IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 12, 2019, accepted July 25, 2019, date of publication August 5, 2019, date of current version October 2, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2933265

A Comprehensive Improved Salp Swarm Algorithm on Redundant Container Deployment Problem

BOTAO MA[1,2], HONG NI[1,2], XIAOYONG ZHU[1], AND RAN ZHAO[1,2]
[1]National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China
[2]University of Chinese Academy of Sciences, Beijing 100049, China

Corresponding author: Xiaoyong Zhu (zhuxy@dsp.ac.cn)

This work was supported by the Strategic Leadership Project of Chinese Academy of Sciences: SEANET Technology Standardization Research System Development under Project XDC 02070100.

ABSTRACT As a representative of lightweight virtualization, container technology has been widely used in cloud services and edge computing applications. However, in the resource pool scenario composed by multiple intelligent terminal devices, considering the limited resources and poor stability of these devices, it is necessary to split the overall service into multiple microservices and deploy the backups of them in respective containers. Traditional container scheduling policies tend to be less effective in solving such problems. Therefore, the article used a meta-heuristic algorithm to solve this kind of problems. Based on a newly proposed salp swarm algorithm (SSA), the paper presented a comprehensive improved SSA (CISSA). CISSA improved the performance of the original SSA by 4 steps. To verify the performance of CISSA in different kinds of test functions, the algorithm was compared with 7 commonly-used meta-heuristic algorithms in 29 benchmark functions provided by the author of SSA. In addition, the article constructed three container cluster models of different sizes, all these algorithms were used to solve these redundant container deployment problems, the experimental results indicate that the CISSA is superior to other algorithms in such problems of different dimensions.

INDEX TERMS Chaotic maps, microservice, redundant container deployment, salp swarm algorithm, terminal device.

I. INTRODUCTION

With the popularization of intelligent terminal equipment, there are a large number of intelligence terminal devices with idle resources, These devices are closer to users, in order to make reasonable use of these idle resources, it can be considered to integrate these devices into a resource pool to provide overall service, each device is responsible for some microservices. However, different from the excellent stability of servers in cloud service, these terminals devices have problems such as small amount of idle resources and poor reliability. Therefore, when building resource pools with these unstable devices, we should consider dividing the overall service into several fine-grained microservices. In the meantime, in order to guarantee the stability of the system, more redundant backups should be added for each microservice. To ensure that when some devices break down, backups on the rest of the devices can continue to complete the work, the backups of the same microservice should be deployed on different devices as much as possible [1].

At the same time, backups for each microservice deployed in multiple devices need to run with the same operating environment. In consideration of the heterogeneity of terminal devices, virtualization techniques are usually used to build multiple isolated running environments in a device. As a representative of lightweight virtualization, Docker is transparent to the underlying physical machine and costs less system resources [2]. Therefore, the common practice in the industry is to run each microservice on a mutually independent Docker container, the independence of each microservice is achieved through the isolation of the container. When one microservice upgrades or fails, the others will not be affected. In summary, the deployment problem for microservices can be equivalent to container deployment problem.

The associate editor coordinating the review of this manuscript and approving it for publication was Rajesh Kumar.

In recent years, many researches on the container deployment problem have been made on the basis of different evaluation indictors [3], [4]. Based on the background above, the paper proposed a scheme to deploy multiple containers in several terminal devices, on the premise that the total resources of containers deployed in one device should be less than the total resources of this device, the fitness of redundant container deployment scheme was evaluated by the robustness of the system and total completion time of service.

It can be seen that the redundant container deployment problem in terminal device is a NPC problem, and there are multiple solving solutions. When the dimensions of the solution sets become larger as the number of container increases, the optimization problem will be much more complex. For solving these single-objective problems with multi-dimensions, several meta-heuristic algorithms have been proposed in recent years. These meta-heuristic algorithms are more effective than the traditional methods.

Salp swarm algorithm (SSA) is a population-based meta-heuristic optimization algorithm proposed by Mirjalili et al in 2017 [5], which mimics the predatory behavior of salp swarm. The algorithm is easy to implement due to the only main controlling parameter. The performance of SSA is verified by 3 types of benchmark functions, according to the comparison of these test results, the excellent exploration ability makes SSA be capable of solving optimization problems in complex situations.

With its good performance, SSA has been widely used in several areas. In [6] and [7], SSA has been used in multi-level color image segmentation, T. K. Mohapatra and B. K. Sahu [8] have used SSA to optimize PID control, and SSA has also been implied in [9] to design power system stabilizer.

However, SSA still has some limitations. The excellent exploration ability may bring the deficiency of optimization ability, meanwhile, SSA also suffer from the shortcoming of mediocre convergence rate. Since the algorithm is relatively new, there are few corresponding improvements of it. Tavazoei and Haeri [10] have designed a chaos-induced and mutation-driven schemes, the algorithm is optimized at the expense of time complexity. Werth *et al.* [6] have added Levy flight in the original SSA, but the main concern is the application scenario of the algorithm rather than algorithm optimization.

To overcome the disadvantage and improve the accuracy of SSA, a comprehensive improved salp swarm algorithm was proposed in the article. At the same time, the proposed algorithm would be used to solve redundant container deployment problem. The main contributions of the work are described as follows:

- A more suitable initial distribution is selected by comparison. And using revised main controlling parameter c_1 to reconstruct two-stage exploration mechanism.
- Setting two thresholds to distinguish the different roles of these search agents during the iterations and using a position updating method based on spiral line to replace the original weighting scheme.

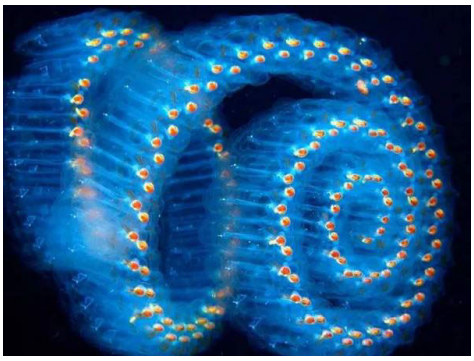

FIGURE 1. The trajectory of salp swarm.

- A redundant container deployment model was proposed, which was judged by the robustness of the system and total completion time of service.

The rest of this paper is organized as follows: the original SSA is briefly introduced in Section II. Section III presents the comprehensive improved salp swarm algorithm (CISSA). Section IV displays experimental results and performance of all test algorithms. Section V describes the redundant container deployment model and some corresponding experimental parameters. Section VI applies the proposed CISSA to solve multi-sized redundant container deployment problem. In the end, conclusions and future expectations are shown in Section VII.

II. A REVIEW OF SSA

The mathematical model of SSA simulates the predatory behavior of salp swarm, which belong to the family of Salpidae. The algorithm divides the population of salp swarm into two groups: leader and followers. In each iteration, all non-leader individual follows its previous individual, rather than move independently towards to the optimal value.

The motion trail of salp swarm is shown in Figure.1, positions of these salp swarm are defined in a D-dimensional search space and D is the dimensions of variables and solutions, meanwhile, initial positions of N search agents are initialized by random distribution. There is a food source in the search space as the target of swarm, which is represented as $F = [F_1, F_1 \ldots F_D]^T$.

The position update formula is proposed as Eq. (2.1):

$$X_j^1 = \begin{cases} F_j + c_1 \left(\left(ub_j - lb_j \right) c_2 + lb_j \right) & 0.5 \le c_3 \le 1 \\ F_j - c_1 \left(\left(ub_j - lb_j \right) c_2 + lb_j \right) & 0 \le c_3 < 0.5 \end{cases} \tag{2.1}$$

where X_j^1 is the position of the leader salp in the j-th dimension, F_j is the position of food source in the j-th dimension, ub_j indicates the upper bound of j-th dimension, lb_j shows the lower bound of j-th dimension, c_1 is nonlinear convergence parameter and c_2, c_3 are random numbers between 0 and 1.

Eq. (2.1) shows that the leader only updates its position based on food source, the most important coefficient c_1 is used to balance exploration and exploitation which is defined

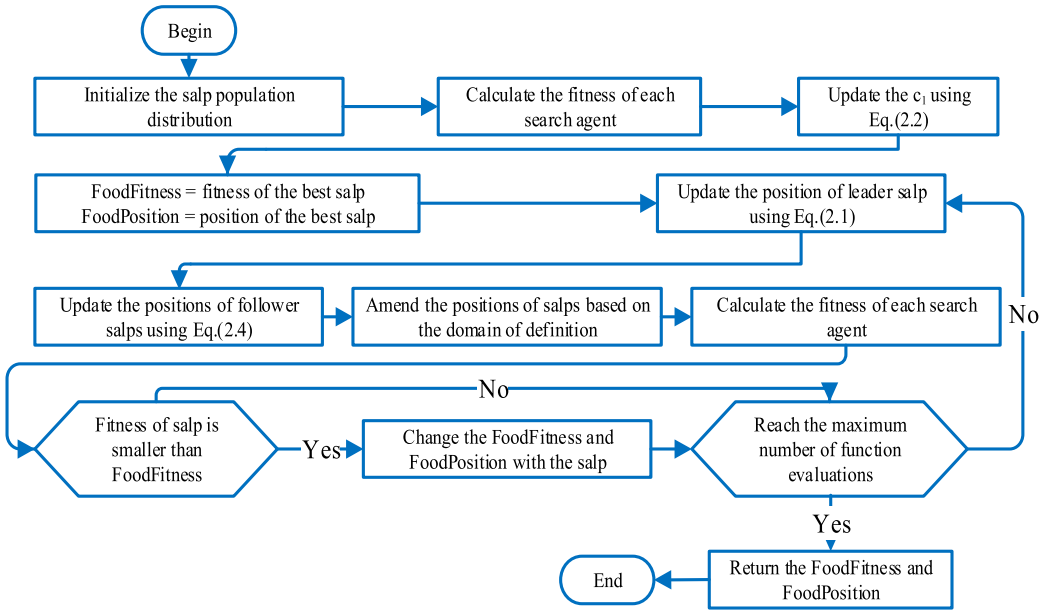

FIGURE 2. The flowchart of SSA.

as follows:

$$c_1 = 2e^{-(\frac{4l}{L})^2} \tag{2.2}$$

where l is the current iteration and L is the maximum number of iterations.

The aim of generating random parameters c_2, c_3 is to make a disturbance. These parameters are used to determine the direction in which each individual moves around its target.

To simulate the trajectory of the swarm, basic SSA updates the position of the swarm as Eq. (2.3):

$$X_j^i = \frac{1}{2}at^2 + v_0 ti \geq 2 \tag{2.3}$$

where X_j^i is the position of i-th follower salp in j-th dimension, t represents time, v_0 shows the initial speed, and a is the acceleration. In the model, time is replaced by iterations, and $v_0 = 0$, the equation can be expressed as follows:

$$X_j^i = \frac{1}{2}\left(X_j^i + X_j^{i-1}\right) \quad i \geq 2 \tag{2.4}$$

where X_j^i shows the position of i-th follower salp in j-th dimension.

III. THE IMPROVED METHOD OF CISSA

The proposed CISSA is equipped with different initial distribution and exploitation mechanism with better physical meaning. Firstly, a kind of chaotic mapping distribution took the place of the current random distribution. Secondly, CISSA used two-stage exploration mechanism to control the step length much precisely. Thirdly, disturbance factor was integrated in the main parameter c_1 to increase the diversity of solutions. Lastly, considering the original convergence method of follower salps was short of physical meaning, a kind of spiral motion was used to replace the original updating method.

A. INITIAL DISTRIBUTION WITH CHAOTIC MAP

Meta-heuristic algorithm usually initializes population distribution with random solutions, then explores and exploits the search space randomly with specific probability. However, meta-heuristic algorithm is sensitive to the initial distribution of search agents [10], in this chapter, our main motivation was utilizing the sequences which were generated from various chaotic maps to replace the random numbers in definitional domain.

