based on differential evolution (DE) was proposed to minimize the makespan required for scheduling multiple tasks on different virtual machines in cloud task scheduling problems. The DE algorithm is used as a local search method to enhance the exploration and exploitation capabilities of MSA. Experimental results show that the algorithm outperforms compared algorithms based on performance indicators. In [14], a Henry gas solubility optimization algorithm (HGSWC) based on the whale optimization algorithm (WOA) and opposition-based comprehensive learning (COBL) was proposed to optimize task scheduling. HGSWC was tested and verified on a set of 36 optimization benchmark functions as well as synthetic and real workloads. Simulation experimental results demonstrate that HGSWC finds near-optimal solutions without the computational overhead and outperforms six well-known metaheuristic algorithms. In [15], a task scheduling method based on the Hungarian algorithm was proposed. It considers unequal numbers of tasks and clouds and pairs the tasks to make scheduling decisions that minimize the mid-stop time. The proposed algorithm was simulated with 22 different datasets and compared with other improved Hungarian algorithms. The performance evaluation shows that the proposed algorithm produces better mid-stop times.

When considering task completion time, resource utilization, and throughput comprehensively, the literature review above did not yield similar optimal results. Therefore, in this study, we employ the backtracking search and Sparrow Search algorithms to optimize tasks and resources based on performance indicators such as minimum makespan time, resource utilization, and throughput between virtual machines. Tasks are classified based on their dissimilarities, taking into account the use of virtual resource nodes. Subsequently, tasks and virtual resources are mapped rationally to generate a scheduling sequence.

3. Problem and algorithm description

3.1. Description of the task scheduling problem

This article addresses the problem of cloud task scheduling, which involves allocating or scheduling a large number of tasks on available computing resources to minimize Makespan and improve resource utilization, and throughput. Specifically, we consider a cloud system (CS) comprising Npm physical machines, with each machine having n virtual machines. Below are definitions for some related issues.

Definition 1 The set of virtual machines $PM=[VM_1,VM_2,\cdots,VM_n]$, where $VM_j=[SIDV_j,MIPS_j]$, *SIDV* represents the sequence number of virtual machines and *MIPS* represents the computing power of virtual machines.

Definition 2 Suppose there are m tasks assigned on the VM and each task has the following characteristics: where $SIDT_i$ is the sequence number of the task and T_len_i is the length of the task. Time ECT_i refers to the expected completion time of T_i, PI_i refers to the priority of the *i*-th task. *m* tasks and *n* VMs have an *ECT* matrix represented by Eq. (1), where ECT_{ij} is pointed out by Definition 3.

$$ECT=\begin{bmatrix} ECT_{1,1} & ECT_{1,2} & \cdots & \cdots & ECT_{1,n} \\ ECT_{2,1} & ECT_{2,2} & \cdots & \cdots & ECT_{2,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ ECT_{m,1} & ECT_{m,2} & \cdots & \cdots & ECT_{m,n} \end{bmatrix} \tag{1}$$

Definition 3 Execution time ECT_{ij}, which represents the execution time of the T_i-th task on VM_j of the virtual machine. It is calculated by Eq. (2).

$$ECT_{ij}=\frac{T\_len_i}{MIPS_j},\quad j=1,2,...,\text{n} \tag{2}$$

where $MIPS_j$ denotes the computing power of the *j*-th VM.

Definition 4 The completion time of a virtual machine (CT_j) refers to the runtime of the virtual machine after executing the last task on the *j*-th virtual machine. It is calculated by Eq. (3).

$$CT_j=\sum_{i=1}^{k}ECT_{ij} \tag{3}$$

where *k* is the number of tasks assigned to V_j by scheduling.

Definition 5 In computer terminology, the maximum completion time of all tasks on all virtual machines, also known as Makespan, is the time interval between the start and end of the last task. This metric is calculated using Eq. (4) and minimizing it indicates better performance in terms of early workload completion [16].

$$Makespan = \max_{1 \le j \le n} \left\{ CT_j \right\} \quad (4)$$

Definition 6 The throughput is a metric that represents the number of tasks executed in a given unit of time, typically measured in tasks per second or tasks per minute [17]. As shown in Eq. (5). A higher throughput indicates better performance, indicating that more tasks can be completed within a given time frame.

$$Throughput = \frac{D}{Makespan} \quad (5)$$

where *D* denotes the total number of tasks performed in the cloud data center.

Definition 7 Resource Utilization Rate (Ru) is a performance metric that calculates the utilization rate of computing resources. A higher utilization rate translates to greater profit for the cloud service provider [18]. It can be calculated using Eq. (6).

$$Ru = \frac{\sum_{j=1}^{n} CT_j}{Makespan \times n} \quad (6)$$

In this article, the objective is to minimize Makespan, maximize resource utilization rate, and achieve high throughput through an appropriate scheduling order for executing all tasks in the shortest time possible. The scheduling algorithm aims to find the Pareto global optimal solution for the best scheduling outcome. The assumption is made that the tasks are independent, and there is no communication time between them. The only evaluated time is the Makespan time, which represents the total completion and waiting time for all tasks. The proposed multi-objective function is formulated as shown in Eq. (7).

$$Fitness = \min \left\{ Makespan + \frac{1}{Ru} + \frac{1}{Throughput} \right\} \quad (7)$$

3.2. The Sparrow Search Algorithm

The Sparrow Search Algorithm (SSA) simulates sparrows' foraging and evading behavior in nature, using the concepts of discoverers and vigilant to represent exploration and exploitation abilities. Sparrows, being social animals, can be categorized into discoverers and joiners during foraging activities. Discoverers are responsible for seeking food and providing the entire sparrow population with foraging locations and directions, while joiners use the information from discoverers to obtain food. When the warning threshold is lower, discoverers can perform extensive search operations; if it exceeds the warning threshold, both discoverers and joiners will escape to a safe area. When vigilantes detect danger, edge sparrows will fly to safety, while middle sparrows will randomly move closer to other sparrows. The specific implementation steps of SSA are as follows:

Step 1: Initialize the population size N, individual space dimension D, the maximum number of iterations T, and the proportion of discoverer individuals PD and sentinel individuals SD in the population.

