Scientific Research Publishing

Journal of Computer and Communications, 2023, 11, 88-117
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Multi-Strategy-Driven Salp Swarm Algorithm for Global Optimization

Zhiwei Gao, Bo Wang*

College of Science, Shenyang University of Technology, Shenyang, China
Email: gzw-2401112886@smail.sut.edu.cn, *5692188@qq.com

How to cite this paper: Gao, Z.W. and Wang, B. (2023) Multi-Strategy-Driven Salp Swarm Algorithm for Global Optimization. *Journal of Computer and Communications*, **11**, 88-117.
https://doi.org/10.4236/jcc.2023.117007

Received: June 2, 2023
Accepted: July 25, 2023
Published: July 28, 2023

Copyright © 2023 by author(s) and Scientific Research Publishing Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/

Open Access

Abstract

In response to the shortcomings of the Salp Swarm Algorithm (SSA) such as low convergence accuracy and slow convergence speed, a Multi-Strategy-Driven Salp Swarm Algorithm (MSD-SSA) was proposed. First, food sources or random leaders were associated with the current bottle sea squirt at the beginning of the iteration, to which Levy flight random walk and crossover operators with small probability were added to improve the global search and ability to jump out of local optimum. Secondly, the position mean of the leader was used to establish a link with the followers, which effectively avoided the blind following of the followers and greatly improved the convergence speed of the algorithm. Finally, Brownian motion stochastic steps were introduced to improve the convergence accuracy of populations near food sources. The improved method switched under changes in the adaptive parameters, balancing the exploration and development of SSA. In the simulation experiments, the performance of the algorithm was examined using SSA and MSD-SSA on the commonly used CEC benchmark test functions and CEC2017-constrained optimization problems, and the effectiveness of MSD-SSA was verified by solving three real engineering problems. The results showed that MSD-SSA improved the convergence speed and convergence accuracy of the algorithm, and achieved good results in practical engineering problems.

Keywords

Salp Swarm Algorithm (SSA), Levy Flight, Brownian Motion, Location Update, Simulation Experiment

1. Introduction

With the continuous development of human cognition and society, the complexity

of various application problems and scientific computations has increased, and the own drawbacks of traditional optimization computation methods have revealed themselves to be unable to meet people's needs in a reasonable time. In recent years, meta-heuristic algorithms have received much attention and have been applied to many fields because of their operational flexibility, ease of implementation and gradient-free mechanism. Among them, the evolutionary algorithm includes Genetic Algorithm (GA) [1], Differential Evolution (DE) [2], etc. The swarm intelligence techniques mainly include Particle Swarm Optimization (PSO) [3], Ant Colony Optimization (ACO) [4], Artificial Bee Colony Algorithm (ABC) [5], Cuckoo Search (CS) [6], etc. The recently proposed swarm intelligence algorithm includes Polar Bear Optimization (PBO) [7], Grey Wolf Algorithm (GWO) [8], Whale Optimization Algorithm (WOA) [9], Salp Swarm Algorithm (SSA) [10], etc. The Salp Swarm Algorithm was proposed by Mirjalili *et al.* The algorithm has adaptive parameters that facilitate both global searches in the early stages of the algorithm and local development in the later stages of the algorithm. Similar to other swarm intelligence algorithms, it suffers from the disadvantages of being prone to local optimality and low convergence accuracy in late iterations.

Since the Salp Swarm Algorithm was proposed in 2017, many scholars at home and abroad have improved the performance of the Salp Swarm Algorithm by addressing its shortcomings. Thawkar [11] proposed a hybrid model using Teaching-Learning-Based Optimization and Salp Swarm Algorithm (TLBO-SSA), and applied it to the diagnosis of breast cancer, and achieved good results. Neggz *et al.* [12] introduced the sine cosine algorithm into the Salp Swarm Algorithm to obtain an enhanced algorithm (ISSAFD), which improved the exploration effect in the global search stage, enhanced the diversity of the population, avoided falling into local optimization, and balanced the exploration and development of the algorithm. Zhang *et al.* [13] proposed a multi-strategy Enhanced Salp Swarm Algorithm (ESSA), which uses orthogonal learning to generate opposite solutions, expands the diversity of the population, uses quadratic interpolation to improve the local search ability of the algorithm, and uses test functions to test the accuracy of the improved algorithm. Heidari *et al.* [14] proposed the Chaotic Salp Swarm Algorithm (CDESSA), chaotic initialization was introduced in the initial stage of the algorithm to expand the diversity of the population, and differential evolution was used to prevent premature convergence. Chen *et al.* [15] used a weighted center of gravity for the leader and adaptive weight for followers, balancing the development and exploration of the algorithm. Yang *et al.* [16] proposed a multi-strategy fusion Salp Swarm Algorithm (ISSA), which uses the mid-vertical theorem to change the position update mode of the follower and introduces the perturbation mechanism of the mid-vertical convergence strategy to improve the ability of the algorithm to jump out of the local optimum. Liu *et al.* [17] proposed a Differential Evolution Parasitic Salp Swarm Algorithm (PDESSA), which introduced the position information of the previous generation of leaders, strengthened the global search, introduced adaptive inertia weight,

balanced the exploration and exploitation of the algorithm, and finally introduced a dual-population mechanism with evolution and parasitism strategies, which increased the diversity of the population and improved the ability of the algorithm to jump out of local extremum. Zhang *et al*. [18] introduced the Levy flight strategy into the algorithm, and updated the follower by comparing the fitness value of the follower, which enhanced the global search ability and convergence speed of the algorithm. Xie *et al*. [19] proposed a New Salp Swarm Algorithm (NSSA), introduced the idea of following the head wolf in the gray wolf optimization algorithm to the update method of leader, and carried out experiments in 23 benchmark test functions to apply it to image matching.

All of the above-improved versions improve the performance of the original algorithm in finding the optimum to a certain extent. In order to make the algorithm capable of solving more complex optimization problems, it still needs to be improved. In this paper, we propose a Multi-Strategy-Driven Salp Swarm Algorithm (MSD-SSA). Firstly, in the first half of the algorithm, the leader's position is updated using Levy flight with small probability, Levy flight is a random wandering with larger probability of large span steps, which can expand the global search range and jump out of local extremes more effectively. Secondly, a crossover operation is performed on the leader and the better leader is retained based on a greedy strategy to improve the convergence speed of the algorithm. Again, in the second half of the algorithm, we introduce Brownian motion random step is introduced in the second half of the algorithm, and Brownian motion is used to search deeply in the latest iteration to improve the convergence accuracy of the algorithm. Finally, the follower update method is different from the original algorithm, and the average position of the leader is used to update the followers, which strengthens the connection between the followers and the leader and makes the convergence speed faster. The performance of MSD-SSA is verified by experimenting with the commonly used benchmark test functions and CEC2017 [20] test set to solve the optimal values and three engineering problems, and comparing with the traditional Salp Swarm Algorithm.

2. Salp Swarm Algorithm (SSA)

The inspiration for the Salp Swarm Algorithm [10] comes from the group movement and predatory behavior of the marine organism salp. The salp is a marine creature with a translucent barrel-shaped body that resembles a jellyfish. Based on the behavior of this creature, the algorithm for SSA was simulated. In SSA, the problem to be solved is to find the largest food source. The fitness function of the algorithm depends on the quality of the food source.

Depending on the position of the individuals in the chain structure, all individuals can be divided into two categories: leaders and followers. The leader guides all individuals to form a herd chain and move towards the food source to find a better food source for the population, and the followers follow the pre-

vious individual and are indirectly guided by the leader. The mathematical model of these two stages is as follows:

In the D-dimensional search space, N individuals of salp are randomly generated. The population of salp is shown in the $N \times D$ matrix in Equation (1)

$$X = \begin{bmatrix} X^1 \\ X^2 \\ \vdots \\ X^N \end{bmatrix} = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_D^1 \\ x_1^2 & x_2^2 & \cdots & x_D^2 \\ \vdots & \vdots & & \vdots \\ x_1^N & x_2^N & \cdots & x_D^N \end{bmatrix}. \tag{1}$$

Among them, X shows the population of salp, X^i shows the i-th individual in the population, and x_j^i shows the j-th attribute of the i-th individual in the population.

The position update of leader is related to the current global optimal food source location, so after generating the population, find the current global optimal food source location F. In the population, leader has leadership functions, and their positions are updated according to Equation (2)

$$x_j^i = \begin{cases} F_j + c_1\left(r_1\left(ub_j - lb_j\right) + lb_j\right), & r_2 \geq \dfrac{1}{2} \\ F_j - c_1\left(r_1\left(ub_j - lb_j\right) + lb_j\right), & r_2 < \dfrac{1}{2} \end{cases} \tag{2}$$

where, x_j^i shows the new position of the i-th leader in the j-th dimension, F_j is the j-th dimension of the best food source, ub_j and lb_j show the upper and lower bounds of the individual in the j-dimensional space, respectively, r_1 and r_2 are random numbers subject to uniform distribution on $[0,1]$. The coefficient c_1 is an important convergence factor of SSA.

c_1 is mainly responsible for the exploration and development of the algorithm, which is calculated by Equation (3)

$$c_1 = 2\exp\left(-\left(\frac{4t}{T_{\max}}\right)^2\right), \tag{3}$$

where, t is the current number of iterations, and $T_{\max}$ is the maximum number of iterations. The value of c_1 gradually decreases from 2, and its variation curve is shown in **Figure 1**. In the early stage of algorithm iteration, c_1 changes quickly and has a large step size, which is conducive to global search. At the later stage of the iteration, c_1 tends to be stable and the step size is small, which is conducive to fine search and converges to the global best food source.

Followers use Newton's law of motion to update the position, give the initial velocity, acceleration and initial position to the follower, and obtain the position update formula as shown in Equation (4)

$$x_j^i = v_0 t + \frac{1}{2}at^2, \tag{4}$$

Assuming its initial velocity $v_0 = 0$, $a = \left(v_{\text{final}} - v_0\right)/t$, Equation (5) is the position update of followers.