Considering the precision of original SSA is affected by its relatively long exploration step, it performs unsatisfactorily in unimodal test functions. However, in order to maintain its good performance in multimodal and composite test functions, we need to retain the exploratory ability of the original algorithm, so choosing a more suitable initial distribution for SSA can improve the accuracy of the algorithm to a certain extent.

It's usually difficult to evaluate the qualities of the generated sequences, in order to achieve the task [11], various SSA algorithms with different chaotic maps were proposed to compare the performance of each initial distribution. In this chapter, 5 most widely used chaotic maps were used to verify the importance of initial distribution for solving the optimal solutions. These chaotic maps have their unique regularity of distribution. The mathematical expressions and characters are described in the following subsections and the visualization figures are shown in Figure 3.

1) CHEBYSHEV MAP

Chebyshev map is formulated in Eq. (3.1).

$$x_{k+1} = \cos(P \times \sim \cos^{-1}(x_k)) \tag{3.1}$$

The iterative equation of Chebyshev chaotic map is simple and easy to implement, when the parameter P is more than 2,

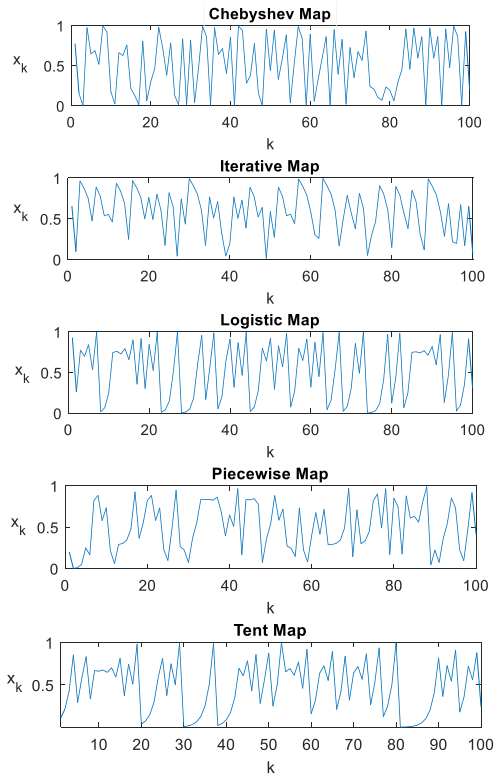

FIGURE 3. Visualization of chaotic maps.

the Chebyshev map can enter the chaotic region and produce an infinite length of non-periodic chaotic real sequences under infinite precision conditions [12]. Meanwhile, in terms of uniformity, Chebyshev chaotic sequences are more likely to be distributed near the boundary. For the test functions with the optimal values near the boundary, heuristic algorithms with initial distribution using the method always work better.

2) ITERATIVE MAP

Iterative chaotic map is between (0,1] with infinite collapses, compared with Chebyshev map, it is more sensitive to the initial value of iteration [13]. The equation is represented in Eq. (3.2).

$$x_{k+1} = \left| sin\ (\frac{P}{x_k}) \right| \tag{3.2}$$

where P is an adjustable parameter, the length of periodic windows could be reduced to 0 along with the increase of the parameter, which would make the sequences to produce chaos easily. In the article, P is set to 2.

3) LOGISTIC MAP

The equation of logistic map is shown in Eq. (3.3)

$$x_{k+1} = Px_k(1-x_k) \tag{3.3}$$

where P is an adjustable parameter, in order to generate sequences between (0,1), P is set to 4. When $k \to \infty$, the probability distribution density function of logistic chaotic sequence is Chebyshev type distribution [14]. Similarly, for those test functions with optimal values near the boundary, using logistic map as initial distribution could help heuristic algorithm solve the problem better.

4) PIECEWISE MAP

This kind of piecewise map is modified from Bernoulli shift map [15]. For some test functions whose optimum values are near the midpoint of the definition domain, using this kind of piecewise function as the initial distribution solution can achieve relatively good results. The piecewise map is characterized as follows:

$$x_{k+1} = \begin{cases} \dfrac{x_k}{P} & 0 < x_k < P \\ \dfrac{x_k - P}{0.5 - P} & P \le x_k < 0.5 \\ \dfrac{1 - P - x_k}{0.5 - P} & 0.5 \le x_k < 1 - P \\ \dfrac{1 - x_k}{P} & 1 - P \le x_k < 1 \\ rand\ (0,\ 1) & x_k = 0 or x_k = 1 \end{cases} \tag{3.4}$$

where P is a control parameter whose range is between (0,0.5). In the article, P is set to 0.3.

5) TENT MAP

Tent map is a piecewise linear map with uniform probability density, power spectral density and ideal correlation characteristics. At the same time, there are four small periods exit in the iteration sequence of tent map, the thresholds are (0, 0.2, 0.4, 0.6, 0.8, 1) [16]. The equation of the tent map is formulated as follows:

$$x_{k+1} = \begin{cases} \mu x_k & x_k < 0.5 \\ \mu\ (1 - x_k) & x_k \ge 0.5 \end{cases} \tag{3.5}$$

where μ is set to 2 to make sure the range is (0,1).

In the initial distribution, by mapping different normalized chaotic distributions to the definition domain of each benchmark function, we could get 5 comparison algorithms. The performances of these proposed approaches were tested on CEC2019 benchmark functions [17]. The test results on these 10 benchmark functions indicate that SSA with initial distribution of piecewise map is able to significantly improve the solution quality. These benchmark functions are shown in Table 1.

In order to validate the performance of these algorithms, these experimental results were averaged over fifty independent runs. The average values and standard deviation of 500 iterations are reflected in Table 2. Compared with other comparative algorithms, SSA with piecewise chaotic map could achieve mean optimization four times.

In addition, for the sake of further studying the differences between these methods, the Friedman test was applied to test the average ranking values of these algorithms. Based on the ranking results in Table 3, SSA with piecewise chaotic map has the best performance in solving these test functions of CEC2019.

TABLE 1. The 100-Digit challenge test functions.

No.	Functions	$F_i^* = F_i(x^*)$	Dim	Search Range
1	Storn's Chebyshev Polynomial Fitting Problem	1	9	[-8192,8192]
2	Inverse Hilbert Matrix Problem	1	16	[-16384,16384]
3	Lennard-Jones Minimum Energy Cluster	1	18	[-4,4]
4	Shifted and Rotated Rastrigin's Functio	1	10	[-100,100]
5	Shifted and Rotated Griewank's Functio	1	10	[-100,100]
6	Shifted and Rotated Weierstrass Function	1	10	[-100,100]
7	Shifted and Rotated Schwefel's Function	1	10	[-100,100]
8	Shifted and Rotated Expanded Schaffer's F6 Function	1	10	[-100,100]
9	Shifted and Rotated Happy Cat Function	1	10	[-100,100]
10	Shifted and Rotated Ackley Function	1	10	[-100,100]

TABLE 2. Results of CEC2019 benchmark functions.

Func_No.	Type	SSA	SSA_chebyshev	SSA_iterative	SSA_logistic	SSA_piecewise	SSA_tent
1	Ave	2.1944e06	3.0210e06	4.0020e06	3.5198e06	**1.0554**	2.0385e06
	Std	1.9860e06	2.4829e06	3.1381e06	3.1185e06	**0.30346**	2.4217e06
2	Ave	1307.0179	1745.472	2028.5618	1477.9685	**5.0169**	1575.4535
	Std	866.4961	1391.417	1494.581	1086.4072	**0.35378**	1088.3823
3	Ave	**4.9344**	5.388	5.0416	5.92	5.1224	5.5076
	Std	1.865	2.0296	**1.7497**	2.0418	2.318	1.9924
4	Ave	**25.2746**	31.8719	33.2122	28.3831	27.0496	29.6587
	Std	12.2134	14.4284	12.9524	**10.9756**	13.2118	12.8541
5	Ave	1.1763	1.206	1.2068	**1.1759**	1.1901	1.2061
	Std	0.12675	0.12424	0.12132	**0.09398**	0.11793	0.14469
6	Ave	5.1298	**4.1411**	5.4315	5.9472	5.1814	5.0732
	Std	2.0155	2.075	2.0656	2.0981	**1.8074**	2.0677
7	Ave	1036.0966	938.6637	1031.1713	1089.0476	**924.4324**	1060.8105
	Std	328.4061	301.052	364.7889	383.3683	**243.457**	286.1128
8	Ave	4.2678	4.2737	**4.2327**	4.3115	4.4077	4.2456
	Std	0.30338	0.37078	0.39743	0.31425	**0.25845**	0.47162
9	Ave	1.4228	1.4045	1.4413	1.3891	**1.3427**	1.4164
	Std	0.15989	**0.1457**	0.18367	0.19239	0.15503	0.16084
10	Ave	21.065	21.0295	21.05	21.0165	**21.0155**	21.0334
	Std	0.085318	0.054012	0.10825	0.049681	**0.040905**	0.073926

The above experimental data show that when SSA adopts piecewise chaotic map as its initial distribution, its optimal performance in unimodal function can be improved. At the same time, in other types of test functions, due to the chaos of the initial distribution, the accuracy of the solution set will not deteriorate compared with the random initial distribution of the original algorithm.

Considering that in our engineering application, it is unlikely that the location of the optimal value will appear near the upper or lower bounds of the definition domain, it is reasonable to choose this kind of chaotic map in practical application. Therefore, using piecewise chaotic map as the initial distribution of CISSA has considerable research value.

B. REVISED EXPLORATION MECHANISM

The original SSA sets the threshold to 0.5N, which means the first half of search agents will work for global search in Eq. (2.1), and the other part will follow the former position to optimize the solution in Eq. (2.4), where N is the number of search agents. After a large number of experimental

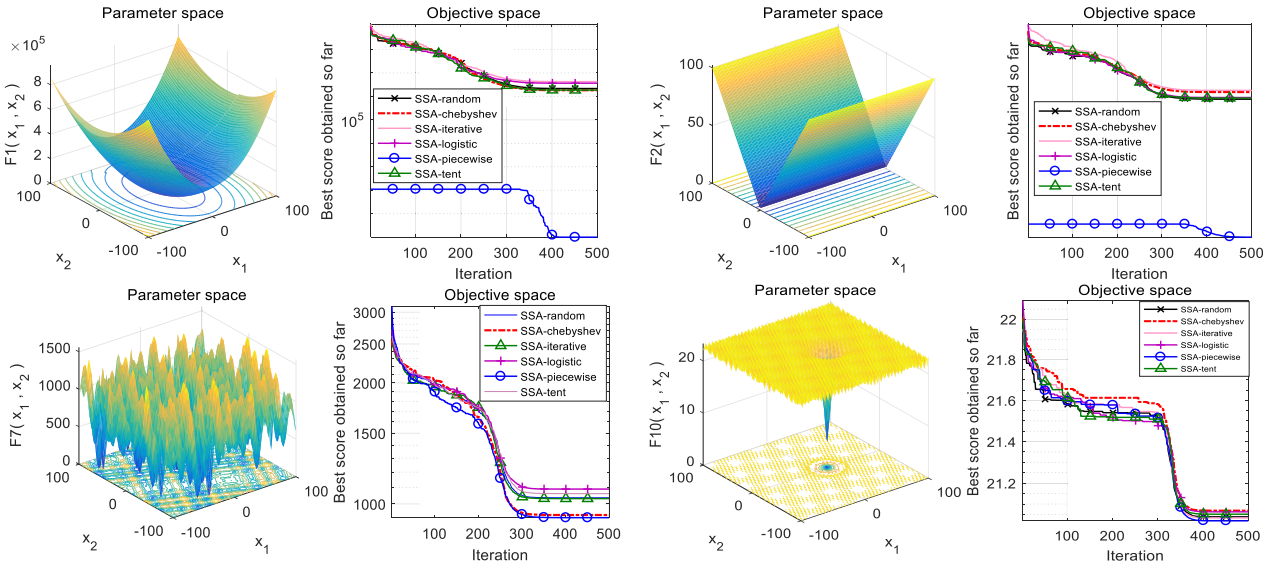

FIGURE 4. Some convergence curves for SSA with different initial distributions.