Step 2: Calculate the fitness of each individual and obtain the current optimal value fMin and optimal solution Xbest as the global best value.

Step 3: Based on the obtained fitness values, sort the corresponding individuals, select the ones with better fitness as discoverers according to the discoverer proportion PD, and update the positions of discoverers using equation (8).

$$X_{i,j}^{t+1} = \begin{cases} X_{i,j}^{t} \cdot exp(-\frac{i}{\alpha \cdot T}), & if \ \mathrm{R}_2 < \mathrm{ST} \\ X_{i,j}^{t} + Q \cdot L, & if \ \mathrm{R}_2 \ge \mathrm{ST} \end{cases} \quad (8)$$

where *t* represents the current number of iterations. *T* denotes the maximum number of iterations. $X_{i,j}$ denotes the virtual machine assigned to the *j*-th task in the *i*-th set of solutions. $\alpha \in (0,1]$ is a random number. $R_2 \in [0,1]$ and denote the warning and safety values, respectively. *Q* is a random number that follows a normal distribution. *L* denotes a 1 × *D* matrix and each element in the matrix is all 1.

Step 4: The remaining joiners update their positions using equation (9).

$$X_{i,j}^{t+1} = \begin{cases} Q \cdot exp(\frac{X_{worst}^{t} - X_{i,j}^{t}}{i^2}), if \quad i > n/2 \\ X_P^{t+1} + \left| X_{i,j}^{t} - X_P^{t+1} \right| \cdot A^{+} \cdot L, otherwise \end{cases} \tag{9}$$

where X_p is the optimal position occupied by the discoverer after the update and X_{worst} denotes the current global worst dispatch sequence. A denotes a 1 × D matrix each element is randomly assigned 1 or -1 and $A^{+} = A^{T}(AA^{T})^{-1}$.

Step 5: Select individuals randomly as sentinels based on the proportion SD, which can be represented as follows:

$$X_{i,j}^{t+1} = \begin{cases} X_{best}^{t} + \beta \cdot \left| X_{i,j}^{t} - X_{best}^{t} \right|, if \ f_i > f_g \\ X_{i,j}^{t} + K \cdot (\frac{X_{i,j}^{t} - X_{worst}^{t}}{(f_i - f_w) + \varepsilon}), if \ f_i = f_g \end{cases} \tag{10}$$

where X_{best} is the current global best dispatch sequence. β as a step control parameter, is a random number with a normal distribution with mean 0 and variance 1. $K \in [-1,1]$ is a random number and f_i is the current fitness value of individual sparrows. f_g and f_w are the current global best and worst fitness values, respectively. ε is the smallest constant to avoid the denominator being zero.

Step 6: After updating the positions using equations (8) to (10), calculate the fitness of each individual, obtain the current optimal value *pMin* and optimal solution *pbest*, and update *fMin* and X_{best}.

Step 7: Check if the maximum number of iterations has been reached. If yes, terminate the loop and output the optimal value fMin and optimal solution Xbest. Otherwise, return to Step 3. The steps of SSA are illustrated in Algorithm 1.

3.3. The Backtracking Search Algorithm

The Backtracking Search Algorithm (BSA) [19]efficiently generates a population in each generation by utilizing selection, mutation, and crossover operations on the current population and historical populations. Compared to other genetic algorithms like Differential Evolution (DE) [20], BSA's search direction matrix allows for more effective control of global and local search. Additionally, BSA uses a complex crossover strategy with random 0-1 matrices to generate new individuals, enhancing its problem-solving capabilities.

3.3.1. Mutation

In the mutation phase of BSSA, to enhance the randomness of individuals in Xbest, it is defined by Equation (11).

$$X_{best} := permuting(X_{best}) \tag{11}$$

where permuting is the random shuffle function.

By using Equation (12), the initial population undergoes mutation operations to generate new individuals.

$$Mutant = X + F(X_{best} - X) \tag{12}$$

where F is the search direction, usually 2*randn, and randn follows a normal distribution from 0 to 1.

3.3.2. Crossover

The crossover operation stage generates the final population T, which generates an N row D column binary matrix map, and if $map_{n,m} = 1$, then $T_{n,m} := X$.

3.3.3. Selection-II

A greedy strategy is employed to select the optimal individuals for the next generation, as defined by Equation (13).

$$X = \begin{cases} T, if \quad f(T) \le f(X) \\ X, otherwise \end{cases} \tag{13}$$

3.4. The proposed BSSA method

In this section, a BSA-based SSA search algorithm is proposed to address the task scheduling problem in cloud computing. To enhance the performance of the SSA algorithm in handling multi-objective optimization and constraint optimization problems, mutation, crossover, and selection mutation strategies from the BSA are employed to optimize the current and historical populations. The resulting task scheduling algorithm, presented in this paper, consists of two main stages. The first stage involves initializing the population and computing the fitness function values. The second stage utilizes the SSA algorithm to update the current population X until the stopping criteria are met. A detailed description is provided below.