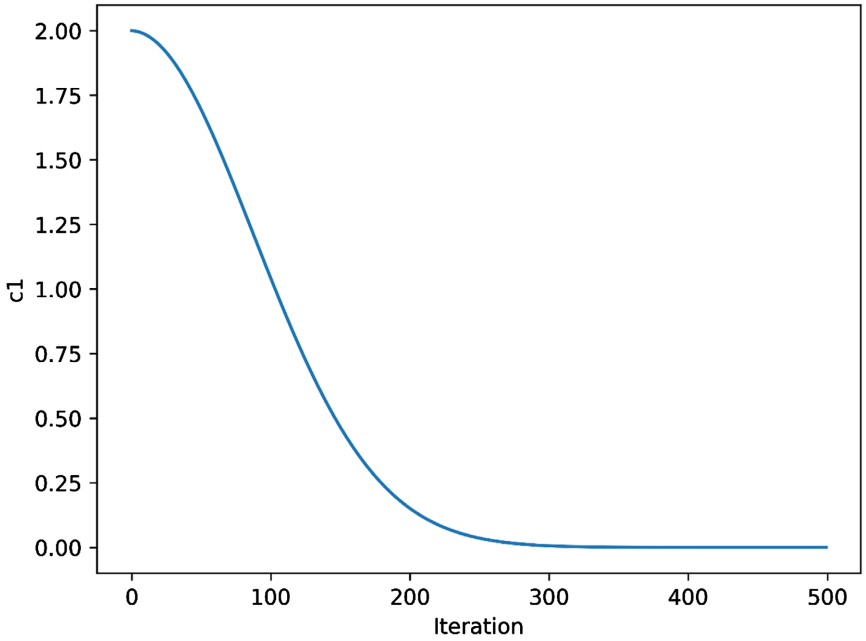


Figure 1. Change curve of c_1 iteration for 500 times.

$$x_j^i(t) = \frac{1}{2}\left(x_j^{i-1}(t-1) + x_j^i(t-1)\right), \tag{5}$$

where, $i \geq 2$. x_j^i shows the new position updated by the i-th follower on the j-th dimension. According to Equation (5), the position update of the follower is related to the position of the previous individual.

The process of SSA is as follows:

Step 1. Set parameters such as population size, maximum number of iterations, upper and lower boundaries.

Step 2. Find individual fitness value and get food source, leader and followers.

Step 3. Judge whether the maximum iteration is reached. If so, output the optimal value position and optimal value, otherwise proceed to the next step.

Step 4. Equation (2) updates leader position and amend boundary, Equation (5) updates follower position and amend boundary.

Step 5. Calculate the fitness value of the individual after the location update.

Step 6. Increase the number of iterations by 1, return to Step 3.

3. Multi-Strategy-Driven Salp Swarm Algorithm (MSD-SSA)

3.1. Random Step of Levy Flight and Brownian Motion

In the standard SSA, the leader only searches according to the location of the food source, and the update mode of the follower is only related to its adjacent Levy flight [21], proposed by French Mathematician Paul Pierre Lévy [21], is a heavy-tailed distribution with a high probability of producing a large stride. Most animals follow this rule in their foraging behavior. The accidental long-distance step of Levy flight helps to expand the search range and jump out of the limit of local optimization, more short distance steps can carefully search the surround-

ing area and improve the local exploitation ability. The trajectory of Levy's flight is shown in Figure 2.

The probability density function of Levy flight obeys the Levy distribution, which can be expressed by Equation (6)

$$\text{Levy}(s)=\frac{1}{\pi}\int_0^{\infty}\exp\left(-\beta|k|^{\lambda}\right)\cos(ks)\,\mathrm{d}k. \tag{6}$$

According to Mantegna method [22], the random step of Levy flight can be expressed as follows

$$l=\frac{u}{|v|^{1/\beta}}. \tag{7}$$

where, u and v are Gaussian distributions that satisfy the following conditions

$$u\sim N\left(0,\delta_u^2\right),v\sim N\left(0,\delta_v^2\right), \tag{8}$$

$$\delta_u=\left(\frac{\Gamma(1+\beta)\cdot\sin\left(\frac{\beta}{2}\pi\right)}{\Gamma\left(\frac{1+\beta}{2}\right)\cdot\beta\cdot 2^{\frac{\beta-1}{2}}}\right)^{1/\beta}, \tag{9}$$

$$\delta_v=1, \tag{10}$$

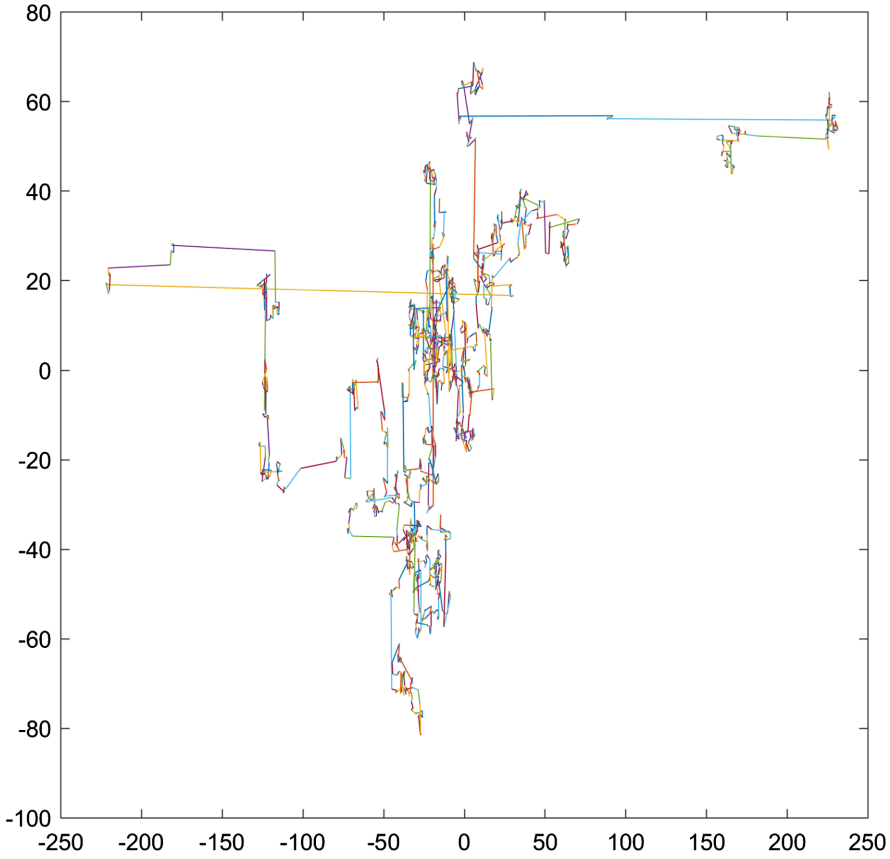


Figure 2. Movement trajectory of Levy flight (a random walk graph generated by Levy flight random step within a certain range and random direction walking).

the value of β is generally 1.5, and Γ is a Gamma function.

Brownian motion is the irregular motion made by a suspended particle, a random wandering process without rules. Because the standard Brownian motion step change image is similar to a Gaussian distribution with mean 0 and variance 1, the Brownian motion random number is generated from Equation (11)

$$\text{Brown} \sim N\left(0,\delta^2\right). \tag{11}$$

3.2. Crossover Operator

The crossover operator helped to increase the diversity of the population, thereby increasing the chance for the population to break out of local optimum food sources.

Different random integers were generated, each corresponding to the respective dimension of an individual, sorted based on their fitness values, and the positions of the corresponding dimensions were exchanged in sequence. The adjacent individuals underwent the crossover operator to generate the crossover individuals, and a greedy strategy was used to retain individuals with better fitness values for the next iteration. c random numbers satisfy

$$c = kD, 0 < s_i \le D, s = \left\{s_1, s_2, \cdots, s_c\right\}, \tag{12}$$

where, the random integers in s are different, and k is the proportion coefficient of the crossover operator.

3.3. Mean Position of the Leader

In standard SSA, the position update of a follower is only related to the previous individual, which is blind and slows down the convergence of the overall algorithm. If the previous individual falls into a local optimum, then all the followers are unable to jump out of the local optimum.

Consider the position relationship between the follower and the average leader, and let the follower follow the average leader closely, with the following update method

$$\overline{x}_j = \frac{\sum_{i=0}^{N_l} x_j^i}{N_l}, \tag{13}$$

where, $\overline{x}_j$ is the average position of all leaders in the j-th dimension, and N_l is the number of leaders.

3.4. Improving Salp Swarm Algorithm (ISSA)

In the standard SSA, the leader only searches according to the location of the food source, and the update mode of the follower is only related to its adjacent individual location. The global search or local search is determined according to c_1 value. If the food source is the local optimum, the leader will fall into it, and the follower will not jump out of the local optimum, so that the SSA cannot jump out of the local optimum. Therefore, in the early stage of the iteration, consider

the location of random leader or food source to establish contact with the current individual to update the leader, and a small probability of Levy flight [21] random step size should be added to help jump out of the local optimal during the iteration process.

A random leader or food source establishes relationship with the current individual location to obtain the location update formula of leader

$$x_j^i(t)=\begin{cases} x_j^i(t-1)+c_1\left(F_j-x_j^i(t-1)\right), & r_2\geq\dfrac{1}{2} \\ x_j^i(t-1)+c_1\left(x_j^i(t-1)-x_j^{rand}\right), & r_2<\dfrac{1}{2} \end{cases} \tag{14}$$

among them, x_j^{rand} is the position of the randomly selected leader in the j-th dimension. In this way, the influence of the food source on the current individual and the influence of other individuals in the population are considered, the exploration and exploitation process of the algorithm is also considered.

Adding Levy flight random number with small probability is beneficial to better jump out of local optimum. When $P<P_0$, calculate the Levy flight [21] random number through Equation (7) to Equation (10), and obtain the leader position update formula to add Levy flight random number

$$x_j^i(t)=\begin{cases} \left(x_j^i(t-1)+c_1\left(F_j-x_j^i(t-1)\right)\right)\cdot l, & r_2\geq\dfrac{1}{2} \\ \left(x_j^i(t-1)+c_1\left(x_j^i(t-1)-x_j^{rand}\right)\right)\cdot l, & r_2<\dfrac{1}{2} \end{cases} \tag{15}$$

where, P_0 is the probability of occurrence, l is the Levy flight [21] random number.

Later, in the iteration, as the value of c_1 continues to decrease, SSA tends to fine tune the search, and to improve the search accuracy, a random step of Brownian motion is used. Equation (11) is the position update method of leader adopting Brownian motion

$$x_j^i=\begin{cases} \left(F_j+c_1\left(r_1\left(ub_j-lb_j\right)+lb_j\right)\right)\cdot\text{Brown}, & r_2\geq\dfrac{1}{2} \\ \left(F_j-c_1\left(r_1\left(ub_j-lb_j\right)+lb_j\right)\right)\cdot\text{Brown}, & r_2<\dfrac{1}{2} \end{cases} \tag{16}$$

where, Brown is the random step of Brownian motion.

The crossover operator can generate new individuals, expand the diversity of the population, improve the search efficiency of the population, and better jump out of the local optimum. Generate random dimensions according to Equation (12), and apply the crossover operator to the leader.

According to Equation (13), the position updating method for the follower is derived.

$$x_j^i(t)=\frac{1}{2}\left(\bar{x}_j+x_j^i(t-1)\right), \tag{17}$$

where, $\bar{x}_j$ is the mean position of the leaders calculated by Equation (13).

Followers are led purposefully, and the population is able to move faster to-

wards the optimal food source, allowing the algorithm to converge more quickly.