TABLE 3. Average ranking values of various SSA with different initial distribution by using friedman test.

Algorithm	Ranking
SSA	3.0
SSA_chebyshev	3.5
SSA_iterative	4.6
SSA_logistic	3.9
SSA_piecewise	**2.3**
SSA_tent	3.7

comparisons, we found the threshold value 0.5N wasn't the best choice for most benchmark functions.

In the article, A two-stage exploration mechanism was proposed to replace Eq. (2.1). In the first stage, the exploration mechanism of SSA was maintained just to ensure the powerful exploring ability of the original algorithm, based on a large number of test comparisons on different types of benchmark functions, we set the first threshold to $\lfloor N/3 \rfloor$. The improved algorithm could keep the exploration ability of the original SSA during this phase. In the second stage, we set the second threshold to $\lfloor 2N/3 \rfloor$, the original exploration step length could be set to a more refined value in Eq. (3.6).

$$X_j^{i+1} = \begin{cases} X_j^1 + c_1\left(2c_2X_j^1 - X_j^i\right) & 0 \leq c_3 < 0.5 \\ X_j^1 - c_1\left(2c_2X_j^1 - X_j^i\right) & 0.5 \leq c_3 \leq 1 \end{cases} \tag{3.6}$$

where $i \geq 2$, X_j^i shows the position of i-th follower salp in j-th dimension, X_j^1 is the position of leader salp in the j-th dimension, c_2, c_3 are random numbers. Compared with the original location update formula, the shorter variable exploration steps of Eq. (3.6) are able to find optimum solutions with higher accuracy.

TABLE 4. Unimodal benchmark functions.

Function	Dim	Range	f_{min}
$F_1(x) = \sum_{i=1}^{n} x_i^2$	30	[-100, 100]	0
$F_2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	30	[-10, 10]	0
$F_3(x) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$	30	[-100, 100]	0
$F_4(x) = max_i\{ \lvert x_i \rvert,\ 1 \leq i \leq n\}$	30	[-100, 100]	0
$F_5(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	30	[-30, 30]	0
$F_6(x) = \sum_{i=1}^{n} ([x_i + 0.5])^2$	30	[-100, 100]	0
$F_7(x) = \sum_{i=1}^{n} ix_i^4 + random[0,1)$	30	[-1.28, 1.28]	0

C. REVISED NONLINEAR CONVERGENCE COEFFICIENT c_1

As mentioned above, the two-stage exploratory mechanism set the threshold value of exploration phase to $\lfloor 2N/3 \rfloor$, the enlarged threshold created a demand for slower decreasing nonlinear convergence coefficient c_1, in order to make the parameter reduce nonlinearly from 1 to 0 in the new domain of definition, c_1 is revised in Eq. (3.7):

$$c_1 = (0.7 + 0.3sin\,(2l))\, e^{-\left(\frac{8l}{3L}\right)^2} \tag{3.7}$$

where l is the current iteration and L is the maximum number of iterations.

Compared with original coefficient c_1, we altered the index of c_1 to slow down the convergence process. In the meantime, the lack of perturbation could lead to the lack of creativity

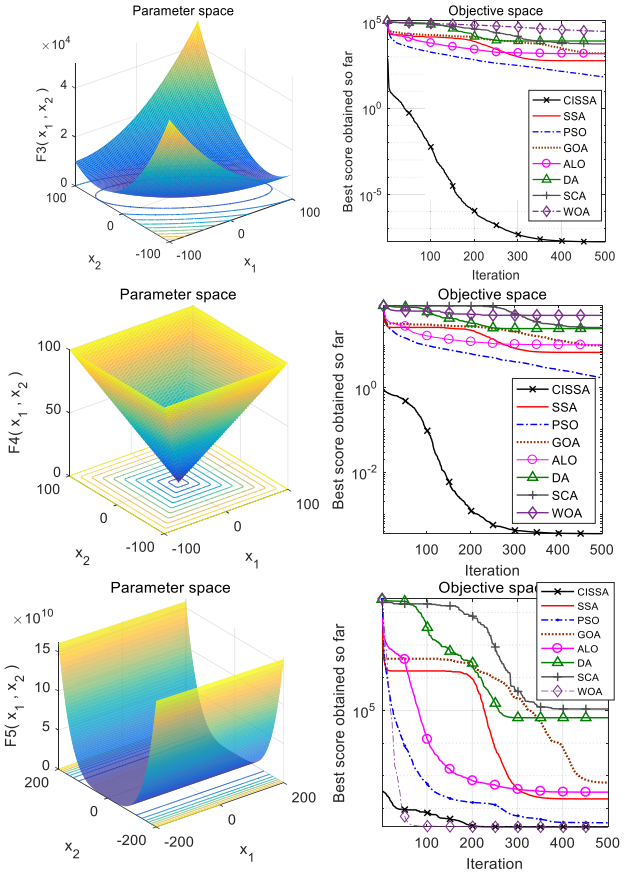

FIGURE 5. Convergence curves for algorithms over some unimodal benchmark functions.

during the search iterations. To remedy this defect, damping motion is added in the convergence coefficient c_1 to increase perturbation. After multiple tests on different test functions, the weight ratio of damping motion was set to 0.3.

D. SPIRAL UPDATING POSITION

Original SSA updates the positions of follower salps by uniformly acceleration relation, and at the end it can simplify the solution to a weighted sum in Eq. (2.4). However, we can see that the shape of the salp chain in Figure 1 is more similar to spiral line. To make the model be more exact in mathematics and clear in physics, when serial numbers of follower salps are between $\lfloor 2N/3 \rfloor + 1$ to N, Eq. (3.8) would be used to replace Eq. (2.4) in our mathematical model. These follower salps spin around the leader by using a shrinking circle or a spiral shaped path [18], and Eq. (3.8) is calculated as follows:

$$X_j^i = \left| X_j^1 - X_j^i \right| e^l cos\left(2\pi l\right) + X_j^1 \tag{3.8}$$

where $\left| X_j^1 - X_j^i \right|$ indicates the distance of i-th follower salp to the position of the first salp in the j-th dimension.

The new approach was physically closer to the trajectory of salp swarm. In each dimension, every follower salps moved in a kind of spiral motion around the leader salp, the magnitude of the spiral line was determined by the distance between the follower salp and the leader salp.

E. TIME COMPLEXITY OF CISSA

The time complexity of SSA and the CISSA depends on the number of iterations (L) and the number of search agents (N). The overall time complexity of SSA and CISSA is combined with these factors [9]:

1. Initializing all search agents.
2. Calculating the fitness of each search agents.
3. Selecting the best one from by fitness.
4. Calculating the fitnesses of all search agents in each iteration.
5. Updating the positions of all search agents in each iteration.

The time complexity of initializing with random number or piecewise chaotic map is $O(N)$. And the time complexities of all remaining steps are also $O(N)$. Hence, the resulting time complexities of SSA and CISSA are as shown in Eq. (3.9):

$$\begin{aligned} O(SSA) &= O(CISSA) \\ &= O(N) + O(N) + O(N) + L \times (O(N) + O(N)) \\ &= (2L + 3) \times O(N) \end{aligned} \tag{3.9}$$

As we can see in the time complexity calculation formula, the improved CISSA has the same time complexity compared with the original SSA. Although some transformative genetic algorithms can sacrifice time complexity to get better solutions, however, in the case of large-scale redundant container deployment problem, because of the slow solving process and the instability of the devices, we need a solution with lower time complexity. In the experimental comparison of the next chapter, it can be found that CISSA are capable of achieving better solutions than SSA in most benchmark functions. Therefore, the proposed CISSA has important value in engineering application.

IV. COMPARISON OF EXPERIMENTAL RESULTS

In this work, in order to better compare the performance differences with the original SSA, we chose the same test functions used by the author of SSA.The improved algorithm CISSA was benchmarked on 29 test functions with 7 metaheuristic algorithms, including the original salp swarm algorithm (SSA) [4], whale optimization algorithm (WOA) [18], grasshopper optimization algorithm (GOA) [19], dragonfly algorithm (DA) [20], sine cosine algorithm (SCA) [21], ant lion optimizer (ALO) [22] and a classical heuristic algorithm particle swarm optimization (PSO) [23]. Each comparison algorithm performed well in certain areas, the implementation details of these algorithms were set as the papers [18]–[23] describe and the key parameters of these comparison algorithms were shown in Table 5.

These comparison algorithms have good performance in certain areas. WOA performs well in unimodal functions.

TABLE 5. Results of unimodal benchmark functions.

Func	Type	CISSA	SSA	PSO	GOA	ALO	DA	SCA	WOA
	Ave	1.5634e-21	2.1493e-08	1.3295e-07	3.6997	1.0062e-04	690.2741	0.1853	**2.2418e-85**
F_1	Std	1.5676e-21	4.3023e-09	2.2219e-07	2.2383	7.639e-05	391.7163	0.5019	**7.0538e-85**
	Best	1.8966e-22	1.4767e-08	5.0672e-09	1.0485	3.5101e-05	18.0476	0.01261	**7.5604e-94**
	Worst	4.8083e-21	3.0263e-08	7.179e-07	8.046	3.0036e-04	1286.6813	3.2457	**2.2317e-84**
	Ave	6.41e-12	0.57613	5.7437e-03	13.9631	51.3154	12.7211	1.4531e-02	**1.9331e-53**
F_2	Std	5.2858e-12	0.81178	1.6384e-02	31.0106	49.8964	7.9679	2.3361e-02	**5.4905e-53**
	Best	6.7697e-13	8.4114e-03	5.726e-05	1.9801	0.26207	2.1563	8.4305e-05	**3.6816e-61**
	Worst	1.728e-11	2.6945	5.2359e-02	102.0169	114.4436	25.8669	7.9335e-02	**1.7488e-52**
	Ave	**1.7123e-08**	599.7873	67.1078	1626.0016	1556.1559	8461.9786	5681.3771	31055.6144
F_3	Std	**3.2212e-08**	315.4712	24.0251	589.3629	542.4914	5956.3839	5661.058	8362.8364
	Best	**5.1866e-12**	140.6055	38.5589	901.6816	637.7909	825.4592	344.0423	17356.9282
	Worst	**9.9451e-08**	1234.2766	104.2286	2760.3188	2196.1845	17798.461	19235.3112	44439.1028
	Ave	**3.5173e-04**	6.8268	1.6793	10.0106	10.2957	24.9477	26.4767	51.236
F_4	Std	**4.9966e-04**	2.644	0.4254	3.1368	2.5859	9.1063	6.6735	30.9855
	Best	**1.2409e-06**	2.9839	1.0261	2.9439	7.0206	10.5737	16.4299	1.4965
	Worst	**1.2535e-03**	12.0446	2.3054	13.6243	13.2154	42.8419	35.5021	89.2581
	Ave	**27.2086**	195.799	36.4128	624.2412	315.1049	5752.4301	1.1065e05	27.2253
F_5	Std	**0.41997**	287.1828	25.5049	523.0412	380.2442	68629.3662	3.2993e05	0.85174
	Best	25.9746	21.798	**15.59**	204.5865	28.0971	4428.0611	50.007	26.7976
	Worst	**27.8929**	967.1992	90.6207	1658.1765	1144.2287	217812.646	1.049.e06	28.7891
	Ave	0.10317	**2.0941e-08**	4.288e-08	4.3867	1.3667e-04	912.3345	8.3867	0.095728
F_6	Std	0.0322	**6.3746e-09**	4.5926e-08	3.9718	5.9304e-05	771.5354	5.3226	0.070031
	Best	0.070493	**1.2103e-08**	1.334e-09	0.5519	5.4194e-05	121.5389	4.6469	0.01735
	Worst	0.17037	**3.4766e-08**	1.3533e-07	12.7758	2.2406e-04	2653.1739	22.7799	0.23919
	Ave	**2.29103e-03**	0.109	0.019639	0.026068	0.1601	0.24394	0.044349	3.8079e-03
F_7	Std	**2.1327e-03**	0.048845	7.8709e-03	9.8314e-03	0.0666	0.14049	0.037054	2.29412e-03
	Best	**1.4663e-04**	0.05719	7.241e-03	0.014109	0.058673	0.04193	0.011497	9.2932e-04
	Worst	**7.2938e-03**	0.211002	0.036706	0.047031	0.25238	0.4954	0.12282	8.5414e-03