Algorithm 1 Sparrow search algorithm（SSA）

```
Input:
G: the maximum iterations
PD: the number of producers
SD: the number of sparrows who perceive the danger
R2: the alarm value
n: the number of sparrows
Initialize a population of n sparrows and define its relevant parameters.
Output: Xbest, fg.
1: while ($t$ < G)
2: Rank the fitness values and find the current best individual and the current worst individual.
3: R2 = rand(1)
4: for i = 1:PD
5:   Using equation (8) update the sparrow's location;
6: end for
7: for i = (PD + 1) : n
8:   Using equation (9) update the sparrow's location;
9: end for
10: for i = 1:SD
11:   Using equation (10) update the sparrow's location;
12: end for
13: Get the current new location;
14: If the new location is better than before, update it;
15: $t = t + 1$
16: end while
17: return Xbest, fg.
```

3.4.1. Initialization phase

The BSSA method starts by setting certain parameters and performing initialization operations. It generates a matrix X with N rows and M columns, representing N sets of solutions, with each set consisting of M dimensions. Here, N represents the population size, and M represents the number of tasks. Each solution X_i , $i \in (1, 2, ..., N)$ in the set is initialized using Eq. (14).

$$X_{i,j} = round(low + rand * (up - low)) \quad (14)$$

where j represents the j-th task, low represents the lower bound of the search space (which is equal to 1) and up represents the upper bound of the search space (which is equal to n, the number of virtual machines). After initialization, the fitness function value for each solution is computed using Eq. (7), and the best solution Best is determined for subsequent update operations.

To illustrate with an example, suppose there are 10 tasks and 5 virtual machines. After evaluating the fitness function for each solution in the population $X_i = [4,1,2,1,4,3,5,2,3,1]$, the best solution is determined. The corresponding scheduling sequence would assign the first task to virtual machine 4, the second task to virtual machine 1, and so on, until all tasks are assigned and the final scheduling sequence is generated.

3.4.2. Update phase

First, the fitness function values are calculated based on Equation (7), and the local optimal values are recorded. The individuals are then sorted, and those with better fitness are selected as explorers according to the proportion PD. Their positions are continuously updated using Equation (8). Similarly, followers update their positions based on Equation (9). Random individuals are selected as vigilantes according to the proportion SD, and they use Equation (10) to search for safe or food-abundant areas.

second, the best population is used as the historical population for relevant operations. To enhance the randomness of individuals in Xbest, they are updated using Equation (11). The population is then subjected to mutation operations to generate a new population, as described in Equation (12). Finally, a greedy strategy is employed to select the best sequence as the new allocation sequence, and it is updated using Equation (13).

The updating stage is repeated until the termination condition, which is the maximum number of iterations, is met. Finally, the best allocation solution for tasks on virtual machines is returned. Algorithm 2 provides the pseudocode for the BSSA method, including the remaining implementation parts.

Algorithm 2 Pseudo-code of proposed BSSA

```
Input: $m$ number of tasks, $n$ number of VMs.
Output: the optimal scheduling sequence.
Initialize. Random assignment of virtual machines to tasks in the population by using Eq. (14).
The value of the fitness function is calculated by using Eq. (7), and the local optimum Xbest is recorded.
Repeat until termination criteria are met:
    Update the discovery rate of each sparrow by using Eq. (8).
    Update the following rate of each sparrow by using Eq. (9).
    Update the vigilance rate of each sparrow by using Eq. (10).
    Randomly of local optima by using Eq. (11).
    for $i$ = 1: N
        Variation operation by using Eq. (12).
        if map = 1
            Mutant = $X_{best}$
        end
    end
    Calculate fitness by using Eq. (7).
    Update the local optimum.
Return to the optimal schedule.
```

Assuming a population size of m, a task number of n, and a maximum iteration number of T, the time complexity of the algorithm can be expressed as O(m*T+ m*n). As the population size m is usually much smaller than the number of iterations T, the overall time complexity of the algorithm can be simplified as O(m*T).

4. Simulation experiments and discussion

This section will present the experimental design, performance evaluation, results analysis, and discussions of the proposed BSSA algorithm. To validate the effectiveness of the proposed method (BSSA), the experiments are divided into two series. The first series of the experiment involves simulation experiments on eight benchmark test functions to evaluate the algorithm's ability to solve global optimization problems. In the second series, experiments are conducted in the CloudSim infrastructure using real and synthetic workload instances to test the BSSA's capability in handling cloud task scheduling problems.

4.1. Experimental series 1: global optimization problems

The experiment introduced eight benchmark test functions, which are presented in Table 1. Functions F1 to F5 are multidimensional single-peak functions commonly used to evaluate the convergence accuracy and speed of the algorithm. Functions F6 to F8 are multidimensional multi-peak functions mainly used to test the global search ability of the algorithm. To reduce the influence of errors, the algorithm was evaluated in 30 independent runs, and three performance indicators were used: the optimal value, the average value, and the standard deviation. The average value reflects the true convergence accuracy and speed of the algorithm, while the standard deviation reflects the robustness and stability of the algorithm. The proposed algorithm was compared with four other algorithms, including SSA, BSA, MFO [21], and GWO [22], with parameter configurations based on the corresponding references, as shown in Table 2.