Each iteration of the algorithm uses a greedy strategy to select the optimal fitness-valued individuals.

The flowchart of MSD-SSA is shown in Figure 3.

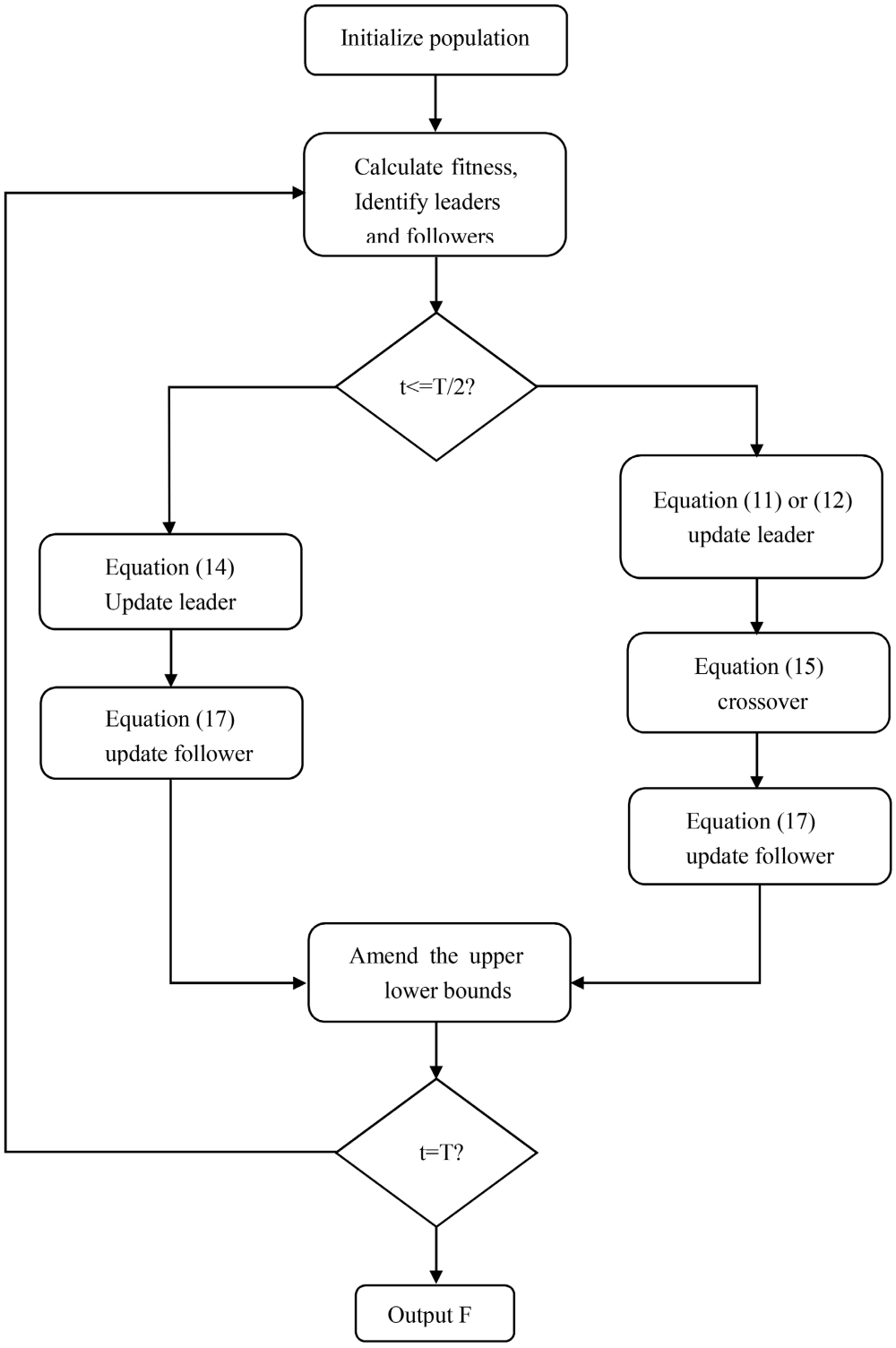


Figure 3. The flowchart of MSD-SSA.

The pseudo code of the MSD-SSA is described as follows:

Pseudo (MSD-SSA)

Initialize the salp population $x_i\left(i=1,2,\cdots,N\right)$ considering upper and lower
Calculate the fitness of each salp
While (end condition is not satisfied)
    F = the best salp
    Equation (3) Calculate c_1
    if $0 \le t < T/2$
        for each salp (x_i)
            if (leaders)
                Equations (7)-(10) calculates the random step of Levy flight
                Update the position of the search agents by Equation (14) or Equation (15)
                Equation (12) leader crossover operator operation
                Greedy algorithm selects better leaders
            else (followers)
                Update the position of the search agents by Equation (17)
        end
    else $T/2 \le t < T$
        for each salp (x_i)
            if (leaders)
                Calculate the random step of Brownian motion
                Update the position of the search agents by Equation (16)
            else (followers)
                Update the position of the search agents by Equation (17)
        end
    end
Amend the salp based on the upper and lower bounds of variables
end
return F

3.5. Time Complexity Analysis

Time complexity is an important indicator of the amount of work required to run the algorithm and to evaluate the time consumption of the algorithm. The time complexity is usually denoted by O. The MSD-SSA algorithm consists of three parts: population initialization, updating the leader position and updating the follower position. Assume that the iteration number of the algorithm is T, the population size is N, and the dimension is D.

1) Initialize the population, which needs to be run ND time.

2) To calculate the individual fitness value and select the best individual as food, it needs to run $\frac{N\left(N-1\right)}{2}$ times.

3) The algorithm parameters are updated and need to be run 4 times.

4) Leaders need to run D times to update in space.

5) Followers update $(N-1)D$ times in space.

6) Levy flight update needs to run $\frac{1}{2}ND$ times.

7) Brownian motion update needs to run $\frac{1}{2}ND$ times.

8) It takes ND time to find out the food source from the population and export it.

Each of the above operation units goes through T iteration, so the total time complexity of MSD-SSA is Equation (18)

$$O(\text{MSD-SSA}) = T\left[ND + \frac{N(N-1)}{2} + 4 + D + (N-1)D\right]. \tag{18}$$

4. Simulation Experiments and Result Discussion

To verify the performance of MSD-SSA, 10 commonly used benchmark test functions [23] and some CEC2017 constraint planning problems [20], as well as three real engineering problems, were selected for experiments, mainly to verify the merit-seeking capability and the convergence speed of MSD-SSA.

The operating device is the Windows 10 operating system, the CPU is Intel Core i5-1135G7@2.4 GHz and 16 G running memory. The editing language is python, and the experimental platform is PyCharm 2022.3.2.

4.1. Benchmarking Functions

The 10 benchmarks test functions [23] are shown in **Table 1**. The benchmark test functions used are all 30-dimensional, the population of the algorithm has 30 individuals and the algorithm is iterated 500 times. Comparison is made using (Wolf Pack Algorithm) WPA, SSA and MSD-SSA. The parameter settings for WPA are $\alpha = 4, \beta = 6, \omega = 30$. The number of leaders and followers is half of the population respectively, the parameter value in MSD-SSA is $P_0 = 0.2, k = 0.2$. The test functions in **Table 1** were solved using WPA, SSA and MSD-SSA respectively, and the python was used to make 30 independent experiments and compare the mean, standard deviation and convergence speed (time) of the three algorithms to avoid the bias caused by the randomness of the algorithms and to ensure the reasonableness of the algorithms. The experimental results are shown in **Table 2** and the resulting convergence curves are plotted in **Figure 4**.

4.2. CEC2017 Constrained Optimization Problems

The CEC2017 test functions [20] are shown in **Table 3**. The test functions are in 10 and 30 dimensions, respectively. The population of the algorithm has 100 individuals, the number of leaders and followers is half of the population respectively, and the algorithm is iterated 1000 times independently. The parameter value in MSD-SSA is $P_0 = 0.2, k = 0.2$. The test functions in **Table 3** were solved using MSD-SSA and SSA respectively, and the python was used to make 30

Table 1. Description of benchmark function.

Function	Range	$f_{\min}$
$f_1(x)=\sum_{i=1}^{n}x_i^2$	$[-100,100]$	0
$f_2(x)=\sum_{i=1}^{n}\lvert x_i\rvert+\prod_{i=1}^{n}\lvert x_i\rvert$	$[-10,10]$	0
$f_3(x)=\sum_{i=1}^{n}\left(\sum_{j=1}^{i}x_j\right)^2$	$[-100,100]$	0
$f_4(x)=\max_i\left\{\lvert x_i\rvert,1\le i\le n\right\}$	$[-100,100]$	0
$f_5(x)=\sum_{i=1}^{n-1}\left[100\left(x_{i+1}-x_i^2\right)^2+\left(x_i-1\right)^2\right]$	$[-30,30]$	0
$f_6(x)=\sum_{i=1}^{n}ix_i^4+random[0,1)$	$[-1.28,1.28]$	0
$f_7(x)=\sum_{i=1}^{n}\left[x_i^2-10\cos(2\pi x_i)+10\right]$	$[-5.12,5.12]$	0
$f_8(x)=-20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right)-\exp\left(\frac{1}{n}\cos(2\pi x_i)\right)+20+e$	$[-32,32]$	0
$f_9(x)=\frac{1}{4000}\sum_{i=1}^{n}x_i^2-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	$[-600,600]$	0
$f_{10}(x)=\frac{\pi}{n}\left\{10\sin^2(\pi y_i)+\sum_{i=1}^{n-1}(y_i-1)^2\left[1+\sin^2(\pi y_{i+1})\right]+(y_n-1)^2\right\}+\sum_{i=1}^{n}u(x_i,10,100,4),$ $y_i=1+\frac{1}{4}(x_i+1)$ $u(x_i,a,k,m)=\begin{cases}k(x_i-a)^m, & x_i>a\\ 0, & -a\le x_i\le a\\ k(-x_i-a)^m, & x_i<-a\end{cases}$	$[-50,50]$	0

Table 2. Performance comparison of WPA, SSA and MSD-SSA on test functions.