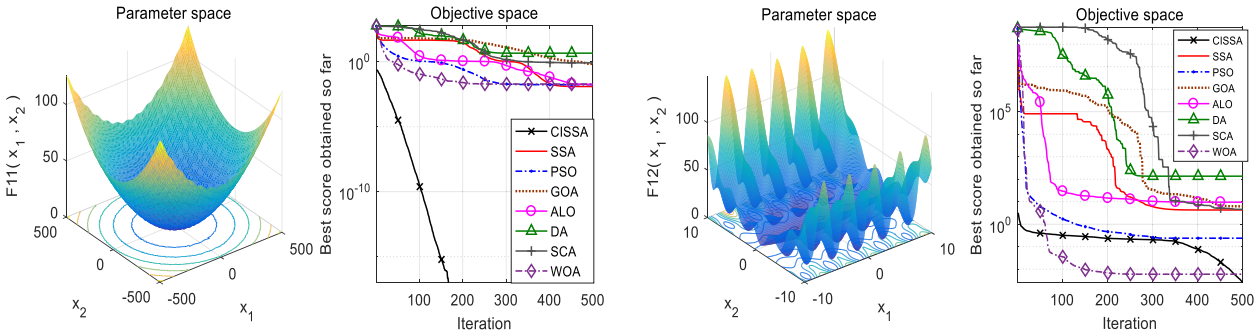

FIGURE 6. Convergence curves for algorithms over some multimodal benchmark functions.

PSO, as a classical comparison algorithm, has excellent comprehensive performance. SCA performs well in stability, and GOA, ALO, DA are proposed in recent years, they are outstanding when dealing with some multimodal test functions and composite test functions.

These 29 benchmark functions are divided into 4 types, which are used to test search ability of the proposed algorithm. These functions are listed in Table 4, Table 7, Table 8 and Table 10, where dim indicates dimension of each function. Search space of each function are limited by domain

Algorithm 1 Pseudo-Code of Comprehensive Improved Salp Swarm Algorithm

(1) Initialize the N salp swarm population $X_i(i = 1, 2, \ldots, N)$ with chaotic map by Eq. (3.4)
(2) Calculate the fitness of each search agent and mark the target position
(3) **while**(current iteration < maximum iteration number)
(4) **for** each search agent
(5) Update the random number c_2, c_3
(6) Set the nonlinear convergence coefficient c_1 by Eq. (3.7)
(5) **for1** $0 < i < N/3$
(7) Update the position with longer step length in exploration phase by Eq. (2.1)
(8) Calculate the fitness of current search agent
(10) **end for1**
(11) **for2** $0.2N < i < 2N/3$
(12) Update the position with refined step length in exploration phase by Eq. (3.6)
(13) Calculate the fitness of current search agent
(14) **end for2**
(15) **for** $2N/3 < i < N$
(16) Update the position with spiral motion formula in exploitation phase by Eq. (3.8)
(17) Calculate the fitness of current search agent
(18) **end for3**
(19) **if** position of salp is not in the domain of definition
(20) Modify the position to the upper or lower limit of the domain
(21) **end if**
(22) Choose the best fitness of all search agents, replace target fitness if it is a better solution
(23) Record position of target
(24) **end for**
(22) **end while**
(23) current iteration+1
(24) end while
(25) return target fitness and target position

TABLE 6. Main parameters of comparison algorithms.

Algorithm	Main parameters
SSA	$c_1 = 2e^{-(\frac{4l}{L})^2}$
PSO	$c_1 = c_2 = 1.4961, \omega = 0.7298$
GOA	$c_{max} = 1, c_{min} = 0.00001$
ALO	$I = 10^{\omega}\frac{t}{T}, \omega = \begin{cases} 1 & 0 < t \le 0.1T \\ 2 & 0.1T < t \le 0.5T \\ 3 & 0.5T < t \le 0.75T \\ 4 & 0.75T < t \le 0.9T \\ 5 & 0.9T < t \le 0.95T \\ 6 & 0.95T < t \le T \end{cases}$
DA	$\beta = 1.5, \omega = 0.9 - 0.2, s = 0.1,$ $a = 0.1, c = 0.7, f = 1, e = 1$
SCA	$a = 2, r_1 = a\left(1 - \frac{t}{T}\right), r_2 = 2\pi * rand$ $r_3 = 2 * rand, r_4 = rand$
WOA	$a = 2\left(1 - \frac{t}{T}\right), b = 1, l = 2 * rand - 1$

of definition $[lb_j, ub_j]$, and f_{min} is the optimum value of each test function. In F1-F23, 30 search agents were employed by each algorithm to conduct optimization over 500 iterations, these algorithms were tested for 50 times. In F24-F29, because of the complexity of benchmark functions, each algorithm ran 100 iterations with 30 search agents, and these algorithms would be tested for 30 times.

A. EVALUATION OF EXPLOITATION ABILITY

The first type of benchmark functions is unimodal functions, which has only one global optimum. In Table 4, the unimodal functions F1-F7 are designed to test the exploitation ability of the algorithms. When an algorithm finds a more precise solution close to the global optimum, it indicates that the algorithm has a stronger exploitation ability.

Comparison data are shown in Table 5. In the table, CISSA works best in 4 of the 7 unimodal benchmark tests, and reaches second place twice. As for standard deviation and worst value, CISSA also perform better than the other algorithms in most cases, which means the proposed algorithm is able to produce an optimal solution with stability.

B. EVALUATION OF EXPLORATION ABILITY

The second and third types of benchmark functions are multimodal functions which own several local optimums. In Table 7 and Table 8, these multimodal functions F8-F23 are designed for evaluating the exploration capability of the algorithms. In the solving process, the solutions may fall into local optimums frequently. Only when the algorithms have excellent exploration abilities, can they avoid involving local optimums as much as possible.

According to the results in Table 9 and Table 11, CISSA outperforms all the other comparison algorithms in terms of average values in 10 of 16 benchmark tests and achieves 12 best values, as for the worst case, CISSA also performs

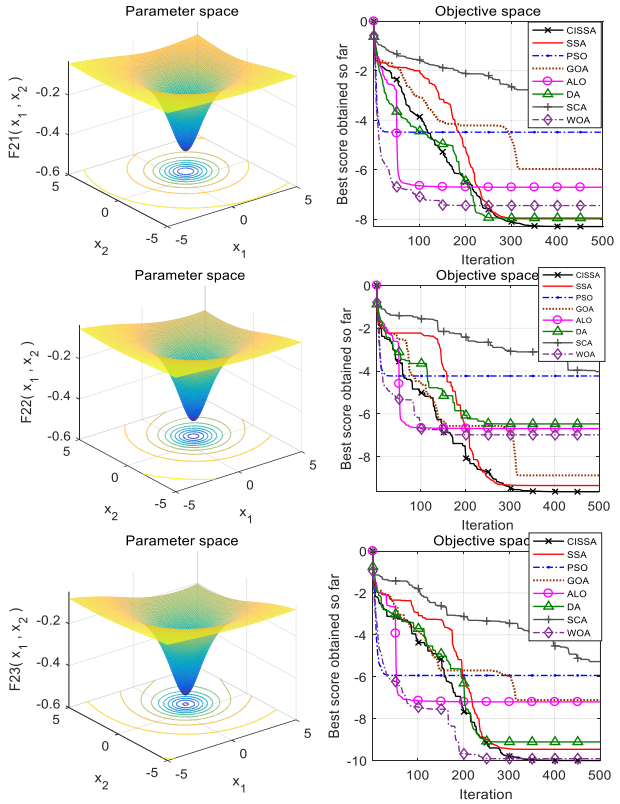

FIGURE 7. Convergence curves for algorithms over some fixed-dimension multimodal benchmark functions.

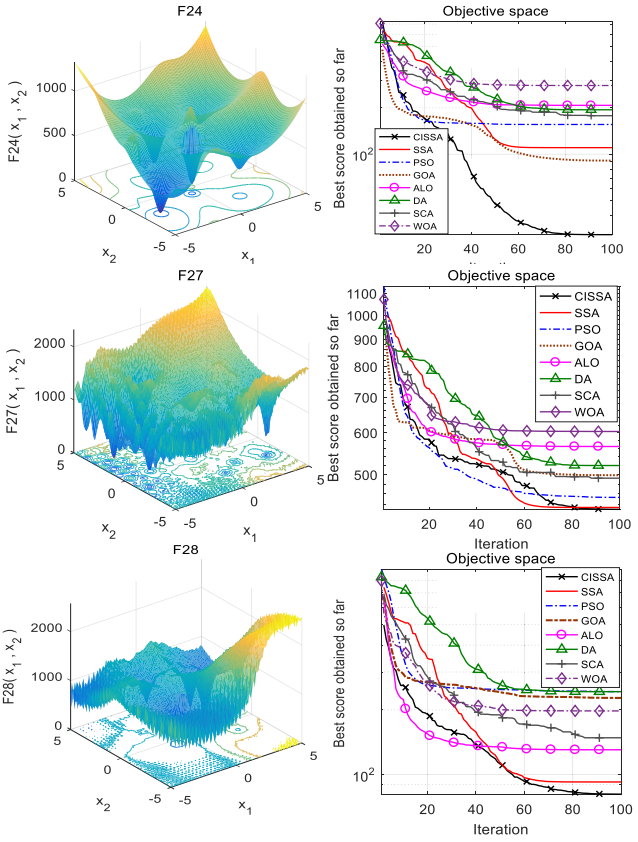

FIGURE 8. Convergence curves for algorithms over some composite benchmark functions.

TABLE 7. Multimodal benchmark functions.

Function	Dim	Range	f_{min}
$F_8(x)=\sum_{i=1}^{n}-x_i\,sin(\sqrt{\lvert x_i\rvert})$	30	[-500, 500]	418.98 29*Dim
$F_9(x)=\sum_{i=1}^{n}[x_i^2-10\,cos(2\pi x_i)+10]$	30	[-5.12, 5.12]	0
$F_{10}(x)=-20\,exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2})-$ $exp(\frac{1}{n}\sum_{i=1}^{n}cos(2\pi x_i))+20+e$	30	[-32, 32]	0
$F_{11}(x)=\frac{1}{4000}\sum_{i=1}^{n}-\prod_{i=1}^{n}cos(\frac{x_i}{\sqrt{i}})+1$	30	[-600, 600]	0
$F_{12}(x)=\frac{\pi}{n}\{10\,sin^2(\pi y_1)+$ $\sum_{i=1}^{n-1}(y_i-1)^2[1+10sin^2(\pi y_{i+1})]$ $+(y_n-1)^2\}+\sum_{i=1}^{n}u(x_i,10,100,4)$ $y_i=1+\frac{x_i+1}{4}\quad u(x_i,a,k,m)$ $=\begin{cases}k(x_i-a)^m & x_i>a\\ 0 & -a<x_i<a\\ k(-x_i-a)^m & x_i<-a\end{cases}$	30	[-50, 50]	0
$F_{13}(x)=0.1\{sin^2(3\pi x_1)+$ $\sum_{i=1}^{n}(x_i-1)^2[1+sin^2(3\pi x_i+1)]$ $+(x_n-1)^2[1+sin^2(2\pi x_n)]\}$ $+\sum_{i=1}^{n}u(x_i,\,5,\,100,\,4)$	30	[-50, 50]	0

TABLE 8. Fixed-dimension multimodal benchmark functions.