Table 1

Unconstrained benchmark function details used in simulations

Function	Equation	Range	Dimension	Optimum
F1	$f(x)=\sum_{i=1}^{D}\lvert x_i\rvert+\prod_{i=1}^{D}\lvert x_i\rvert$	[-10,10]	30	0
F2	$f(x)=\max_{i=1,2,3,\ldots n}\lvert x_i\rvert$	[-100,100]	30	0
F3	$f(x)=\sum_{i=1}^{D-1}[100(x_{i+1}-x_i^{2})^2+(x_i-1)^2]$	[-30,30]	30	0
F4	$f(x)=\sum_{i=1}^{D}(x_i+0.5)^2$	[-100,100]	30	0
F5	$f(x)=\sum_{i=1}^{D}x_i^4+rand(0,1)$	[-1.28,1.28]	30	0
F6	$f(x)=-\sum_{i=1}^{D}x_i sin(\sqrt{\lvert x_i\rvert})$	[-500,500]	30	− 12,569
F7	$f(x)=-20\exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}x_i^2})-\exp(\frac{1}{D}\sum_{i=1}^{D}cos(2\pi x_i))+20+\exp(1)$	[-32,32]	30	0
F8	$f(x)=\frac{\pi}{n}\left\{10sin(\pi y_1)+\sum_{i=1}^{D-1}(y_i-1)^2[1+10sin^2(\pi y_{i+1})]+(y_D-1)^2\right\}$ $+\sum_{i=1}^{D}u(x_i,10,100,4); y_i=1+\frac{x_i+1}{4}; u(x_i,a,k,m)=\begin{cases}k(x_i-a)^m & x_i>a\\ 0 & -a<x_i<a\\ k(-x_i-a)^m & x_i<-a\end{cases}$	[-50,50]	30	0

Table 2

Parameter settings

Algorithm	Parameter
BSA	*mix-rate* = 1.0,F = 3 × *randn*
MFO	b = 1
GWO	A = 2*a*rand-a, a* = 2-*gen*(2⁄*maxgen*),C=2**rand*
SSA	PD=0.2,SD=0.2,R_2=0.8
BSSA	PD=0.2,SD=0.2,R_2=0.8, F = 2 × *randn*

Based on the results of the test functions presented in Table 3, it can be concluded that the BSSA algorithm achieved the optimal value in all the tested functions, except for F5 and F8. Regarding optimization accuracy, the BSSA algorithm outperformed other algorithms with better average values, indicating its stronger optimization ability. As for stability, the BSSA algorithm exhibited a lower standard deviation than the SSA algorithm in some tested functions, indicating its high stability.

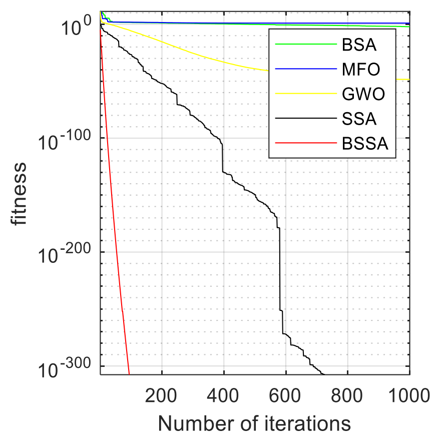
Table 3

Comparison of optimization results of benchmark test functions

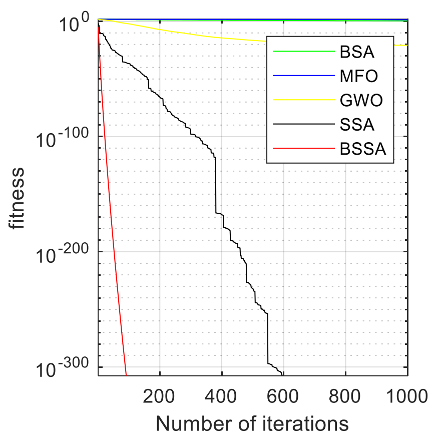
Function	Res	BSA	MFO	GWO	SSA	BSSA
F1	Best	4.78E-04	7.44E-05	9.86E-51	0.00E+00	**0.00E+00**
	Mean	7.87E-04	2.77E+01	2.76E-49	5.32E-270	**0.00E+00**
	Std.	2.01E-04	1.77E+01	2.93E-49	0.00E+00	**0.00E+00**
F2	Best	1.20E+00	1.39E+01	5.77E-23	0.00E+00	**0.00E+00**
	Mean	2.64E+00	2.84E+01	1.52E-21	4.05E-298	**0.00E+00**
	Std.	6.92E-01	1.16E+01	2.58E-21	0.00E+00	**0.00E+00**
F3	Best	2.01E+01	3.49E+00	2.51E+01	1.21E-12	**0.00E+00**
	Mean	5.92E+01	1.57E+04	2.62E+01	1.56E-06	**1.28E-29**
	Std.	3.15E+01	3.38E+04	6.85E-01	3.13E-06	**3.62E-29**
F4	Best	5.10E-06	2.74E-06	3.61E-06	7.72E-19	**0.00E+00**
	Mean	2.01E-05	1.33E+03	1.70E-01	1.56E-15	**0.00E+00**
	Std.	1.34E-05	3.46E+03	1.75E-01	3.57E-15	**0.00E+00**
F5	Best	5.61E-03	1.90E-02	4.89E-05	9.50E-06	**4.28E-07**
	Mean	1.21E-02	1.02E+00	2.36E-04	9.25E-05	**2.13E-05**
	Std.	5.13E-03	3.19E+00	1.41E-04	7.65E-05	**1.98E-05**
F6	Best	-1.02E+04	-1.08E+04	-7.61E+03	-9.94E+03	**-1.26E+04**
	Mean	-9.58E+03	-8.76E+03	-6.46E+03	-8.66E+03	**-1.26E+04**
	Std.	3.06E+02	8.79E+02	6.96E+02	6.00E+02	**1.85E-12**
F7	Best	1.01E-02	2.46E-04	7.99E-15	8.88E-16	**8.88E-16**
	Mean	5.27E-02	8.47E+00	1.17E-14	8.88E-16	**8.88E-16**
	Std.	5.69E-02	9.35E+00	2.87E-15	0.00E+00	**0.00E+00**
F8	Best	1.41E-08	2.32E-07	3.65E-07	5.90E-21	**1.57E-32**
	Mean	8.20E-08	2.43E-01	1.49E-02	1.14E-15	**1.57E-32**
	Std.	4.52E-08	6.72E-01	1.14E-02	5.98E-15	**5.57E-48**