Function	Stats	MSD-SSA	SSA	WPA
f_1	Mean	8.98E−11	9.42E−09	4.28E+00
	Std	6.65E−12	2.38E−08	7.27E−12
	Time(s)	5.704768753	6.055757221	4.325863247
f_2	Mean	4.19E−06	0.35004655	1.2226230412
	Std	3.30E−07	0.00191453	1.52E−05
	Time(s)	5.858462628	6.122387815	5.813986003
f_3	Mean	2.94E−10	53.75321536	68.26151173
	Std	2.30E−10	200.9015373	0.0

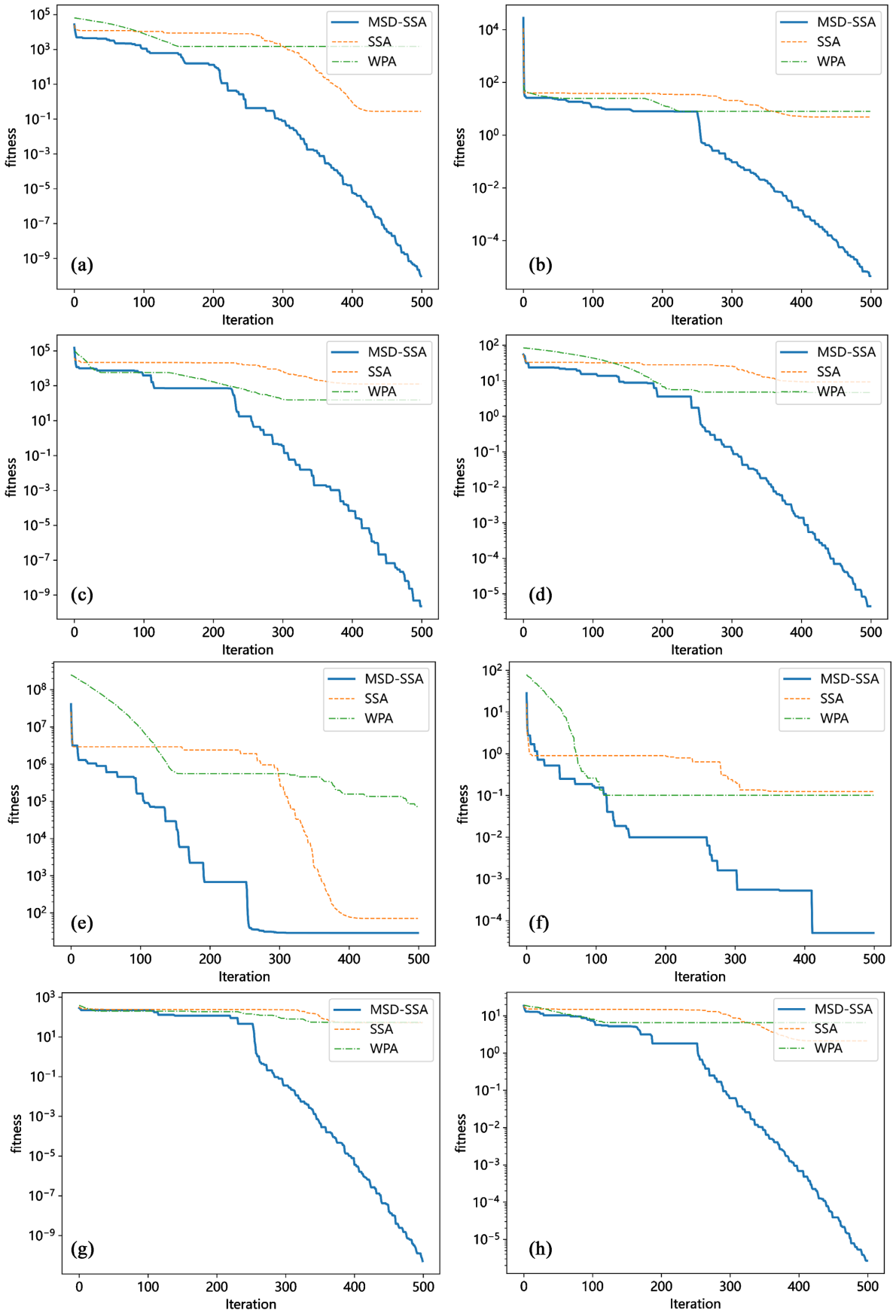
Continued

	Time(s)	7.408410994	7.686855181	5.0839362025
f_4	Mean	3.47E−06	11.45540334	8.71E+10
	Std	3.09E−07	0.147363826	2.84E−14
	Time(s)	5.779169146	6.071856912	3.682288539
f_5	Mean	28.12139887	390.9104529	486.1950995
	Std	0.165077021	19.94642673	0.0
	Time(s)	6.140934952	6.516275398	6.908285319
f_6	Mean	4.45E−05	0.033427049	0.001022907
	Std	0.000118094	0.049622245	1.42E−14
	Time(s)	5.421882804	5.717415269	4.837871611
f_7	Mean	3.46E−11	41.78822494	26.68294181
	Std	5.57E−12	6.15E−09	5.68E−14
	Time(s)	6.163218141	6.453282237	2.911670494
f_8	Mean	2.07E−06	2.495401167	20.79011457
	Std	2.84E−07	9.90E−10	3.55E−15
	Time(s)	7.183958522	7.221007339	7.085961198
f_9	Mean	1.18E−10	0.010155491	686.3675368
	Std	1.52E−11	0.001599289	0.0
	Time(s)	7.029453381	7.650319187	6.498791745
f_{10}	Mean	0.379057981	4.053250733	3.487683579
	Std	0.097481408	0.421909641	1.19E−07
	Time(s)	7.203487563	7.375670958	17.2234921

independent experiments and compare the mean, standard deviation and convergence speed (time) of the two algorithms to avoid the bias caused by the randomness of the algorithms and to ensure the reasonableness of the algorithms. Table 4 and Table 5 show the experimental results in 10 and 30 dimensions, respectively, and the obtained convergence curves are shown in Figure 5 and Figure 6, respectively.

4.3. Result Discussion

For benchmarks test functions, the experimental results in Table 2 show that MSD-SSA has significant advantages over SSA in optimizing most functions. In terms of convergence precision, the convergence accuracy of MSD-SSA for $f_1, f_2, f_3, f_4, f_6, f_7, f_8, f_9$ is significantly better than that of SSA. For f_5 and f_{10}, the final convergence is not optimal, but the convergence results are greatly improved compared with SSA. In terms of algorithm robustness, except f_8, the stability is better than SSA. Although the stability of f_8 is slightly weaker than


MSD-SSA
SSA
WPA
fitness
Iteration
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)

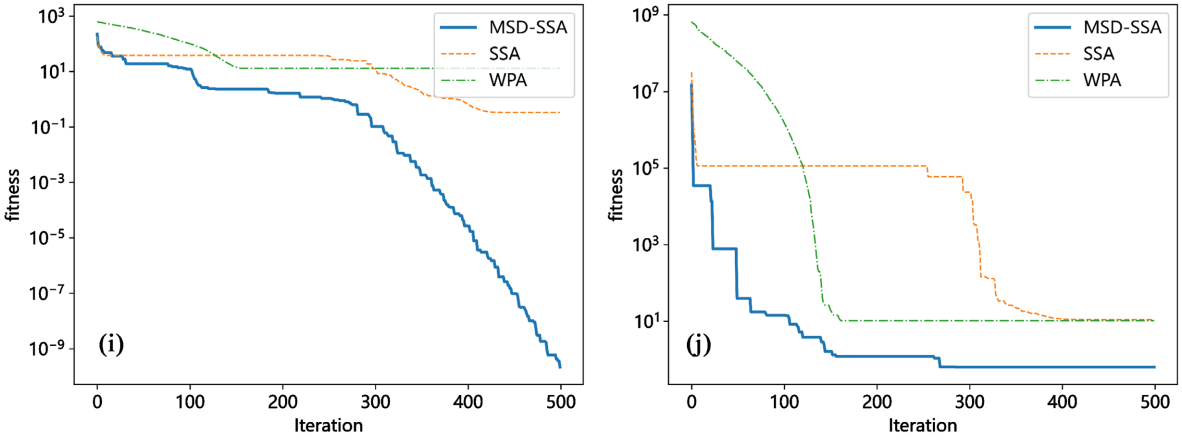


Figure 4. Corresponds to the convergence curve of test function f_1 - f_{10} respectively.

Table 3. CEC2017 description of benchmark function.

(a)

Problem	Function
CEC01	$\min \quad f(x)=\sum_{i=1}^{D}\left(\sum_{j=1}^{i}x_j\right)^2$ $\text{s.t.} \quad g(x)=\sum_{i=1}^{D}\left[x_i^2-5000\cos(0.1\pi x_i)-4000\right]\le 0$
CEC02	$\min \quad f(x)=\sum_{i=1}^{D}\left(\sum_{j=1}^{i}Mx_j\right)^2$ $\text{s.t.} \quad g(x)=\sum_{i=1}^{D}\left[(Mx_i)^2-5000\cos(0.1\pi Mx_i)-4000\right]\le 0$
CEC03	$\min \quad f(x)=\sum_{i=1}^{D}\left[x_i^2-10\cos(2\pi x_i)+10\right]$ $\text{s.t.} \quad g_1(x)=-\sum_{i=1}^{D}x_i\sin(2x_i)\le 0,\ g_2(x)=\sum_{i=1}^{D}x_i \quad (2x_i)\le 0$
CEC04	$\min \quad f(x)=\sum_{i=1}^{D-1}\left[100\left(x_i^2-x_{i+1}\right)^2+\left(x_i^2-1\right)^2\right]$ $\text{s.t.} \quad g_1(x)=\sum_{i=1}^{D}\left[(M_1x_i)^2-50\cos(2\pi M_1x_i)-40\right]\le 0$ $g_2(x)=\sum_{i=1}^{D}\left[(M_2x_i)^2-50\cos(2\pi M_2x_i)-40\right]\le 0$
CEC05	$\min \quad f(x)=\sum_{i=1}^{D}\left[x_i^2-10\cos(2\pi x_i)+10\right]$ $\text{s.t.} \quad h_1(x)=-\sum_{i=1}^{D}x_i\sin(x_i)=0$ $h_2(x)=\sum_{i=1}^{D}x_i\sin(\pi x_i)=0$ $h_3(x)=-\sum_{i=1}^{D}x_i\cos(x_i)=0$ $h_4(x)=\sum_{i=1}^{D}x_i\cos(\pi x_i)=0$ $h_5(x)=\sum_{i=1}^{D}\left(x_i\sin\left(2\sqrt{\lvert x_i\rvert}\right)\right)=0$ $h_6(x)=-\sum_{i=1}^{D}\left(x_i\sin\left(2\sqrt{\lvert x_i\rvert}\right)\right)=0$

Continued

CEC06

$$\begin{aligned}&\min && f(x)=\max(x_i)\, y_l=x_{2l-1}, w_l=x_{2l} \text{ where } l=1,\cdots,D/2\\ &\text{s.t.} && h_1(x)=\sum_{i=1}^{D/2}\left(\sum_{j=1}^{i} y_j\right)^2=0\\ & && h_2(x)=\sum_{i=1}^{D/2}\left(\sum_{j=1}^{i} w_j\right)^2=0\end{aligned}$$

CEC07

$$\begin{aligned}&\min && f(x)=\max(x_i)\, y_l=x_{2l-1}, w_l=x_{2l} \text{ where } l=1,\cdots,D/2\\ &\text{s.t.} && g(x)=\prod_{i=1}^{D/2} w_i\le 0\\ & && h(x)=\sum_{i=1}^{D/2}\left(y_i^2-y_{i+1}\right)^2=0\end{aligned}$$