Function	Dim	Range	f_{min}
$F_{14}(x)=(\frac{1}{500}+\sum_{j=1}^{25}\frac{1}{j+\sum_{i=1}^{2}(x_i-a_{ij})^6})^{-1}$	2	[-65, 65]	0
$F_{15}(x)=\sum_{i=1}^{11}[a_i-\frac{x_1(b_i^2+b_ix_2)}{b_i^2+b_ix_3+x_4}]^2$	4	[-5, 5]	0.00030
$F_{16}(x)=4x_1^2-2.1x_1^4+\frac{1}{3}x_1^6+x_1x_2-4x_2^2$ $+4x_2^4$	2	[-5, 5]	-1.0316
$F_{17}(x)=(x_2-\frac{5.1}{4\pi^2}x_1^2+\frac{5}{\pi}x_1-6)^2$ $+10(1-\frac{1}{8\pi})cos\,x_1+10$	2	[-5, 5]	0.398
$F_{18}(x)=[1+(x_1+x_2+1)^2(19-14x_1+3x_1^2$ $-14x_2+6x_1x_2+3x_2^2)]\times[30+(2x_1-3x_2)^2$ $\times(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)]$	2	[-2, 2]	3
$F_{19}(x)=-\sum_{i=1}^{4}c_i\,exp(-\sum_{j=1}^{3}a_{ij}(x_j-p_{ij})^2)$	3	[1, 3]	-3.86
$F_{20}(x)=-\sum_{i=1}^{4}c_i\,exp(-\sum_{j=1}^{6}a_{ij}\,(x_j-p_{ij})^2)$	6	[0, 1]	-3.32
$F_{21}(x)=-\sum_{i=1}^{5}[(X-a_i)(X-a_i)^T+c_i]^{-1}$	4	[0, 10]	-10.1532
$F_{22}(x)=-\sum_{i=1}^{7}[(X-a_i)(X-a_i)^T+c_i]^{-1}$	4	[0, 10]	-10.4028
$F_{23}(x)=-\sum_{i=1}^{10}[(X-a_i)(X-a_i)^T+c_i]^{-1}$	4	[0, 10]	-10.5363

best in 12 benchmark tests. At the same time, CISSA behaves better than the original algorithm in 12 sets of data, which means the proposed CISSA behaves better in exploration phase compared with the original SSA.

C. EVALUATION OF AVOIDING LOCAL OPTIMUMS

Composite functions are the combination of several some basic benchmark functions with same domain of definition [24]. Compared with the multimodal test functions, this kind of functions have much more local optimal solutions,

TABLE 9. Results of multimodal benchmark functions.

Func	Type	CISSA	SSA	PSO	GOA	ALO	DA	SCA	WOA
F_8	Ave	-7862.1735	-7013.2794	-6513.9984	-7692.1739	-5503.8875	-5488.4604	-4027.3611	**-10518.0296**
	Std	689,9675	762.5983	754.2618	758.6933	**65.2731**	574.0214	282.3384	1871.2576
	Best	-9521.2833	-8735.6871	-7682.9305	-8695.9302	-5751.2842	-6743.3691	-4985.6271	**-12569.0502**
	Worst	-6249.8003	-5304.9362	-5033.2861	-6031.5517	-5403.9675	-4502.6918	-3118.2463	**-7562.2803**
F_9	Ave	0.40369	44.2756	45.2777	86.5068	69.9456	165.532	36.0698	**5.6843e-15**
	Std	0.97473	15.2289	15.3878	22.6649	15.126	32.6436	45.1902	**1.7975e-14**
	Best	**0**	28.8538	29.8488	51.8451	49.748	92.7266	1.116e-03	0
	Worst	3.0265	76.6117	74.6928	121.5069	87.5563	210.8159	153.4426	**5.6843e-14**
F_{10}	Ave	**2.3093e-14**	1.7417	1.1296	3.7358	2.1993	7.4515	17.2497	4.4519e-14
	Std	**1.1782e-14**	1.1229	1.2903	0.58767	0.62255	1.8275	6.2695	1.8346e-14
	Best	1.6606e-14	3.5226e-05	2.6161e-05	2.7586	1.64663	5.279	2.2611e-02	**8.8818e-16**
	Worst	**3.1117e-14**	3.5177	3.1582	4.4675	3.4617	11.0779	20.3186	4.4409e-14
F_{11}	Ave	**0**	0.12539	0.017431	0.69623	0.018197	4.6281	0.82824	0.019449
	Std	**0**	0.011164	0.018222	0.16024	0.010298	1.9089	0.3256	0.041441
	Best	**0**	..3545e-06	1.293e-08	0.56481	3.5989e-03	2.2632	0.26164	0
	Worst	**0**	0.029707	0.044058	1.0766	0.035149	9.1856	1.1421	0.11
F_{12}	Ave	**2.6674e-03**	4.279	0.2389	6.2375	9.5421	136.5521	4.5883	5.9864e-03
	Std	**2.3245e-03**	2.4211	0.41579	2.5468	3.0934	386.6621	3.3128	5.3882e-03
	Best	5.9608e-04	1.5457	**7.4629e-10**	3.2373	5.0669	3.3231	1.2458	9.2701e-04
	Worst	**7.7207e-03**	8.9211	1.0406	9.8671	13.1219	1236.2374	11.2189	0.019135
F_{13}	Ave	0.18562	4.7385	**0.11028**	14.3416	4.3452	11648.2322	7632.2918	0.19281
	Std	**0.10497**	12.39	0.38859	20.3789	13.382	20732.7869	22561.7784	0.13032
	Best	0.38596	1.83e-05	**2.063e-09**	0.34126	0.014011	20.9492	2.9854	0.065653
	Worst	**0.42339**	39.7499	2.1804	51.9456	42.4295	60655.8721	71726.8536	0.43455

at the same time, it's hard to find the correlations among local optimal solutions and global optimal solution. Only when the algorithms strike the balance between exploration ability and exploitation ability, can they find solutions closer to the global optimal solutions.

The results in Tables 12 show that compared with other 7 meta-heuristic algorithms, CISSA still outperforms other algorithms on F24, F27, F28, and it ranks second among the other test functions, which means the proposed algorithm provides very competitive performance in the composite functions.

D. SIGNIFICANCE OF THE RESULTS

Simply comparing the average values couldn't reflect the difference between the results of each algorithm. Heuristic algorithms have some regularity in the results of test function, however, the distribution laws of these results are complex and they can't fully fit the normal distribution [25]. Therefore, researchers usually use non-parametric methods to statistic the results [26]. As two commonly-used methods of non-parametric statistics, Wilcoxon test and Friedman test are widely used in many areas.

In the article, in order to judge whether the results of CISSA were significantly different from results of other algorithms, the Wilcoxon test with 0.05 significant level was applied to investigate the statistical significant differences [27]. The result calculated in this way is marked as p-value. In this work, p-values were calculated based on averaged values between CISSA and the other algorithms.

When p-value is less than 0.05, it could be considered that difference between two samples is significant by definition. The results shown in Table 13 indicate that p-values between CISSA and other algorithms are less than 0.05 in most cases, which means the solutions of CISSA and the other could be regarded uncorrelated.

Meanwhile, we ranked the average performance of all algorithms by Friedman test [28]. The ranking results are as shown in Table 4 11, according to these statistical results, we can see that by improving the original SSA in 4 steps, CISSA can achieve better performances than the other comparison algorithms in all kinds of test functions.

E. COMPARISON OF CONVERGENCE PERFORMANCE BETWEEN CISSA AND SSA

In the chapter, we chose Page's trend test to compare the convergence performance between CISSA and the original SSA [29]. The procedure works by considering the null hypothesis of equality between the 10 treatments analyzed, which can be rejected in favor of an ordered alternative [30].

TABLE 10. Composite benchmark functions.

Function	Dim	Range	f_{min}
$F_{24}(CF1)$			
$f_1, f_2, f_3, \ldots, f_{10} = Sphere\ Function,$	30	[-5, 5]	0
$[\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_{10}] = [1,\ 1,\ 1, \ldots,\ 1]$			
$[\lambda_1, \lambda_2, \lambda_3\ \ldots,\ \lambda_{10}] = [5/100,\ 5/100,\ 5/100,\ \ldots,\ 5/100]$			
$F_{25}(CF2)$			
$f_1, f_2, f_3, \ldots, f_{10} = Griewank's\ Function,$	30	[-5, 5]	0
$[\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_{10}] = [1,\ 1,\ 1,\ \ldots,\ 1]$			
$[\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_{10}] = [5/100,\ 5/100,\ 5/100,\ \ldots,\ 5/100]$			
$F_{26}(CF3)$			
$f_1, f_2, f_3, \ldots, f_{10} = Griewank's\ Function$	30	[-5, 5]	0
$[\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_{10}] = [1,\ 1,\ 1,\ \ldots,\ 1]$			
$[\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_{10}] = [1,\ 1,\ 1,\ \ldots,\ 1]$			
$F_{27}(CF4)$			
$f_1, f_2 = Ackley's\ Function,$			
$f_3, f_4 = Rastrigin's\ Function,$			
$f_5, f_6 = Weierstrass\ Function,$			
$f_7, f_8 = Griewank's\ Function,$	30	[-5, 5]	0
$f_9, f_{10} = Sphere\ Function,$			
$[\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_{10}] = [1,\ 1,\ 1, \ldots,\ 1]$			
$[\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_{10}] = [5/32,\ 5/32,\ 1,\ 1,\ 5/0.5,$			
$5/0.5,\ 5/100,\ 5/100,\ 5/100,\ 5/100]$			
$F_{28}(CF5)$			
$f_1, f_2 = Rastrigin's\ Function,$			
$f_3, f_4 = Weierstrass\ Function,$			
$f_5, f_6 = Griewank's\ Function,$			
$f_7, f_8 = Ackley's\ Function,$	30	[-5, 5]	0
$f_9, f_{10} = Sphere\ Function,$			
$[\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_{10}] = [1,\ 1,\ 1,\ \ldots,\ 1]$			
$[\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_{10}] = [1/5,\ 1/5,\ 5/0.5,\ 5/0.5,\ 5/100,$			
$5/100,\ 5/32,\ 5/32,\ \ 5/100,\ \ 5/100]$			
$F_{29}(CF6)$			
$f_1, f_2 = Rastrigin's\ Function,$			
$f_3, f_4 = Weierstrass\ Function,$			
$f_5, f_6 = Griewank's\ Function,$			
$f_7, f_8 = Ackley's\ Function,$			
$f_9, f_{10} = Sphere\ Function$	30	[-5, 5]	0
$[\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_{10}]$			
$= [0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.5,\ 0.6,\ 0.7,\ 0.8,\ 0.9,\ 1]$			
$[\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_{10}] = [0.1 * 1/5,\ 0.2 * 1/5,\ 0.3 * 5/0.5,$			
$0.4 * 5/0.5,\ 0.5 * 5/100, 0.6 * 5/100,\ 0.7 * 5/32,\ 0.80.9$			
$* 5/32, * 5/100,\ 1 * 5/100]$			

The average values of CISSA and SSA on 23 test functions were used as input data in Page's trend test. Table 4.12 shows the treatments' ranks computed for the (CISSA-SSA) differences in fitness values. For completeness, the relevant data

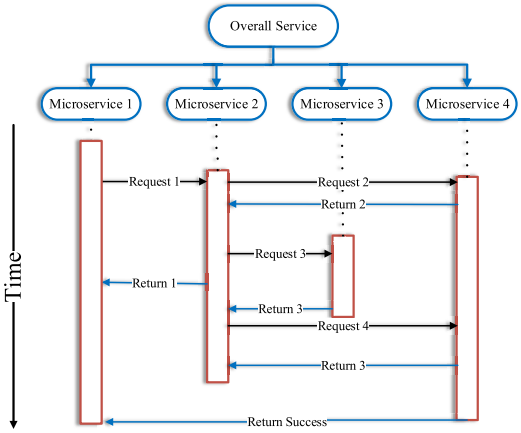

FIGURE 9. Process diagram of microservice architecture.

of the opposite comparison (SSA-CISSA) is also included. On each test function, the convergence curves of the two algorithms were divided into 10 equal parts, and the slope of tangent between adjacent equal points were calculated. Meanwhile, the test set the ranks' values from 1 to 10 according to the order of slope from small to large [31].