From the single-peak convergence curve plots in Fig.1(a) to (e), it can be obtained that for functions F1 and F2, BSSA shows a nearly linearly decreasing curve towards the theoretical optimum without stagnation and reaches the optimal value early, indicating strong convergence speed and accuracy. SSA exhibits a flat trend or stagnation on the convergence curves for functions F3, F4, and F5, indicating lower optimization accuracy and getting trapped in local optima. The convergence curve of the improved BSSA algorithm demonstrates a steeper trend and reaches the turning point faster, indicating a significant enhancement in convergence speed and accuracy. It can be inferred from the multimodal convergence curve plots in Fig.1(f) to (h) that for the test functions F6 to F8, the BSSA algorithm achieves the smallest solution value and reaches the minimum value first, demonstrating excellent optimization capabilities. For function F7, both SSA and BSSA attain the theoretical optimal value of 0, but the turning point of the BSSA convergence curve is earlier and the convergence speed is faster.

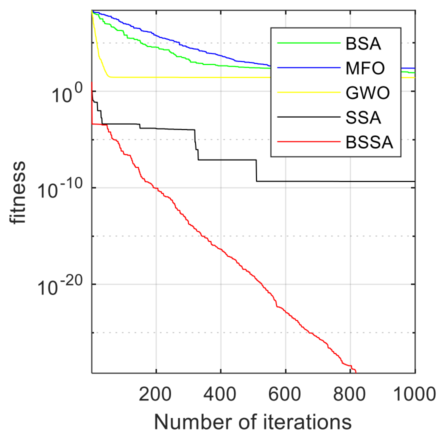
By analyzing the convergence curves, it is clear that the BSSA algorithm performs exceptionally well in solving both unimodal and multimodal functions. This is primarily attributed to the incorporation of a backtracking search strategy, which perturbs the algorithm when it is trapped at a local extreme and expands the search space, thereby increasing the likelihood of escaping from local optimal solutions.



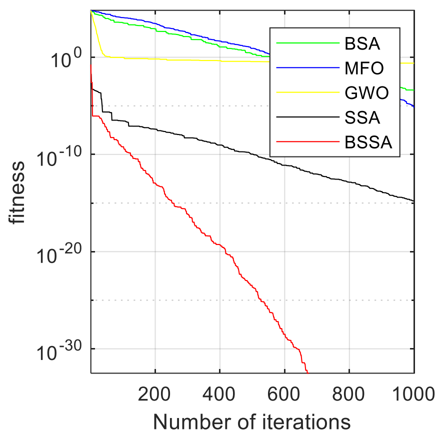
(a)F1



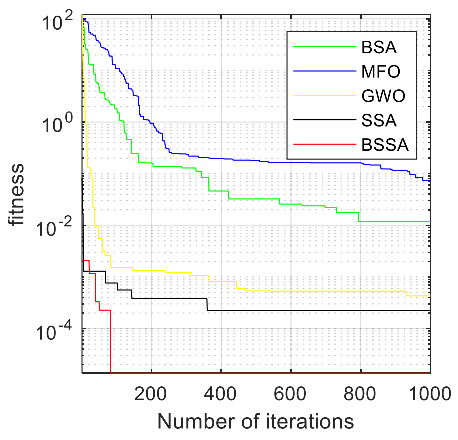
(b)F2



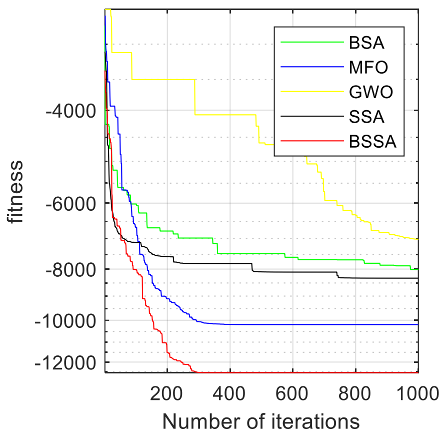
(c)F3



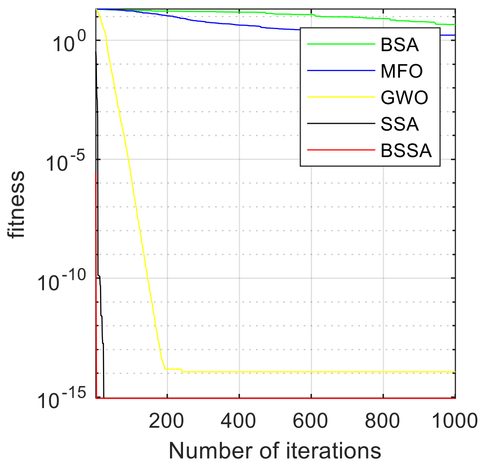
(d) F4



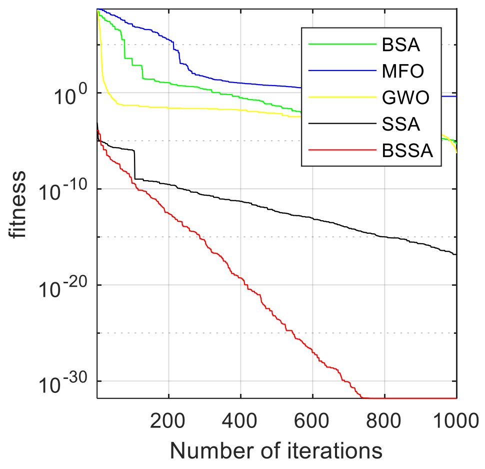
(e)F5



(f)F6



(g)F7



(h) F8

Fig. 1. Convergence curve of the test function.