CEC08

$$\begin{aligned}&\min && f(x)=\max(x_i)\\ &\text{s.t.} && h_1(x)=\sum_{i=1}^{D}\left(\sum_{j=1}^{i} x_j\right)^2=0\\ & && h_2(x)=\sum_{i=1}^{D-1}\left(x_i-x_{i+1}\right)^2=0\end{aligned}$$

CEC09

$$\begin{aligned}&\min && f(x)=\sum_{i=1}^{D} x_i\\ &\text{s.t.} && g(x)=\prod_{i=1}^{D} x_i\le 0\\ & && h(x)=\sum_{i=1}^{D-1}\left(x_i^2-x_{i+1}\right)^2=0\end{aligned}$$

CEC10

$$\begin{aligned}&\min && f(x)=\sum_{i=1}^{D-1}\left(100\left(x_i^2-x_{i+1}\right)^2+\left(x_i-1\right)^2\right)\\ &\text{s.t.} && g_1(x)=\sum_{i=1}^{D}\left(x_i^2-10\cos\left(2\pi x_i\right)+10\right)-100\le 0\\ & && g_2(x)=\sum_{i=1}^{D} x_i-2D\le 0\\ & && g_3(x)=5-\sum_{i=1}^{D} x_i\le 0\end{aligned}$$

CEC11

$$\begin{aligned}&\min && f(x)=\max\left\{\left|x_i\right|, 1\le i\le D\right\}\\ &\text{s.t.} && g(x)=\sum_{i=1}^{D} x_i^2-100D\le 0\\ & && h(x)=\cos f(x)+\sin f(x)=0\end{aligned}$$

CEC12

$$\begin{aligned}&\min && f(x)=\sum_{i=1}^{D}\left|x_i\right|\\ &\text{s.t.} && g(x)=\sum_{i=1}^{D} x_i^2-100D\le 0\\ & && h(x)=\left(\cos f(x)+\sin f(x)\right)^2-\exp\left(\cos f(x)+\sin f(x)\right)-1+\exp(1)=0\end{aligned}$$

CEC13

$$\begin{aligned}&\min && f(x)=\sum_{i=1}^{D}\left(z_i^2-10\cos\left(2\pi z_i\right)+10\right), z_i^2=\begin{cases}x_i^2, \text{if } \left|x_i^2\right|<0.5\\ 0.5round\left(2x_i\right), \text{otherwise}\end{cases}\\ &\text{s.t.} && g_1(x)=1-\sum_{i=1}^{D}\left|x_i\right|\le 0\\ & && g_2(x)=\sum_{i=1}^{D} x_i^2-100D\le 0\\ & && h(x)=\sum_{i=1}^{D} 100\left(x_i^2-x_{i+1}\right)^2+\prod_{i=1}^{D}\sin^2\left(x_i-1\right)\pi=0\end{aligned}$$

Continued

CEC14	min	$f(x)=\sum_{i=1}^{D-1} g(x_i,x_{i+1})+g(x_D,x_1), g(x_i,x_{i+1})=0.5+\frac{\sin^2\left(\sqrt{x_i^2+x_{i+1}^2}\right)-0.5}{\left(1+0.001\sqrt{x_i^2+x_{i+1}^2}\right)^2}$
	s.t.	$g_1(x)=\cos^2\left(\sum_{i=1}^{D} x_i\right)-0.25\cos\left(\sum_{i=1}^{D} x_i\right)-0.125\le 0$
		$g_2(x)=\exp\left(\cos\left(\sum_{i=1}^{D} x_i\right)\right)-\exp(0.25)\le 0$
CEC15	min	$f(x)=\sum_{i=1}^{D}\left[y_i^2-10\cos(2\pi y_i)+10\right], y=Mx$
	s.t.	$g_1(x)=4-\sum_{i=1}^{D}\lvert y_i\rvert\le 0$
		$g_2(x)=\sum_{i=1}^{D} y-4\le 0$
CEC16	min	$f(x)=\sum_{i=1}^{D}\left[100\left(y_i^2-x_{i+1}\right)^2+\left(y_i-1\right)^2\right], y=Mx$
	s.t.	$g_1(x)=\sum_{i=1}^{D}\left(y_i^2-10\cos(2\pi y_i)+10\right)\le 0$
		$g_2(x)=\sum_{i=1}^{D} y_i-2D\le 0$
		$g_2(x)=5-\sum_{i=1}^{D} y_i\le 0$

(b)

Problem	Range	Number of Bindings	
		E	*I*
CEC01	[−100, 100]	0	1
CEC02	[−100, 100]	0	1
CEC03	[−10, 10]	0	2
CEC04	[−10, 10]	0	2
CEC05	[−20, 20]	6	0
CEC06	[−100, 100]	2	0
CEC07	[−10, 10]	2	0
CEC08	[−100, 100]	2	0
CEC09	[−100, 100]	1	1
CEC10	[−100, 100]	0	3
CEC11	[−100, 100]	1	1
CEC12	[−100, 100]	1	1
CEC13	[−100, 100]	1	2
CEC14	[−100, 100]	0	2
CEC15	[−100,100]	0	2
CEC16	[−100,100]	0	3

Table 4. Performance comparison of MSD-SSA and SSA on test functions (10*D*).

Function	Stats	MSD-SSA	SSA
CEC01	Mean	4.95E−12	1.40E−10
	Std	9.24E−13	1.21E−10
	Time(s)	17.15478114	17.28749408
CEC02	Mean	5.76E−12	3.09E−10
	Std	4.13E−12	2.58E−10
	Time(s)	18.4906462	20.90592599
CEC03	Mean	1.36E+01	52.73252666
	Std	2.14E−02	8.29E−11
	Time(s)	16.40691744	15.34842814
CEC04	Mean	8.998415914	8.998926142
	Std	1.96E−05	6.60E−05
	Time(s)	16.44151483	17.00241801
CEC05	Mean	4.65E−08	2.85E+02
	Std	2.16E−08	2.22E−04
	Time(s)	16.64680717	15.42366769
CEC06	Mean	1.82E−06	43.35792483
	Std	5.63E−07	3.18E−06
	Time(s)	15.58736248	15.84327443
CEC07	Mean	9.47E−08	1.433217519
	Std	7.41E−08	6.15E−09
	Time(s)	11.6278939	11.85922258
CEC08	Mean	9.86E−07	1.25E−05
	Std	4.70E−07	8.69E−06
	Time(s)	16.0366317	16.74224679
CEC09	Mean	−1.33E−06	−4.401215885
	Std	1.61E−06	3.409789061
	Time(s)	12.24845986	12.47100568
CEC10	Mean	66.18822172	77.81946403
	Std	112.4076859	2.520774604
	Time(s)	14.35157094	14.49965868
CEC11	Mean	14.92256511	18.06415685
	Std	4.54E−10	2.24E−06
	Time(s)	12.37109933	12.83158839

Continued

CEC12	Mean	33.92920169	76.96902002
	Std	7.539802782	1.83E−09
	Time(s)	12.85865462	12.98471289
CEC13	Mean	0.001098596	258.125
	Std	3.29E−03	2.125
	Time(s)	20.21733594	20.87134676
CEC14	Mean	0.244309396	0.96925917
	Std	0.316349688	2.71E−02
	Time(s)	16.51488175	18.18085654
CEC15	Mean	11.74930116	19.49499386
	Std	3.007734851	5.007423369
	Time(s)	14.43568053	14.38639846
CEC16	Mean	7.705309519	264727.795
	Std	0.165002392	6.04E+05
	Time(s)	14.64010904	14.70395908

Table 5. Performance comparison of MSD-SSA and SSA on test functions (30*D*).

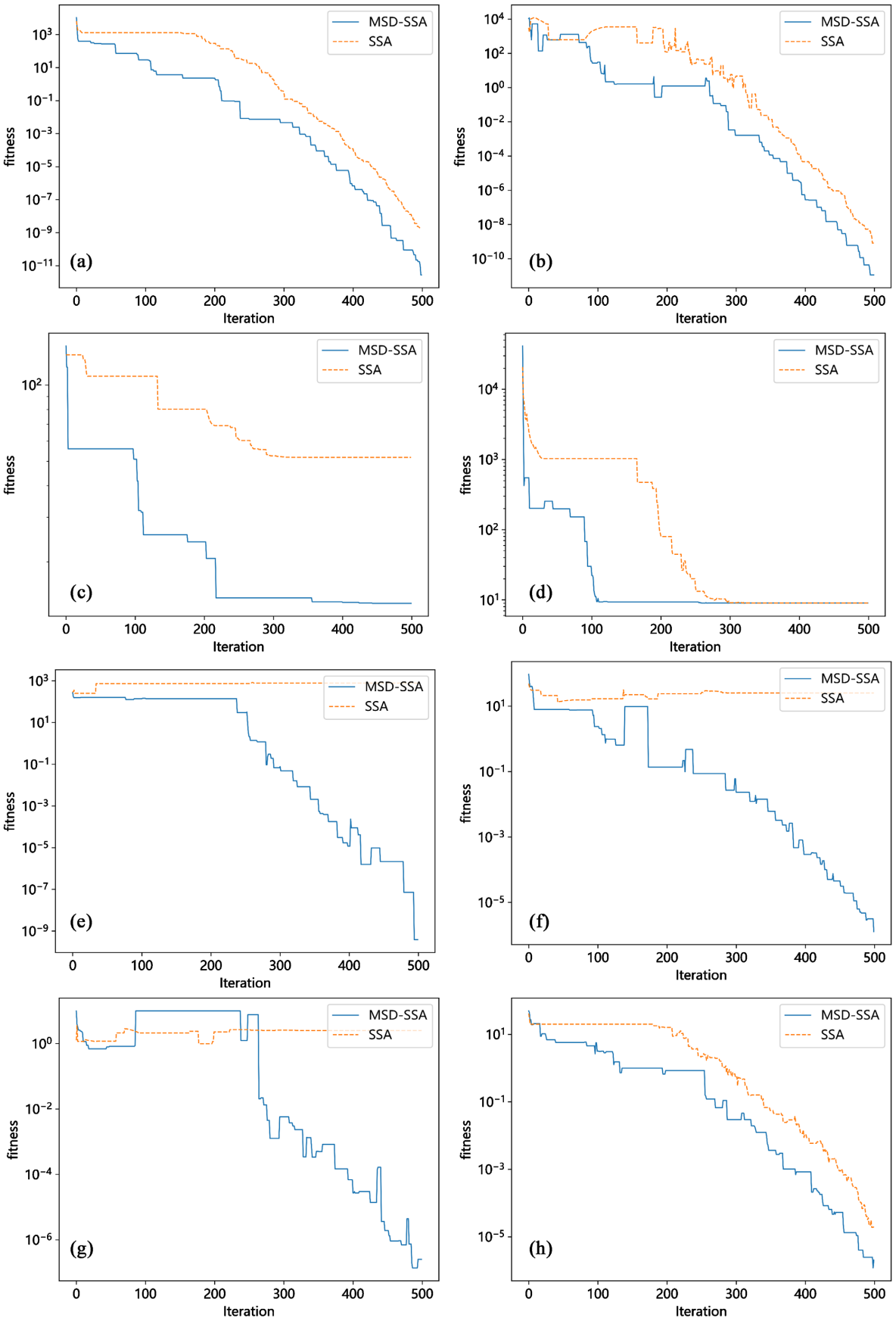
Function	Stats	MSD-SSA	SSA
CEC01	Mean	4.07E−11	8.49E−02
	Std	4.14E−11	2.53E−01
	Time(s)	51.15962758	52.07183607
CEC02	Mean	8.43E−11	4.20E−05
	Std	3.43E−11	1.23E−04
	Time(s)	56.50194526	63.13624792
CEC03	Mean	13.58830671	216.8995591
	Std	4.74E−03	3.99E−09
	Time(s)	49.91018164	46.72936332
CEC04	Mean	28.99603765	28.99797197
	Std	1.02E−04	1.28E−04
	Time(s)	53.07714818	52.0836818
CEC05	Mean	1.33E−07	7.50E+02
	Std	2.69E−07	1.54E−04
	Time(s)	37.88733799	39.50639632
CEC06	Mean	2.03E−06	0.17485355
	Std	1.49E−07	4.80E−01
	Time(s)	60.88767364	52.5819994