By adding the k-th slopes' ranking values of each test function, we could get the k-th parameter of differences in Table 15. After obtaining these values, the Page L statistic can be computed using Eq. (4.1):

$$L = \sum_{k=1}^{10} kC_k \tag{4.1}$$

where C_k is the sum of k-th slopes' ranking values of all test functions, where $1 \leq k \leq 10$.

The comparison (CISSA-SSA) shows roughly increasing trend in the ranks, which is confirmed by a very low p-value. At the same time, the opposite comparison (SSA-CISSA), shows clearly that the ranks are not increasing, which is rejected by a p-value near to 1.0. These results show that the CISSA is converging faster than the original SSA.

V. REDUNDANT CONTAINER DEPLOYMENT MODEL

In microservice architecture, the overall service is divided into multiple microservices with corresponding functions. Microservices call each other though lightweight communication mechanisms [1]. A simple example diagram is shown in Figure 9, in the diagram, the red square represents the completion time for series of tasks on each microservice, and the dotted line represents the start time.

The total completion time depends on the microservice with the longest completion time and the total communication overhead [32]. For fixed business service, the number of calls between microservices is constant [33].

Simultaneously, considering that all microservices work on devices within the same LAN, communication overhead between devices could be assumed to be the same [34]–[36].

Figure 10 displays an example of microservice scheme. In the diagram, each microservice runs in the separate container. For example, microservice 2 runs in container 2, which

TABLE 11. Results of fixed-dimension multimodal benchmark functions.

Func	Type	CISSA	SSA	PSO	GOA	ALO	DA	SCA	WOA
F_{14}	Ave	1.0973	1.2853	3.1136	**0.998**	1.9565	1.4277	1.1971	1.8197
	Std	0.39953	0.31258	2.9817	**5.0824e-16**	1.1175	0.88946	0.60516	1.8725
	Best	**0.998**	0.998	0.998	0.998	0.998	0.998	0.998	0.998
	Worst	2.9821	1.992	11.7187	**0.998**	4.9505	3.9683	2.9821	10.7632
F_{15}	Ave	9.6215e-04	2.9635e-03	**6.1483e-04**	8.3279e-03	3.6735e-03	1.3971e-03	6.5871e-03	9.0263e-04
	Std	**4.4982e-04**	7.5627e-04	5.8846e-03	7.9179e-04	9.5532e-03	6.5757e-03	5.7726e-04	6.2958e-04
	Best	**3.0749e-04**	3.7476e-04	3.6295e-04	3.0823e-04	4.3143e-04	4.7354e-04	4.2984e-04	3.1366e-04
	Worst	**1.0583e-03**	9.0597e-03	1.9753e-03	0.020367	0.020513	0.019222	1.7815e-03	1.6624e-03
F_{16}	Ave	**-1.0316**	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316
	Std	2.2056e-10	**1.8467e-16**	6.5195e-16	5.5669e-13	1.5017e-13	3.0768e-11	3.1238e-05	9.7031e-11
	Best	**-1.0316**	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316
	Worst	**-1.0316**	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316
F_{17}	Ave	**0.39789**	0.39789	0.39789	0.39789	0.39789	0.39789	0.39789	0.39789
	Std	1.8355e-10	9.3729e-14	**0**	8.6237-13	1.2135e-13	8.2854e-07	1.2863e-03	2.3657e-05
	Best	**0.39789**	0.39789	0.39789	0.39789	0.39789	0.39789	0.39789	0.39789
	Worst	**0.39789**	0.39789	0.39789	0.39789	0.39789	0.39789	0.40265	0.39789
F_{18}	Ave	**3**	3	3.9	8.4	3	3	3	3.0001
	Std	7.1321e-07	**2.3159e-13**	5.1089	21.3316	3.5836e-13	3.5784e-06	1.5182e-04	1.4069e-04
	Best	**3**	3	3	3	3	3	3	3
	Worst	**3**	3	3	84	3	3	3.0003	3.0006
F_{19}	Ave	**-3.8628**	-3.8628	-3.8628	-3.7148	-3.8628	-3.8627	-3.8551	-3.8597
	Std	7.1328e-06	7.5284e-12	**2.7925e-15**	0.3114	1.2237e-12	2.6491e-04	2.8834e-03	9.7328e-03
	Best	**-3.8628**	-3.8628	-3.8628	-3.8628	-3.8628	-3.8628	-3.8628	-3.8628
	Worst	**-3.8628**	-3.8628	-3.8628	-2.7125	-3.8628	-3.8615	-3.8622	-3.8611
F_{20}	Ave	**-3.2789**	-3.2238	-3.2743	-3.2778	-3.2704	-3.2383	-2.9778	-3.2745
	Std	0.059643	0.062036	0.04225	0.061051	0.061197	0.092459	**0.025686**	0.094857
	Best	**-3.322**	-3.322	-3.322	-3.322	-3.322	-3.322	-3.2596	-3.3216
	Worst	-3.1518	-3.1519	-3.1376	-3.1859	**-3.2019**	-2.9165	-2.0354	-3.0735
F_{21}	Ave	**-8.2967**	-7.982	-4.4834	-5.9692	-6.7064	-7.9522	-3.1166	-7.4474
	Std	**2.7312**	3.2007	2.7712	3.3753	2.9855	2.7832	2.9923	3.0323
	Best	**-10.1532**	-10.1532	-10.1532	-10.1532	-10.1532	-10.1532	-6.5391	-10.1528
	Worst	**-2.6305**	-2.6305	-2.6305	-2.6305	-2.6305	-2.6305	-0.49726	-2.6294
F_{22}	Ave	**-9.7261**	-9.5517	-4.1653	-8.8142	-6.6108	-6.4231	-3.2905	-6.8745
	Std	**2.4925**	3.2012	3.4315	3.498	3.5186	2.893	2.6523	2.7518
	Best	**-10.4029**	-10.4029	-10.4029	-10.4029	-10.4029	-10.4029	-10.4029	-10.4024
	Worst	**-2.7609**	-1.8376	-2.7519	-2.7519	-1.8376	-2.7519	-1.8371	-1.8376
F_{23}	Ave	**-10.0003**	-9.4643	-5.9379	-7.1109	-7.2008	-9.1123	-5.2768	-9.9114
	Std	1.6952	2.2603	3.9764	3.7181	3.6154	2.7792	**1.1582**	1.9434
	Best	**-10.5364**	-10.5364	-10.5364	-10.5364	-10.5364	-10.5364	-7.513	-10.5364
	Worst	**-5.1756**	-5.1756	-2.4217	-2.4273	-2.4273	-2.8066	-3.1584	-4.3804

is deployed in terminal device 1. Therefore, the deployment mode of microservices is equal to the deployment mode of corresponding containers.

Considering the instability of the terminal devices, the container where each microservice resides should be backed up on different devices to ensure the robustness of the

TABLE 12. Results of composite benchmark functions.

Func	Type	CISSA	SSA	PSO	GOA	ALO	DA	SCA	WOA
F_{24}	Ave	**30.0053**	110.2327	155.8776	90.8453	207.4952	194.1754	177.7113	279.1412
	Std	48.3032	128.4497	71.736	98.4356	117.9897	122.083	**22.0105**	152.4998
	Best	**1.7424e-04**	1.9503e-03	95.0093	6.1901e-04	5.7448e-03	30.6855	143.6707	37.4585
	Worst	**100.0057**	400.0002	300.0004	300.001	400.037	400.7501	207.9961	433.2938
F_{25}	Ave	136.7543	**115.1595**	195.8424	226.950	159.9945	237.4597	141.9592	236.0729
	Std	81.2607	95.2572	162.658	115.3919	110.7163	141.8029	**24.9221**	66.6575
	Best	**15.148**	15.4722	32.0426	130.7801	19.0267	84.7149	130.7801	119.3591
	Worst	214.5828	**209.8059**	400.3332	425.931	293.2121	403.0255	209.8059	287.0019
F_{26}	Ave	314.6852	333.6369	**294.4965**	417.2032	323.5607	456.8633	513.6812	485.5221
	Std	**41.7187**	82.533	108.8329	150.6854	84.5351	157.066	79.3718	171.0242
	Best	221.2605	206.9833	**157.13**	243.4726	254.8953	310.9156	411.1139	340.898
	Worst	**411.1139**	438.033	411.1139	635.9919	460.4438	706.5522	700.3358	739.0262
F_{27}	Ave	**429.933**	433.5041	452.8456	498.8176	565.1525	520.2384	492.3794	603.2559
	Std	108.1088	139.5072	126.4346	139.6885	133.1088	170.8394	**30.9326**	132.7753
	Best	**299.2959**	309.7755	319.2001	334.0825	426.6405	333.541	461.1001	398.8769
	Worst	695.0759	720.4287	705.7962	727.9242	753.2168	785.0839	**567.53**	753.2168
F_{28}	Ave	**81.1352**	92.4789	242.4538	227.3485	130.5133	243.0856	148.4005	197.8198
	Std	94.3711	137.0837	210.5017	170.4024	105.6583	169.8291	**59.3569**	112.7564
	Best	**4.725**	5.424	22.5445	20.4809	23.9793	59.8906	98.0655	105.0495
	Worst	301.3921	413.6859	539.1734	511.4358	**270.0358**	535.6988	309.3857	472.0563
F_{29}	Ave	**743.5929**	745.2853	824.4995	909.4391	749.5765	905.3214	781.6804	758.4001
	Std	219.9683	220.9136	174.9161	3.5473	216.9906	**1.6295**	196.4364	211.0174
	Best	**500.7066**	500.7424	511.6002	905.2314	505.3516	903.4708	502.0492	526.3388
	Worst	903.4629	907.0808	**903.0404**	912.9092	908.6582	907.8138	939.9401	918.2034

resource pool. At the same time, on account of the limited resources of terminal devices, the maximum number of containers per terminal device is also far less than the number of containers that can be deployed on the server [32].

In the article, we assume that there are N microservice and each one handles a series of similar tasks. The total size of these tasks on each microservice are defined as $\{T_1, T_2, \ldots T_N\}$. Based on the above introduction, we added two backups for every container, therefore, there are 3N containers and M terminal devices for deploying these containers. The redundant container deployment model is described as follows:

1. There are N types of all the tasks, which are processed in N container, everyone is added 2 backups and runs in the corresponding backup container. The set of 3N tasks is $\{T_1, T_2, \ldots T_{3N}\}$, T_i represents the total size of tasks to be processed by the i-th container, and $T_i = T_{N+i} = T_{2N+i}$, where $1 \leq i \leq N$, the total size of each tasks is as shown in Table 18.
2. The set of 3N resources is $\{R_1, R_2, \ldots R_{3N}\}$ and R_i represents the pre-allocated resources of the i-th container, and $R_i = R_{N+i} = R_{2N+i}$, where $1 \leq i \leq N$, the normalized resource of each container is as shown in Table 17.