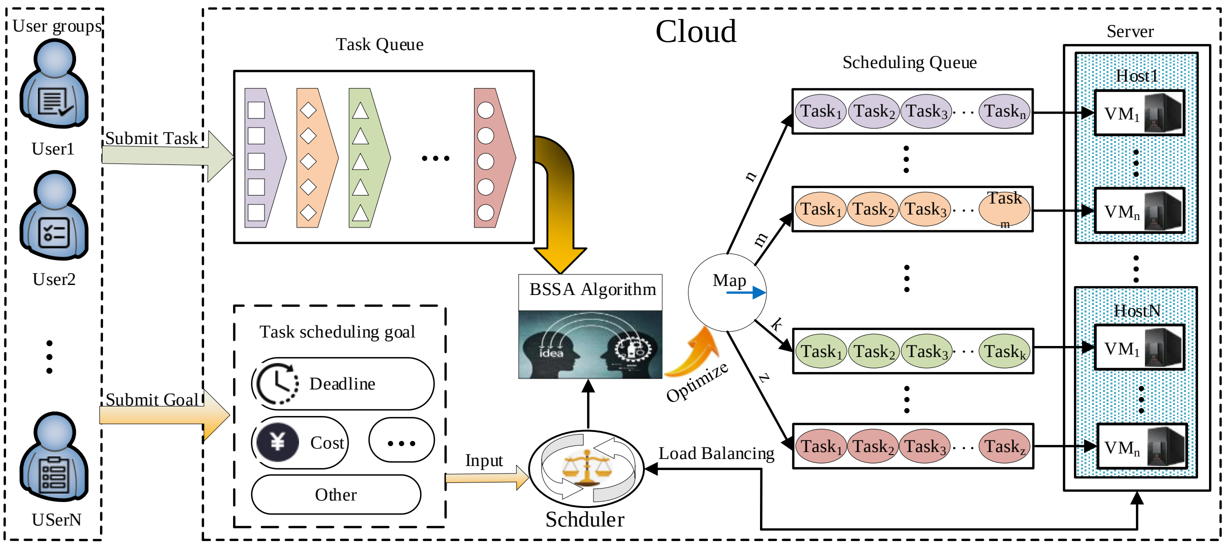


Fig. 2. Experimental model of cloud task scheduling.

4.2. Experimental series 2: task scheduling problem in cloud computing

The experiments are divided into two parts, both using CloudSim 4.0 for simulation to validate the performance of the proposed algorithm under the same initial conditions. In the first part, metaheuristic algorithms SSA, PSO [23], and the proposed algorithm are applied using the Google Cloud Job dataset (GoCJ) [24] to simulate high-performance computing tasks in cloud computing. This aims to verify the improved performance of the proposed algorithm. In the second part, to compare the scheduling performance of the algorithms, the Random dataset is used to compare this algorithm with the CSSA-DE [9] algorithm and LMBPSO [25] algorithm. The experimental model for cloud computing task scheduling is shown in Figure 2.

4.2.1. Evaluation results of real trace datasets

Two data centers are created, and the tasks are assigned using the Google Cloud Job dataset (GoCJ). The virtual machines are allocated dynamically, with each task being assigned to 10% of the total number of virtual machines. The virtual machines have a range of MIPS from 100 to 4000, RAM of 512MB, bandwidth of 1000 Mbps, and memory of 10GB. The parameters for the algorithm are listed in Table 4.

Table 4
Parameter settings

Algorithm	Parameter
PSO	*N = 100,w=0.9,c1=1.49,c2=1.49*
SSA	*N = 100,PD=0.2,SD=0.2,R2=0.8*
BSSA	*N = 100,PD=0.2,SD=0.2,R2=0.8, F = 2 × randn*

Fig.3 shows the Makespan of task execution time. For the task dataset, BSSA improved 23.21% and 45.07% on average compared to the original SSA and PSO, respectively. It can be concluded that BSSA has better performance than the original SSA and PSO algorithms in task scheduling.

The paragraph describes the results shown in Fig.4, which displays the average resource utilization for all datasets. BSSA and SSA achieved average scores of 98.91% and 98.55%, respectively, on the task dataset, while PSO also had good results with an average of 97.49%. It can be concluded that, for experiments using datasets with quantifiable tasks, BSSA has the shortest generation time and increased resource utilization compared to SSA and PSO algorithms. This means that it has excellent behavior when performing task scheduling using improved local and global search operations.

The paragraph describes the results shown in Fig.5, which presents the throughput values of the datasets with the same number of virtual machines. Higher throughput values indicate more effective scheduling algorithms that can run more tasks per unit of time. BSSA had an average throughput value of 0.87, while the original SSA and PSO had average throughput values of 0.65 and 0.44, respectively. This indicates that BSSA also improved the scheduling efficiency in terms of the number of tasks processed by virtual machines. It can also be noted that BSSA achieved the best throughput result of 1.61 for a task number of 1000, which is significantly higher than the results for typical-sized datasets. This suggests excellent utilization of random assignment results.

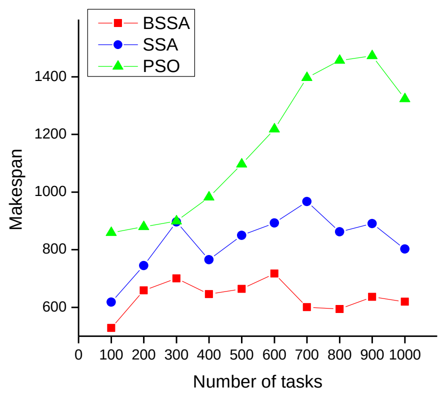


Fig. 3. Makespan time using the GoCJ dataset.