Continued

CEC07	Mean	1.69E−07	4.251761281
	Std	7.76E−08	9.14E−07
	Time(s)	33.93889532	35.76503084
CEC08	Mean	1.96E−06	1.00E−01
	Std	4.65E−07	2.32E−01
	Time(s)	47.31576941	16.74224679
CEC09	Mean	−1.22E−06	140.9718139
	Std	7.82E−06	14.72215499
	Time(s)	34.64287498	35.75151873
CEC10	Mean	412.7758021	3.40E+24
	Std	192.0322279	2.06E+16
	Time(s)	50.08848522	50.88623056
CEC11	Mean	15.86505171	18.06415776
	Std	7.441012651	2.00E−09
	Time(s)	46.07220759	47.57178798
CEC12	Mean	18.84955905	246.6150233
	Std	13.15724649	1.83E−09
	Time(s)	46.61474404	47.4385241
CEC13	Mean	0.001098596	258.125
	Std	3.29E−03	2.125
	Time(s)	20.21733594	20.87134676
CEC14	Mean	2.774791549	3.080194387
	Std	0.158284286	1.94E−03
	Time(s)	53.51142323	54.34439452
CEC15	Mean	55.81424714	80.32612567
	Std	2.267501187	17.39791280
	Time(s)	46.47466836	48.06013227
CEC16	Mean	1265.662073	1.24365E+11
	Std	737.6460717	1.12E+07
	Time(s)	52.5728548	51.41614304

SSA, its standard deviation also has reaches 10^{-7}. In terms of convergence speed, MSD-SSA converges to the optimal point faster than SSA.

For CEC2017 test functions, the CEC2017 test functions [20] include single and multi-modal problems, which can effectively evaluate the performance of algorithms. **Table 4** tests 16 CEC2017 test functions [20] with 10-dimensions. The experimental results show that under the same environment, MSD-SSA has


MSD-SSA
SSA
fitness
Iteration
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)

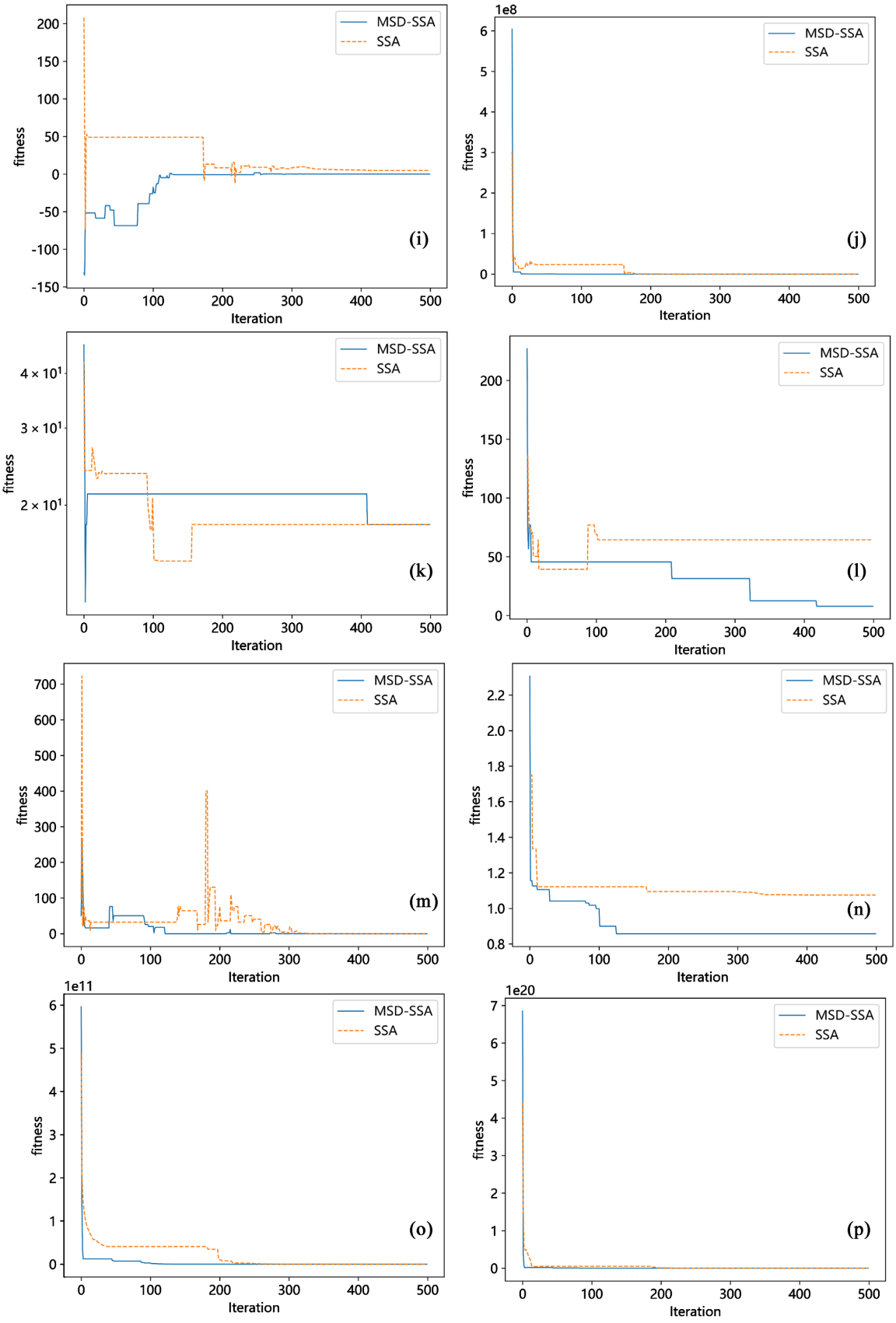
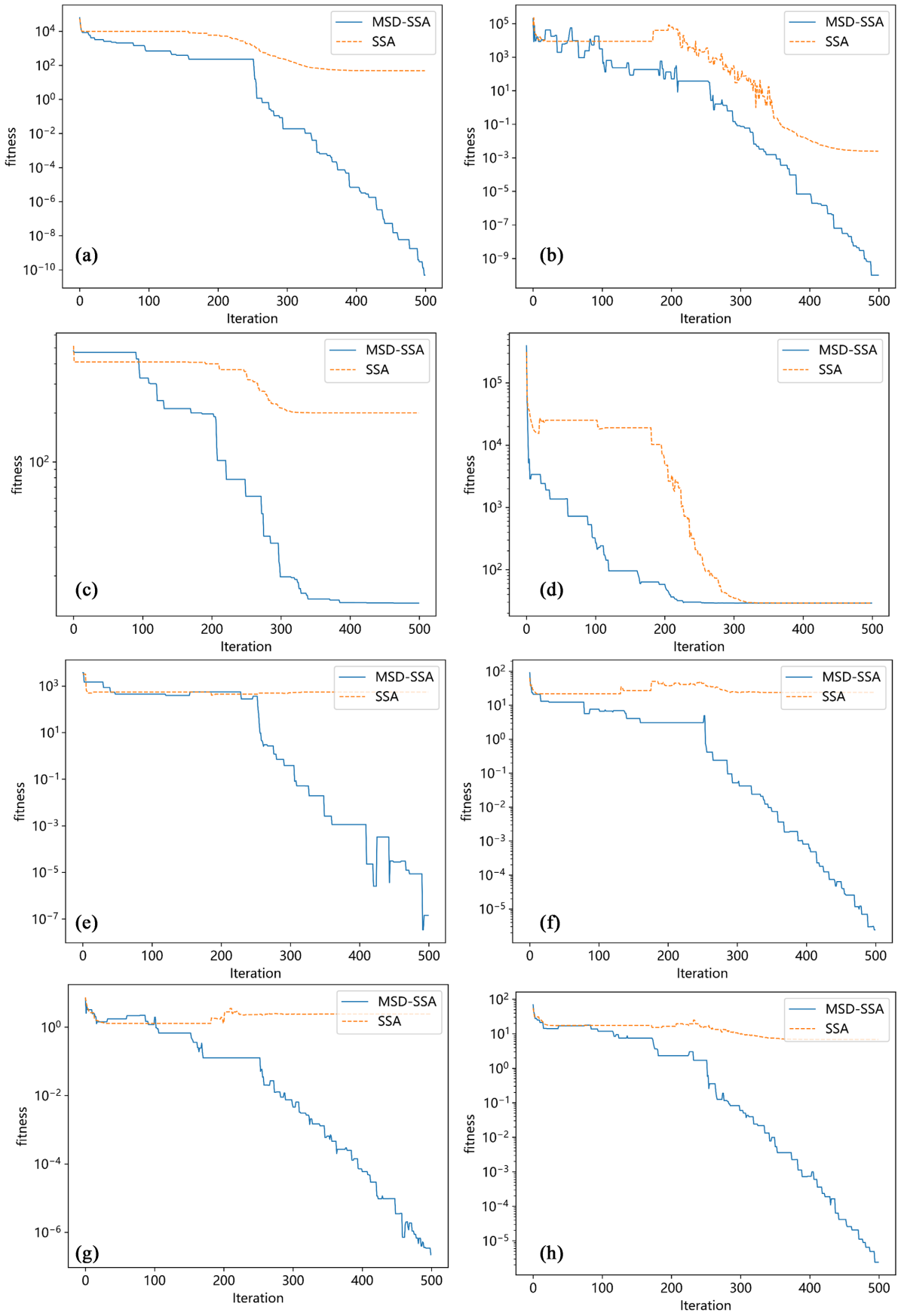


Figure 5. CEC2017 test function iteration curve (10*D*).