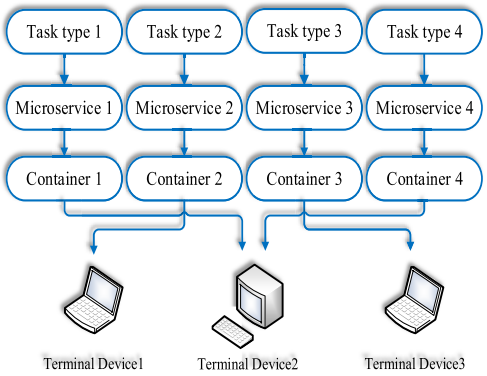

FIGURE 10. Diagram of container deployment relationships in microservice architecture.

3. We use $X(k)$ to represent the device, in which k-th container is deployed, and $1 \leq X(k) \leq M$, $1 \leq k \leq 3N$.
4. The set of M resources is $\{S_1, S_2, \ldots S_M\}$ and Sx represents the maximum amount of resources that the x-th terminal device can provide. To simplify the model, we only consider memory resource to reduce the dimension of resources, the normalized resource of each device is as shown in Table 16.

TABLE 13. p-Values of the Wilcoxon rank-sum test between CISSA and the other algorithms.

Func	SSA	PSO	GOA	ALO	DA	SCA	WOA
F_1	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11
F_2	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11
F_3	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11
F_4	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11
F_5	0.017257	5.3903e-05	3.0199e-11	1.2683e-10	3.0199e-11	3.0199e-11	9.0972e-03
F_6	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	0.042736
F_7	3.0199e-11	6.8265e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	0.027304
F_8	1.8267e-11	2.4613e-11	1.8267e-11	1.8267e-11	1.8267e-11	1.8267e-11	0.027304
F_9	1.7168e-11	1.7168e-11	1.7168e-11	1.7168e-11	1.7168e-11	8.6583e-11	1.1055e-09
F_{10}	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	1.1055e-12
F_{11}	6.3864e-05	6.3864e-05	6.3864e-05	6.3864e-05	6.3864e-05	6.3864e-05	0.016808
F_{12}	1.8267e-07	1.4047e-04	1.8267e-07	1.8267e-07	1.8267e-07	1.8267e-07	0.075662
F_{13}	6.7758e-04	1.8267e-07	3.2984e-07	2.4132e-03	1.8267e-07	1.8267e-07	0.062318
F_{14}	1.1717e-11	1.3206e-05	2.2319e-11	5.8909e-07	1.0921e-04	4.998e-09	1.4918e-06
F_{15}	0.016238	1.4733e-07	3.6709e-03	0.021702	2.4905e-06	0.048413	0.077312
F_{16}	3.0142e-11	4.0806e-12	3.0199e-11	3.0161e-11	6.8593e-06	3.0199e-11	7.2208e-06
F_{17}	3.0199e-11	1.2118e-12	3.0199e-11	4.1997e-10	4.9818e-04	6.0584e-11	4.2911e-10
F_{18}	3.0199e-11	1.2379e-11	3.0199e-11	1.689e-08	5.3686e-10	1.3594e-07	0.012212
F_{19}	3.0142e-11	1.7203e-12	9.049e-05	3.0161e-11	8.883e-04	3.0199e-11	3.0199e-11
F_{20}	2.7086e-08	2.3302e-09	9.8834e-09	1.1228e-08	1.3832e-08	1.2057e-12	1.0035e-09
F_{21}	2.8913e-04	4.1082e-04	6.4142e-03	6.735e-03	1.8349e-06	2.3897e-11	6.765e-08
F_{22}	9.6283e-05	5.8346e-07	3.5573e-07	3.4029e-04	5.4133e-04	3.6328e-04	1.8724e-04
F_{23}	0.017257	0.047218	3.8345e-03	3.0199e-11	5.8282e-03	2.1544e-10	6.387e-12
F_{24}	2.7304e-04	1.1706e-08	4.2736e-04	1.008e-06	2.2022e-06	1.8267e-09	4.3964e-09
F_{25}	5.4133e-04	8.4848e-09	3.0142e-11	7.8759e-07	9.3519e-08	1.698e-08	0.12967
F_{26}	1.6079e-07	7.9365e-04	7.861e-03	3.3342e-11	3.0161e-11	1.9527e-03	7.0881e-08
F_{27}	9.6985e-03	5.7075e-04	1.8588e-04	1.7257e-06	1.4047e-05	2.5748e-06	7.2846e-08
F_{28}	3.8467e-08	4.5864e-10	4.5864e-10	1.4019e-08	3.6105e-10	2.1134e-08	1.133e-08
F_{29}	7.8367e-04	5.6958e-06	7.9365e-04	2.2674e-07	8.3146e-08	3.4783e-03	6.9048e-03

5. A container can be operated at any device in the resource pool as long as the device can provide enough resource, which means $\sum_{k=1}^{3N} R_k^x \leq Sx$ and R_k^x represents the k-th container is deployed in the x-th device.
6. The paper use matrix P[3N,M] to represent the uniqueness of containers, $P_k^x = 1$ when k-th container is deployed in the x-th device, otherwise $P_k^x = 0$.
7. The article uses $F(X(k))$ to represent whether $\sum_{k=1}^{3N} R_k^x \leq Sx$, when the condition is met, $F(X(k)) = 1$, otherwise $F(X(k)) = \text{inf}$.
8. Containers running in the same device work in parallel, and the operating speed of each container will not be influenced by other containers, because the resource of each container is pre-allocated.
9. Different microservices handle tasks at different starting times, we use time set $\{t_1, t_2, \ldots t_{3N}\}$ to represent them, t_i represents the starting time of task on the i-th container, and $t_i = t_{2i} = t_{3i}$, where $1 \leq i \leq N$, the starting time of each container is as shown in Table 19.
10. Given the fixed resource cap for each container, the abilities of each device to handle different type of tasks on each container are different [35], [37]. The matrix of operating speed is set as mips[3N,M], $mips_k^x$ represents the instruction processing speed of x-th device for similar tasks on k-th container. Considering that there is a positive correlation between instruction number and task size [38], the working time for the k-th container running on the x-th device is et_{kx}, which is calculated as follow:

$$et_{kx} = \begin{cases} \dfrac{\alpha T_k}{mips_k^x} + t_k \\ \quad container\ k\ is\ deployed\ on\ device\ x \\ 0 \quad container\ k\ isn't\ deployed\ on\ device\ x \end{cases} \tag{5.1}$$

where α is a constant coefficient, to simplify the model, assuming $\alpha = 1$ in the calculation process.
11. The article set penalty factors to avoid backup containers being deployed on the same device as much as

TABLE 14. Average ranking values using friedman test in different types of test functions.

Algorithm	Unimodal Function	Multimodal Function	Fixed-Multimodal Function	Composite Function	Total Ranking
CISSA	**1.8571**	**1.5**	**2.05**	**1.3333**	**1.7414**
SSA	3.8571	4.3333	3.55	2	3.4655
PSO	2.7143	3.1667	5.1	5	4.1035
GOA	5.5715	5.6667	5	5.5	5.3793
ALO	5.2857	5.1667	4.55	4.6667	4.8793
DA	7.1429	7.6667	4.35	7	6.2586
SCA	6.1429	6.3333	5.95	3.8333	5.6379
WOA	3.7143	2.1667	4.2	6.6667	4.1724

TABLE 15. Results of Page's trend test between CISSA and SSA in F1-F23.

Differences	CISSA-SSA	SSA-CISSA
C_1	104	149
C_2	97	156
C_3	102	151
C_4	117	136
C_5	125	128
C_6	139	114
C_7	122	131
C_8	148	105
C_9	156	97
C_{10}	155	98
L-statistic	7523	6392
p-value	0.0012	0.9990

TABLE 16. Pre-allocated memory about each type of container.

Container type	1	2	3	4	5	6	7	8
R:1-8 (MB)	300	200	300	400	500	600	600	400

possible, the relationship is shown as follows:

$$penalty_i^{x(i)} = \begin{cases} 0 & node\ x\ only\ has\ one\ or\ zero\ container\ of\ i \\ 10 & node\ x\ has\ two\ identical\ containers\ of\ i \\ \text{inf} & node\ x\ has\ all\ three\ identical\ containers\ of\ i \end{cases} \tag{5.2}$$

where $1 \le i \le N$, and the total penalty is calculated as follows:

$$total_{penalty} = 1 + \sum_{i=1}^{N} penalty_i^{x(i)} \tag{5.3}$$

12. The total completion time of all microservices depends on the maximum work time of container, which is shown as:

$$total_{time} = max\ (et_{kx})\ 1 \le k \le 3N, \quad 1 \le X\ (k) \le M \tag{5.4}$$

In consideration of the resource limitation relationship between containers and devices, Eq. (5.4) should be revised as follows:

$$total_{time} = max\ (et_{kx}) * F\ (X\ (k)) * P_k^{x(k)} \tag{5.5}$$

TABLE 17. Memory about each type of terminal device.

Device type	1	2	3	4	5
S:1-5 (GB)	2	2.5	2	1.8	1.5

TABLE 18. Total size of tasks on each container.

Container type	1	2	3	4	5	6	7	8
T:1-5 (GB)	5	4.5	4	3	4	3.8	4.6	2.5

TABLE 19. Starting time of tasks on each container.

Container type	1	2	3	4	5	6	7	8
t:1-8 (s)	100	200	150	0	180	100	20	100

TABLE 20. Mips about each type of container and device.

$Mips_k^x$	k:1-8							
x:1-5	5	3	6	3	2	5	6	7
	4	3	5	3	2	3	5	6
	6	2	7	3	2	6	7	7
	4	4	5	2	1	4	4	6
	5	3	7	2	3	7	4	8

13. Based on the above descriptions and assumptions, the quality of the redundant container deployment model is mostly depended on the total completion time and whether multiple backup containers are deployed on the same device. Hence, the overall evaluation indicator can be expressed as follows:

$$fitness = total_{time} * total_{penalty} \tag{5.6}$$

VI. EXPERIMENT OF CISSA ON REDUNDANT CONTAINER DEPLOYMENT MODEL

In the chapter, we have designed multi-sized redundant container deployment scenarios, which roughly cover the commonly-used size of terminal device cluster [39], [40]. To simplify the description, all containers and terminal device are composed by the basic types described in Table 16-Table 20.

In scenario 1, there are 8 containers which involve container type 1-8 in Table 16, and each container has 2replicas as backup containers, these backup containers are numbered as 9-16 and 17-24, therefore, there are 24 containers in all. Besides, there are 10 terminal devices, each device type in Table 17 has 2 terminal devices, for example, terminal device 7 and terminal device 8 are device type 4.

In scenario 2, there are 96 containers in total, including 32 containers and their replicas, the sequences of container

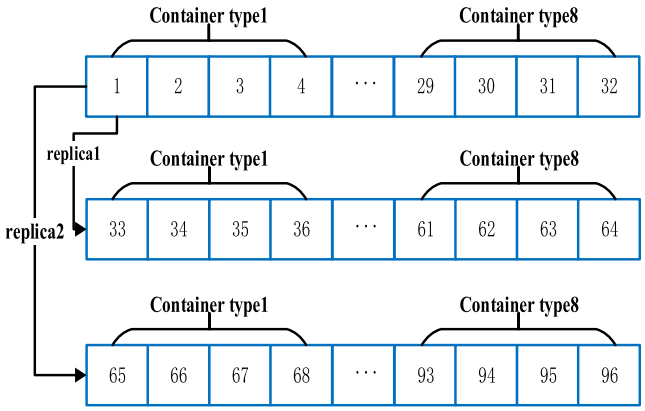

FIGURE 11. Diagram of container number in scenario 2.

number are as shown in Figure 11. And each device type has 8 terminal devices, which means there are 40 terminal devices in all.