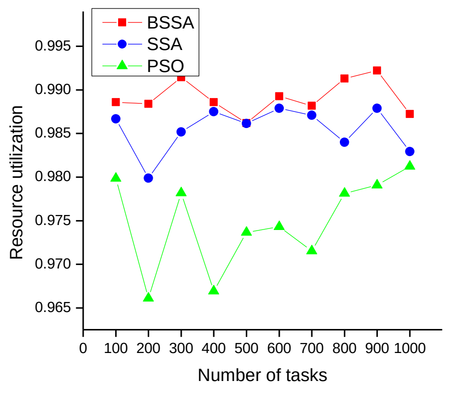


Fig. 4. Resource Utilization using the GoCJ dataset.

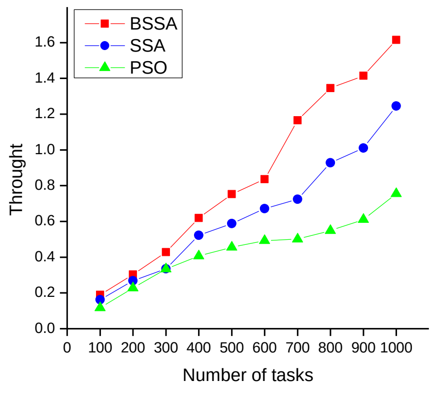


Fig. 5. Throughput using the GoCJ dataset.

4.2.2. Evaluation results of random datasets

In this section, the proposed optimization algorithm (BSSA) is evaluated and compared with CSSA-DE and LMBPSO using simulation in CloudSim. The task dataset is generated using a random dataset, with task sizes ranging from 1k to 5k instructions (MIs). The experiment allocates 100 virtual machines and assigns 150, 300, 450, 600, 750, 900, and 1050 tasks to the system. The virtual machines have a range of MIPS from 700 to 2500, RAM of 512MB, bandwidth of 1000 Mbps, and memory of 10GB.

Based on the experimental results shown in Figure 6, it is observed that the BSSA method outperforms CSSA-DE and LMBPSO in reducing Makespan when tested with a random dataset. On average, BSSA saves 18.45 and 35.54% of time performing workloads compared to CSSA-DE and LBMPSO.

Resource utilization reflects the importance of efficiently using cloud resources, including the appropriate ratio of server running time to total cost. The experimental results, as shown in Figure 7, confirm that the BSSA heuristic algorithm effectively utilizes resources. The highest resource utilization achieved by the BSSA algorithm is 97.77%, which represents an improvement of 1.78% for the workload of 600 tasks compared to CSSA-DE and an improvement of nearly 24.86% compared to LBMPSO. Therefore, it can be concluded that the BSSA method effectively schedules tasks in the system and helps evenly distribute tasks among available resources, thus maintaining a balanced mode in the system.

Efficiency is compared from the perspective of throughput, and the experimental results, as shown in Figure 8, indicate that the BSSA algorithm has an average throughput of 49.69, which is 28.31% and 62.57% higher than CSSA-DE and LBMPSO, respectively. This demonstrates that the BSSA algorithm achieves better throughput compared to the other algorithms.

5. Conclusion and future work

In this paper, we propose the BSSA algorithm as a multi-objective optimization scheduling method for cloud computing. This algorithm combines the Sparrow Search Algorithm (SSA) with the Backtracking Search Algorithm (BSA), using BSA as a local search strategy to enhance the exploration capability of SSA and improve its convergence speed and solution quality. To validate the improved performance of the proposed algorithm, we conducted experiments on eight optimization benchmark functions and compared them with other algorithms. The experimental results conclude that the proposed algorithm achieves better performance than traditional algorithms.

Additionally, we conducted two series of experiments using the CloudSim toolkit on synthetic and real trace datasets. We measured the Makespan, resource utilization, and throughput of each algorithm and compared them with SSA, PSO, CSSA-DE, and LMBPSO algorithms. The experimental results demonstrate that the proposed BSSA algorithm outperforms the compared algorithms in terms of execution time, resource utilization, and throughput, enabling better resource management and task scheduling in cloud environments.

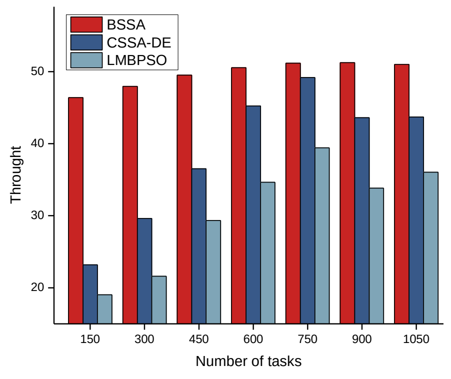


Fig. 6. Makespan time using the random dataset.

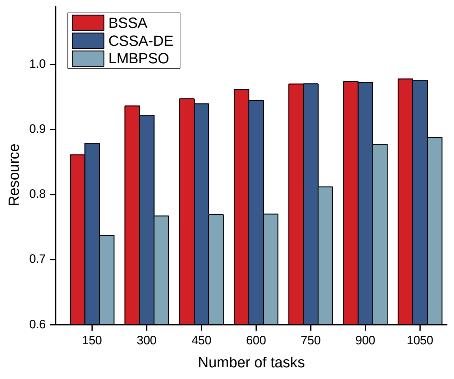


Fig. 7. Resource Utilization using the random dataset.