MSD-SSA
SSA
fitness
Iteration
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)

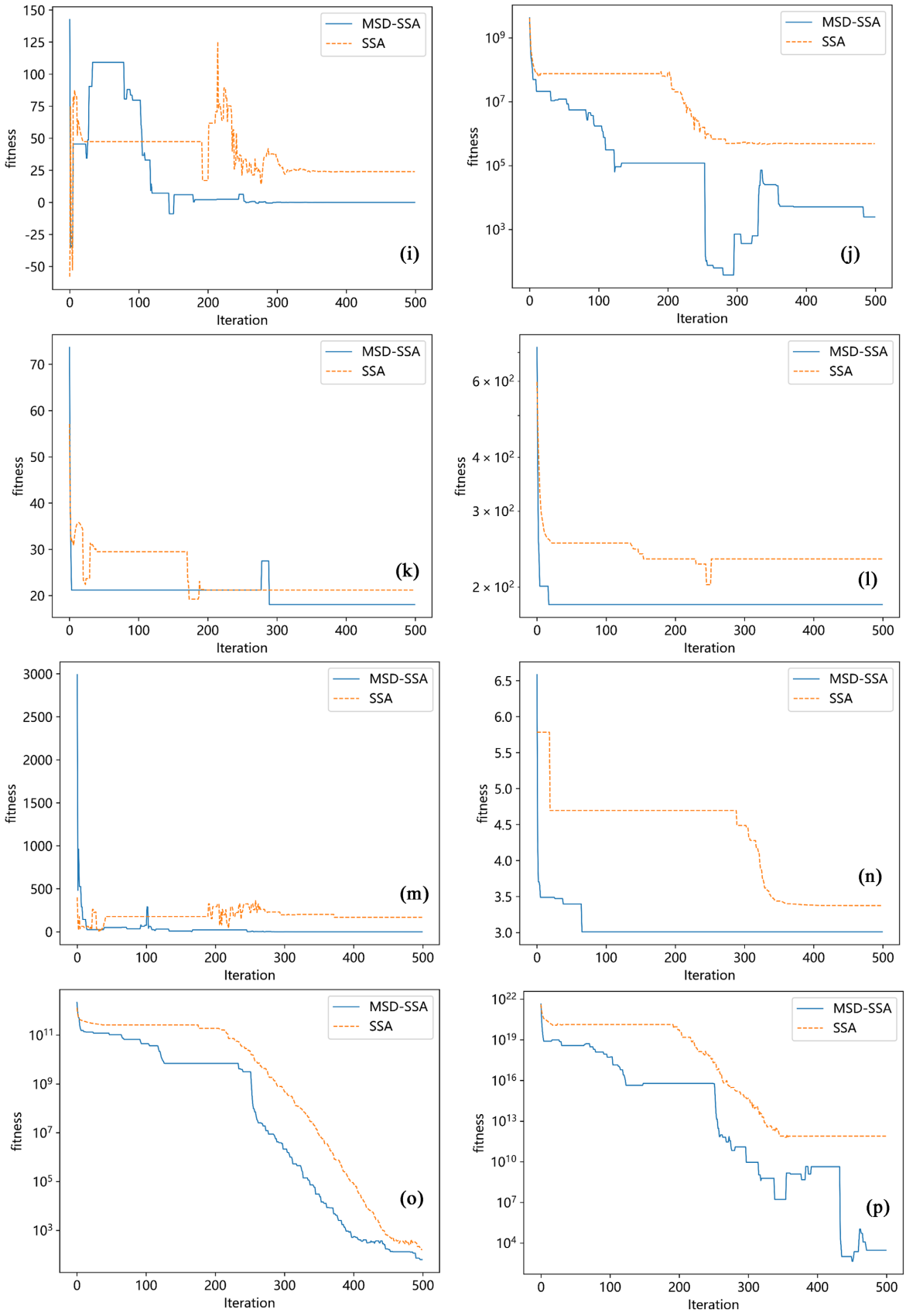


Figure 6. CEC2017 test function iteration curve (30*D*).

a better convergence effect and faster convergence speed than SSA. For most CEC2017 test functions, MSD-SSA is more stable. However, Table 4 shows that the standard deviation of CEC10 and CEC12 is too large, indicating that the robustness of the improved algorithm is poor for CEC10 and CEC12.

Table 5 tests 16 CEC2017 test functions [20] with 10 dimensions. Compared with the CEC2017 10-dimensionals test function, the 30-dimensionals test function is more complex. According to Table 5, MSD-SSA is superior to SSA in terms of convergence accuracy, manipulation speed and robustness. However, the standard deviations of CEC10, CEC11, CEC12, CEC15 and CEC16 are too large, indicating that the robustness of MSD-SSA is poor and needs to be improved.

In summary, the improved algorithm has a stronger search capability and faster convergence to the target value for most functions.

5. Engineering Problems and Result Discussion

5.1. Cantilever Beam Design Problem

In the era of big data, solving the constrained optimization problem is crucial to engineering. Although benchmark functional testing was addressed in the previous section, the issues in actual projects are within specific limitations. Optimize practical engineering problems using SSA and MSD-SSA, and verify the feasibility of MSD-SSA for practical engineering problems.

The goal of CBD [24] is to minimize the weight of cantilever beam and square section, see Figure 7. The variables of this problem are composed of five hollow square members. The CBD problem [24] is expressed as follows:

$$\begin{aligned} \min \quad & f(x) = 0.0624\left(x_1 + x_2 + x_3 + x_4 + x_5\right) \\ \text{s.t.} \quad & g(x) = \frac{61}{x_1^3} + \frac{27}{x_2^3} + \frac{19}{x_3^3} + \frac{7}{x_4^3} + \frac{1}{x_5^3} - 1 \le 0 \\ & 0.01 \le x_1, x_2, x_3, x_4, x_5 \le 100 \end{aligned} \tag{19}$$

Do 30 independent experiments to verify the effectiveness of the algorithm. See Table 6 for the experimental results.

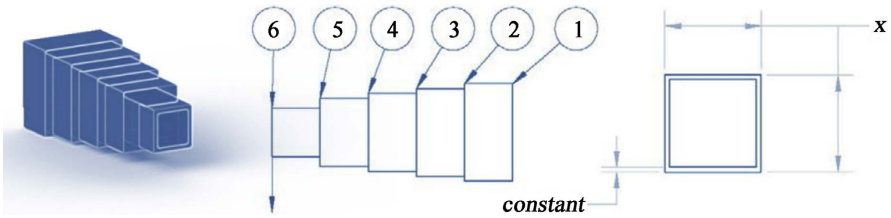


Figure 7. Cantilever beam structure [25].

Table 6. Comparison results of MSD-SSA with SSA for CBD problem [24].

Algorithm	Optimize the Solution					Optimal Cost
	x_1	x_2	x_3	x_4	x_5	
MSD-SSA	6.221899	4.667276	4.307131	3.680037	2.177050	1.313731
SSA	6.010624	5.320413	4.493760	3.498178	2.151536	1.339961

5.2. Tension/Compression Spring Design Problem

The purpose of TCSD [26] is to minimize the weight of the spring. This problem has three variables, namely wire diameter, mean coil diameter, and the number of dynamic coils. Equation (20) describes the TCSD [26] problem.

Do 30 independent experiments to verify the effectiveness of the algorithm. See Table 7 for the experimental results.

$$\begin{aligned}
&\min \quad f(x)=(x_3+2)x_2x_1^2\\
&\text{s.t.} \quad g_1(x)=1-\frac{x_2^3x_3}{71785x_1^4}\le 0\\
&g_2(x)=\frac{4x_2^2-x_1x_2}{12566\left(x_2x_1^3-x_1^4\right)}+\frac{1}{5180x_1^2}\le 0\\
&g_3(x)=1-\frac{140.45x_1}{x_2^3x_3}\le 0\\
&g_4(x)=\frac{x_1+x_2}{1.5}-1\le 0\\
&0.05\le x_1\le 2, 0.25\le x_2\le 1.3, 2\le x_3\le 15
\end{aligned} \tag{20}$$

5.3. Schematic Views of Speed Reducer Design

The goal is to minimize the weight of a speed reducer so that the engine and propeller can rotate efficiently. This problem involves constraints on stresses in the shafts, transverse deflection of the shafts, surface stress and bending stress of the gear teeth (see Figure 8). Equation (21) describes the SRD problem [27].

Do 30 independent experiments to verify the effectiveness of the algorithm. See Table 8 for the experimental results.

Table 7. Comparison results of MSD-SSA with SSA for TCSD problem [26].

Algorithm	Optimize the Solution			Optimal Cost
	x_1	x_2	x_3	
MSD-SSA	0.050000	0.317851	13.97190	0.0126917
SSA	0.054734	0.434490	7.854553	0.0128271

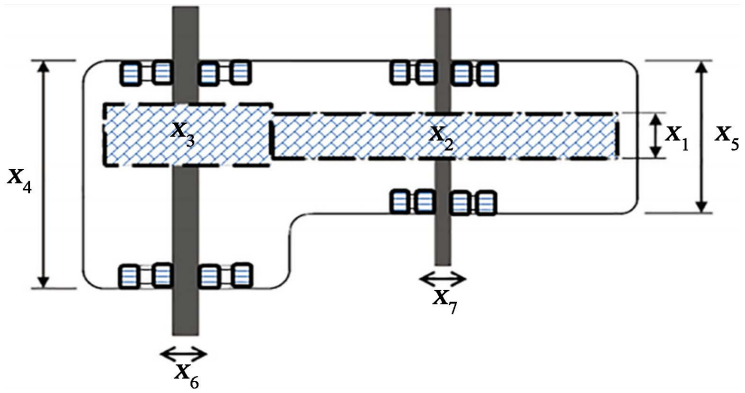


Figure 8. Schematic views of speed reducer design [27].

$$
\begin{aligned}
\min \quad & f(x)=0.7854x_1x_2^2\left(3.3333x_3^2+14.9334x_3-43.0934\right)\\
& -1.508x_1\left(x_6^2+x_7^2\right)+7.4777\left(x_6^3+x_7^3\right)+0.7854\left(x_4x_6^2+x_5x_7^3\right)\\
\text{s.t.} \quad & g_1(x)=\frac{27}{x_1x_2^2x_3}-1\le 0, g_2(x)=\frac{397.5}{x_1x_2^2x_3^2}-1\le 0,\\
& g_3(x)=\frac{1.93x_4^3}{x_2x_6^4x_3}-1\le 0, g_4(x)=\frac{1.93x_5^3}{x_2x_7^4x_3}-1\le 0,\\
& g_5(x)=\frac{\left[\left(745x_4/x_2x_3\right)^2+16.9\times10^6\right]^{1/2}}{110x_6^3}-1\le 0,\\
& g_6(x)=\frac{\left[\left(745x_5/x_2x_3\right)^2+157.5\times10^6\right]^{1/2}}{85x_7^3}-1\le 0,\\
& g_7(x)=\frac{x_2x_3}{40}-1\le 0, g_8(x)=\frac{5x_2}{x_1}-1\le 0,\\
& g_9(x)=\frac{x_1}{12x_2}-1\le 0, g_{10}(x)=\frac{1.5x_6+1.9}{x_4}-1\le 0,\\
& g_{11}(x)=\frac{1.1x_7+1.9}{x_5}-1\le 0\\
& 2.6\le x_1\le 3.6, 0.7\le x_2\le 0.8, 17\le x_3\le 28,\\
& 7.3\le x_4\le 8.3, 7.3\le x_5\le 8.3, 2.9\le x_6\le 3.9, 5.0\le x_7\le 5.5
\end{aligned}
\tag{21}
$$

5.4. Result Discussion

For the CBD problem [24], the optimization results of MSD-SSA and SSA are 1.313731 and 1.339961, the optimal solution is shown in Table 6.

For the TCSD problem [26], the optimization results of MSD-SSA and SSA are 0.0126917 and 0.0128271, the optimal solution is shown in Table 7.

For the SRD problem [27], the optimization results of MSD-SSA and SSA are 2997.0889 and 3041.6166, the optimal solution is shown in Table 8.

The average operation time(s) obtained from the above practical engineering problems are as follows:

It can be seen from Table 9 that MSD-SSA takes less time to solve the same

Table 8. Comparison results of MSD-SSA with SSA for SRD problem.

Algorithm	Optimize the Solution							Optimal Cost
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
MSD-SSA	3.5018	0.7	17	7.3000	7.7348	3.3526	5.2880	2997.0889
SSA	3.5000	0.7	17	7.3188	8.0557	3.4982	5.2868	3041.6166

Table 9. Time spent by MSD-SSA and SSA to solve practical problems.

Algorithm	Time(s)		
	CBD	TCSD	SRD
MSD-SSA	1.083599512	0.768733430	1.719269466
SSA	1.306071830	0.799066846	1.804705365

practical engineering problems than SSA, which shows that the improved algorithm is superior to the original algorithm in terms of convergence speed.

6. Conclusions and Implications

Based on the drawbacks of low convergence accuracy, slow convergence speed and easy to fall into a local optimum of the Salp Swarm Algorithm, this paper proposes a Multi-Strategy-Driven Salp Swarm Algorithm. Firstly, the Levy flight strategy with small probability is introduced to the leaders and the crossover operator operation is adopted to improve the global exploration ability in the early stage of the algorithm, so that the leaders can effectively jump out of the trap of local optimum. Secondly, Brownian motion is adopted in the latest iteration to improve the exploitation ability of the algorithm near the global optimum and improve the convergence accuracy of the algorithm. Finally, the positional relationship between the leaders is used to update the followers, so that the leaders have the purpose. The final update of the followers with the positional relationship between leaders, so that the leaders purposefully follow the better individual, improves the convergence speed of the algorithm. The above improvements optimize the global and local search of the algorithm, balancing the exploration and exploitation functions of the algorithm. In the simulation experiments, benchmark test functions and three real engineering problems were used for comparison tests to examine the performance of the algorithm. The experimental results show that the improved algorithm has higher global convergence and faster convergence than the original algorithm, and also outperforms the original algorithm in terms of algorithm robustness. The next steps will continue to improve the optimal performance of the tunicate algorithm, and increase its optimization accuracy, convergence speed and convergence stability. Based on this, it will be applied to solve more practical problems.

SSA can be applied in many different fields, such as optimization design, machine learning, and so on. Specifically, MSD-SSA can be applied to problems that require finding the optimal solution, such as optimizing design variables to achieve the best performance in engineering, or adjusting model parameters to obtain the best prediction results in machine learning. The optimization capability of this algorithm can improve the convergence speed and accuracy of the algorithm, and it has certain advantages for solving complex and high-dimensional problems. In general, the application scenarios of MSD-SSA are relatively broad, and they can provide assistance in solving problems in multiple fields.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Holland, J.H. (1992) Genetic Algorithms. *Scientific American*, **267**, 66-73.

https://doi.org/10.1038/scientificamerican0792-66

[2] Das, S. and Suganthan, P.N. (2010) Differential Evolution: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation*, **15**, 4-31. https://doi.org/10.1109/TEVC.2010.2059031

[3] Kennedy, J. and Eberhart, R. (1995) Particle Swarm Optimization. *Proceedings of ICNN'95—International Conference on Neural Networks*, Perth, 27 November-December 1995, 1942-1948. https://doi.org/10.1109/ICNN.1995.488968

[4] Dorigo, M., Birattari, M. and Stutzle, T. (2006) Ant Colony Optimization. *IEEE Computational Intelligence Magazine*, **1**, 28-39. https://doi.org/10.1109/MCI.2006.329691

[5] Karaboga, D. (2010) Artificial Bee Colony Algorithm. *Scholarpedia*, **5**, Article No. 6915. https://doi.org/10.4249/scholarpedia.6915

[6] Mareli, M. and Twala, B. (2018) An Adaptive Cuckoo Search Algorithm for Optimization. *Applied Computing and Informatics*, **14**, 107-115. https://doi.org/10.1016/j.aci.2017.09.001

[7] Połap, D. and Woźniak, M. (2017) Polar Bear Optimization Algorithm: Meta-Heuristic with Fast Population Movement and dynamic Birth and Death Mechanism. *Symmetry*, **9**, Article 203. https://doi.org/10.3390/sym9100203

[8] Mirjalili, S., Mirjalili, S.M. and Lewis, A. (2014) Grey Wolf Optimizer. *Advances in Engineering Software*, **69**, 46-61. https://doi.org/10.1016/j.advengsoft.2013.12.007

[9] Mirjalili, S. and Lewis, A. (2016) The Whale Optimization Algorithm. *Advances in Engineering Software*, **95**, 51-67. https://doi.org/10.1016/j.advengsoft.2016.01.008

[10] Mirjalili, S., *et al*. (2017) Salp Swarm Algorithm: A Bio-Inspired Optimizer for Engineering Design Problems. *Advances in Engineering Software*, **114**, 163-191. https://doi.org/10.1016/j.advengsoft.2017.07.002

[11] Thawkar, S. (2021) A Hybrid Model Using Teaching-Learning-Based Optimization and Salp Swarm Algorithm for Feature Selection and Classification in Digital Mammography. *Journal of Ambient Intelligence and Humanized Computing*, **12**, 8793-8808. https://doi.org/10.1007/s12652-020-02662-z

[12] Neggaz, N., Ewees, A.A., Abd Elaziz, M. and Mafarja, M. (2020) Boosting Salp Swarm Algorithm by Sine Cosine Algorithm and Disrupt Operator for Feature Selection. *Expert Systems with Applications*, **145**, Article ID: 113103. https://doi.org/10.1016/j.eswa.2019.113103

[13] Zhang, H., *et al*. (2022) A Multi-Strategy Enhanced Salp Swarm Algorithm for Global Optimization. *Engineering with Computers*, **38**, 1177-1203. https://doi.org/10.1007/s00366-020-01099-4

[14] Zhang, H., *et al*. (2023) Differential Evolution-Assisted Salp Swarm Algorithm with Chaotic Structure for Real-World Problems. *Engineering with Computers*, **39**, 1735-1769. https://doi.org/10.1007/s00366-021-01545-x

[15] Chen, L.X. and Mu, Y.M. (2021) Improved Salp Swarm Algorithm. *Application Research of Computers*, **38**, 1648-1652. (In Chinese)

[16] Yang, G.Y., Wu, D.F., Liu, F.K. and Xu, T.Q. (2023) Improved Salp Swarm Algorithm with Multi-Strategy. *Application Research of Computers*, **40**, 704-709. (In Chinese)

[17] Liu, J.S., Yuan, M.M. and Li, Y. (2022) Robot Path Planning Based on Improved Salp Swarm Algorithm. *Journal of Computer Research and Development*, **59**, 1297-1314. (In Chinese)

[18] Zhang, Y. and Qin, L.X. (2020) Improved Salp Swarm Algorithm Based on Levy

Flight Strategy. *Computer Science*, **47**, 154-160. (In Chinese)

[19] Xie, C. and Zheng, H.Q. (2022) A Novel Salpa Warm Algorithm and Application. *Computer Engineering & Science*, **44**, 84-190. https://doi.org/10.54097/hset.v24i.3896

[20] Wu, G.H., Mallipeddi, R. and Suganthan, P.N. (2017) Problem Definitions and Evaluation Criteria for the CEC 2017 Competition on Constrained Real-Parameter Optimization. Technical Report. https://www.researchgate.net/publication/317228117

[21] Zhang, J. and Wang, J.-S. (2020) Improved Salp Swarm Algorithm Based on Levy Flight and Sine Cosine Operator. *IEEE Access*, **8**, 99740-99771. https://doi.org/10.1109/ACCESS.2020.2997783

[22] Mantegna, R.N. (1994) Fast, Accurate Algorithm for Numerical Simulation of Levy Stable Stochastic Processes. *Physical Review E*, **49**, 4677-4683. https://doi.org/10.1103/PhysRevE.49.4677

[23] Yao, X., Liu, Y. and Lin, G. (1999) Evolutionary Programming Made Faster. *IEEE Transactions on Evolutionary Computation*, **3**, 82-102. https://doi.org/10.1109/4235.771163

[24] Yildiz, A.R. (2019) A Novel Hybrid Whale-Nelder-Mead Algorithm for Optimization of Design and Manufacturing Problems. *The International Journal of Advanced Manufacturing Technology*, **105**, 5091-5104. https://doi.org/10.1007/s00170-019-04532-1

[25] Saremi, S., Mirjalili, S. and Lewis, A. (2017) Grasshopper Optimization Algorithm: Theory and Application. *Advances in Engineering Software*, **105**, 30-47. https://doi.org/10.1016/j.advengsoft.2017.01.004

[26] Zhao, S., *et al*. (2022) Elite Dominance Scheme Ingrained adaptive Salp Swarm Algorithm: A Comprehensive Study. *Engineering with Computers*, **38**, 4501-4528. https://doi.org/10.1007/s00366-021-01464-x

[27] Askari, Q., Saeed, M. and Younas, I. (2020) Heap-Based Optimizer Inspired by Corporate Rank Hierarchy for Global Optimization. *Expert Systems with Applications*, **161**, Article ID: 113702. https://doi.org/10.1016/j.eswa.2020.113702