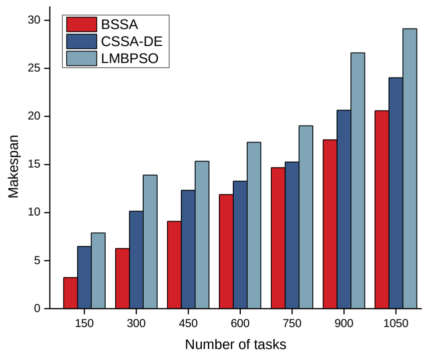


Fig. 8. Throughput using the random dataset.

In the future, we plan to apply the BSSA algorithm to edge computing scenarios and combine it with machine learning algorithms. Moreover, the BSSA algorithm can be extended to consider other optimization objectives in cloud computing environments, such as peak demand, reliability, security, etc., to further enhance its applicability and practicality.

References

[1] Prem Jacob T, Pradeep K. A multi-objective optimal task scheduling in cloud environment using cuckoo particle swarm optimization[J]. Wireless Personal Communications, 2019, 109: 315-331.

[2] Salahudeen A, Junaidu S B, Ayeni A K. An Improved Ant Colony Optimization Algorithm for Scheduling in Cloud Computing Environment[J]. Recent Trends in Cloud Computing and Web Engineering, 2021, 3(2).

[3] Shukri S E, Al-Sayyed R, Hudaib A, et al. Enhanced multi-verse optimizer for task scheduling in cloud computing environments[J]. Expert Systems with Applications, 2021, 168: 114230.

[4] Juarez F, Ejarque J, Badia R M. Dynamic energy-aware scheduling for parallel task-based application in cloud computing[J]. Future Generation Computer Systems, 2018, 78: 257-271.

[5] Houssein E H, Saad M R, Hashim F A, et al. Lévy flight distribution: A new metaheuristic algorithm for solving engineering optimization problems[J]. Engineering Applications of Artificial Intelligence, 2020, 94: 103731.

[6] Xue J, Shen B. A novel swarm intelligence optimization approach: sparrow search algorithm[J]. Systems Science & Control Engineering, 2020, 8(1): 22-34.

[7] Zaman H R R, Gharehchopogh F S. An improved particle swarm optimization with backtracking search optimization algorithm for solving continuous optimization problems[J]. Engineering with Computers, 2022, 38(Suppl 4): 2797-2831.

[8] Chen D, Ge Y, Wan Y, et al. Poplar optimization algorithm: A new meta-heuristic optimization technique for numerical optimization and image segmentation[J]. Expert Systems with Applications, 2022, 200: 117118.

[9] Khaleel M I. Efficient job scheduling paradigm based on hybrid sparrow search algorithm and differential evolution optimization for heterogeneous cloud computing platforms[J]. Internet of Things, 2023: 100697.

[10] Ali A, Iqbal M M, Jamil H, et al. An efficient dynamic-decision-based task scheduler for task offloading optimization and energy management in mobile cloud computing[J]. Sensors, 2021, 21(13): 4527.

[11] Chen X, Cheng L, Liu C, et al. A WOA-based optimization approach for task scheduling in cloud computing systems[J]. IEEE Systems Journal, 2020, 14(3): 3117-3128.

[12] Guo X. Multi-objective task scheduling optimization in cloud computing based on fuzzy self-defense algorithm[J]. Alexandria Engineering Journal, 2021, 60(6): 5603-5609.

[13] Abd Elaziz M, Xiong S, Jayasena K P N, et al. Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution[J]. Knowledge-Based Systems, 2019, 169: 39-52.

[14] Abd Elaziz M, Attiya I. An improved Henry gas solubility optimization algorithm for task scheduling in cloud computing[J]. Artificial Intelligence Review, 2021, 54: 3599-3637.

[15] Panda S K, Nanda S S, Bhoi S K. A pair-based task scheduling algorithm for cloud computing environment[J]. Journal of King Saud University-Computer and Information Sciences, 2022, 34(1): 1434-1445.

[16] Abualigah L, Diabat A. A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments[J]. Cluster Computing, 2021, 24: 205-223.

[17] Nabi S, Ahmad M, Ibrahim M, et al. AdPSO: adaptive PSO-based task scheduling approach for cloud computing[J]. Sensors, 2022, 22(3): 920.

[18] Stan R G, Băjenaru L, Negru C, et al. Evaluation of Task Scheduling Algorithms in Heterogeneous Computing Environments[J]. Sensors, 2021, 21(17): 5906.

[19] Zhang Y. Backtracking search algorithm with specular reflection learning for global optimization[J]. Knowledge-Based Systems, 2021, 212: 106546.

[20] Storn R, Price K. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces[J]. Journal of global optimization, 1997, 11: 341-359.

[21] Hongwei L I, Jianyong L I U, Liang C, et al. Chaos-enhanced moth-flame optimization algorithm for global optimization[J]. Journal of Systems Engineering and Electronics, 2019, 30(6): 1144-1159.

[22] Amirsadri S, Mousavirad S J, Ebrahimpour-Komleh H. A Levy flight-based grey wolf optimizer combined with back-propagation algorithm for neural network training[J]. Neural Computing and Applications, 2018, 30: 3707-3720.

[23] Ebadifard F, Babamir S M. A PSO-based task scheduling algorithm improved using a load-balancing technique for the cloud computing environment[J]. Concurrency and Computation: Practice and Experience, 2018, 30(12): e4368.

[24] Hussain A, Aleem M. GoCJ: Google cloud jobs dataset for distributed and cloud computing infrastructures[J]. Data, 2018, 3(4): 38.

[25] Pradhan A, Bisoy S K. A novel load balancing technique for cloud computing platform based on PSO[J]. Journal of King Saud University-Computer and Information Sciences, 2022, 34(7): 3988-3995.