In scenario 3, there are 1008 containers in all, including 336 containers and their replicas, each container type has 40 containers, and all these containers have 2 replicas of their own. And there are 420 devices in total with 84 terminal devices per type.

Considering that in real application scenarios, the resource pool composed by terminal devices is more prone to fluctuations compared with commonly-used cloud environments due to the instability of terminal devices, it's necessary to solve the problem by using an algorithm with lower time complexity [32].

In terms of container cluster scheduling systems, two of the most popular tools in industry are Kubernetes and Docker swarm [41]. However, in the terminal resource pool scenario where device resources are limited, deploying Kubernetes may consume a lot of resources [42], for example, the recommended memory reserve configuration should be up to 2G, such large resource consumption is intolerable for terminal devices. Considering the resource usage, we chose Docker swarm for practical application scenarios [43].

The backup strategy of Docker swarm is implemented by data volume mounting [37], but the recovery speed of this method is slow when the terminal device fails. There are no pre-reserved resources for backup containers. When failure occurs, the corresponding data needs to be transferred from the shared volume, and new container will restart. In resource-constrained embedded devices, these processes often take a lot of time. However, when the way of data volume sharing is not considered, redundant containers are deployed by using its own deployment strategies, the results are often unsatisfactory.

Docker swarm contains three default container deployment strategies, which are Binpack, Spread and Random [44]. In the process of experiment, each native functionality of Docker swarm is tested for 100 times, we found that all these default container deployment strategies of Docker swarm were uncapable to handle such complex problem.

TABLE 21. Results of redundant container deployment model by using docker swarm.

	Strategy	Scenario 1	Scenario 2	Scenario3
Average	Binpack	Inf	Inf	Inf
	Spread	Inf	Inf	Inf
	Random	inf	inf	Inf
Std	Binpack	Inf	Inf	Inf
	Spread	Inf	Inf	Inf
	Random	inf	inf	Inf
Best Value	Binpack	inf	Inf	Inf
	Spread	422180	Inf	Inf
	Random	26950	inf	Inf

In these scenarios, containers and their own backups would be deployed in the same device inevitably. After multiple experiments, the mean values and standard deviations on the evaluation function measured by the three basic methods are all inf, the experiment results are shown in Table 21. These results indicate that simply using the native functionality of these common scheduling tools couldn't solve the real problem of these scenarios.

Meanwhile, considering that the terminal devices in the resource pool switch frequently, the corresponding container deployment method should have low time complexity. Compared to a series of algorithms that sacrifice time complexity for higher accuracy, we incline to use CISSA and these comparison algorithms, which can guarantee the accuracy of solving redundant container deployment problem while considering the time complexity.

In each scenario, 7 algorithms are compared with the proposed CISSA, and every algorithm runs 1000 iterations with 100 search agents. In order to reduce the accidental factors, every algorithm is tested for 100 times. In Table 22, we record the statistical data such as average values, standard deviations and best values.

As can be observed from Table 21 that CISSA have indicated an obvious advantage over the other 7 comparative algorithms on solving redundant container deployment problem. For example, in scenario 1, the best value obtained by CISSA is 2180, and the corresponding terminal devices number of containers are (5, 2, 2, 5, 6, 6, 3, 5, 3, 9, 6, 9, 3, 9, 5, 8, 3, 10, 1, 2, 9, 4, 9, 6).

In the model, the penalty factor as a multiplier will greatly affect the fitness of the final solution, therefore, the solutions obtained by these algorithms inevitably deteriorate when the problem dimension increases, that's to say, there are containers and their backups deployed on the same device. Although the tendency of the solutions is to deteriorate as the dimension increases, however, in each scenario, CISSA can still obtain the minimum average value and minimum standard deviation in all dimensions.

TABLE 22. Results of redundant container deployment model by using meta-heuristic algorithms.

	Function	Scenario 1	Scenario 2	Scenario3
Average	**CISSA**	**2.1908e03**	**2.5334e03**	**6.4926e03**
	SSA	2.2497e03	2.7528e03	7.1974e03
	PSO	2.2162e03	2.7231e03	6.6154e03
	GOA	2.2751e03	6.9192e03	1.1551e05
	ALO	2.2551e03	3.9932e03	5.0810e04
	DA	2.3113e03	7.3154e03	6.3664e04
	SCA	2.2951e03	4.2074e05	5.2897e04
	WOA	2.2108e03	2.6859e03	6.9742e03
Std	**CISSA**	**53.1755**	**839.3206**	**1.1879e04**
	SSA	369.4234	2.0752e03	1.2839e04
	PSO	208.5653	2.0813e03	1.1977e04
	GOA	417.2795	4.1535e04	4.7616e05
	ALO	370.3465	1.2302e04	3.0603e05
	DA	561.7930	4.1689e04	5.0554e05
	SCA	458.5627	4.1797e06	3.0567e05
	WOA	205.8914	1.8264e03	1.2729e04
Best Value	**CISSA**	**2180**	**2180**	**2180**
	SSA	2180	2180	2180
	PSO	2180	2180	2180
	GOA	2180	2180	2450
	ALO	2180	2180	2180
	DA	2180	2180	4180
	SCA	2180	2180	2450
	WOA	2180	2180	2180

VII. CONCLUSION AND FUTURE WORKS

The article proposed a comprehensive improved salp swarm algorithm, which was modified in 4 mechanisms on the basis of the original SSA. The proposed CISSA was designed to improve the exploitation capacity and slow convergence rate without increasing the time complexity. In order to more convincingly compare the performance of CISSA and the original SSA, this article used the same benchmark functions used by SSA. The performance of the algorithm was superior to the other 7 comparison algorithms in most benchmark functions, and p-values of Wilcoxon rank-sum statistic tests were also used to prove the irrelevances of the results.

At the same time, this paper designed an evaluation index of redundant container deployment model, which was based on the total completion time and the robustness of the system. To verify the availability of the scheme, we used container clusters of different sizes. When CISSA was applied to redundant container deployment problem of different sizes, it could promote the performance of the original SSA and outperformed all the other comparison algorithms.

The research of CISSA and its application on redundant container deployment problem still remain many issues worthy of further study. Firstly, we should enunciate a theoretical model of convergence to the optimum [45], and the performance of CISSA needs further improvement. Then, given the varying importance of the various tasks running on the containers, we could set up different number of backups for each container. Lastly, in order to meet the increasing industrial demand, the optimized algorithm could be applied in multiple terminal devices located in different network segments, with the additional consideration of communication overhead between devices [46], [47].

REFERENCES

[1] R. Zhao and X. Zhu, "A review of the microservice architecture," *J. Netw. New Media*, vol. 8, no. 1, pp. 58–61 and 65, 2019.

[2] D. Bernstein, "Containers and cloud: From LXC to docker to kubernetes," *IEEE Cloud Comput.*, vol. 1, no. 3, pp. 81–84, Sep. 2014.

[3] R. Morabito, "A performance evaluation of container technologies on Internet of Things devices," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, San Francisco, CA, USA, Apr. 2016, pp. 999–1000.

[4] S. Mirjalili, A. H. Gandomi, S. Z. Mirjalili, S. Saremi, H. Faris, and S. M. Mirjalili, "Salp Swarm Algorithm: A bio-inspired optimizer for engineering design problems," *Adv. Eng. Softw.*, vol. 114, pp. 163–191, Dec. 2017.

[5] Z. Xing and H. Jia, "Multilevel color image segmentation based on GLCM and improved Salp Swarm algorithm," *IEEE Access*, vol. 7, pp. 37672–37690, 2019.

[6] M. Werth, R. Holmes, M. Roggemann, J. Lucas, M. Abercrombie, and D. Thompson, "Improving optical imaging of dim SSA targets with simplified adaptive optics systems," in *Proc. IEEE Aerosp. Conf.*, Big Sky, MT, USA, Mar. 2018, pp. 1–12.

[7] T. K. Mohapatra and B. K. Sahu, " Design and implementation of SSA based fractional order PID controller for automatic generation control of a multi-area, multi-source interconnected power system," in *Proc. Technol. Smart-City Energy Secur. Power (ICSESP)*, Bhubaneswar, India, Mar. 2018, pp. 1–6.

[8] S. Ekinci and B. Hekimoglu, "Parameter optimization of power system stabilizer via Salp Swarm algorithm," in *Proc. 5th Int. Conf. Electr. Electron. Eng. (ICEEE)*, Istanbul, Turkey, May 2018, pp. 143–147.

[9] Q. Zhang, H. Chen, A. A. Heidari, X. Zhao, Y. Xu, P. Wang, Y. Li, and C. Li, "Chaos-induced and mutation-driven schemes boosting Salp chains-inspired optimizers," *IEEE Access*, vol. 7, pp. 31243–31261, 2019.

[10] M. S. Tavazoei and M. Haeri, "Comparison of different one-dimensional maps as chaotic search pattern in chaos optimization algorithms," *Appl. Math. Comput.*, vol. 187, no. 2, pp. 1076–1085, Apr. 2007.

[11] S. Arora and P. Anand, "Chaotic grasshopper optimization algorithm for global optimization," *Neural Comput. Appl.*, pp. 1–21, Jan. 2018.

[12] Y. Zhen and Z. Hongyan, "Research on uniformity based on the chebyshev chaotic map," in *Proc. IEEE Int. Conf. Comput. Intell. Commun. Technol.*, Ghaziabad, India, Feb. 2015, pp. 177–179.

[13] D. He, C. He, L.-G. Jiang, H.-W. Zhu, and G.-R. Hu, "Chaotic characteristics of a one-dimensional iterative map with infinite collapses," *IEEE Trans. Circuits Syst. I. Fundam. Theory Appl.*, vol. 48, no. 7, pp. 900–906, Jul. 2001.

[14] Y. Dai and X. Wang, "Medical image encryption based on a composition of logistic maps and chebyshev maps," in *Proc. IEEE Int. Conf. Inf. Automat.*, Shenyang, China, Jun. 2012, pp. 210–214.

[15] J. Meng, "Realization of periodic orbits of Bernouilli shift map in a digital computer," *J. Zhejiang Univ. Eng. Sci.*, vol. 35, no. 4, pp. 111–114, 2001.

[16] A. G. Radwan and S. K. Abd-El-Hafiz, "Image encryption using generalized tent map," in *Proc. IEEE 20th Int. Conf. Electron., Circuits, Syst. (ICECS)*, Abu Dhabi, United Arab Emirates, Dec. 2013, pp. 653–656.

[17] K. V. Price, N. H. Awad, M. Z. Ali, and P. N. Suganthan, "The 100-digit challenge: Problem definitions and evaluation criteria for the 100-digit challenge special session and competition on single objective numerical optimization," Nanyang Technol. Univ., Singapore, Tech. Rep., 2019.

[18] E. S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Adv. Eng. Softw.*, vol. 95, pp. 51–67, May 2016.

[19] S. Saremi, S. Mirjalili, and A. Lewis, "Grasshopper optimisation algorithm: Theory and application," *Adv. Eng. Softw.*, vol. 105, pp. 30–47, Mar. 2017.

[20] S. Mirjalili, "Dragonfly algorithm: A new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems," *Neural Comput. Appl.*, vol. 27, no. 4, pp. 1053–1073, May 2016.
[21] S. Mirjalili, "SCA: A sine cosine algorithm for solving optimization problems," *Knowl.-Based Syst.*, vol. 96, pp. 120–133, Mar. 2016.
[22] S. Mirjalili, "The ant lion optimizer," *Adv. Eng. Softw.*, vol. 83, pp. 80–98, May 2015.
[23] M. R. AsadiandS and S. M. Kouhsari, " Optimal overcurrent relays coordination using particle-swarm-optimization algorithm," in *Proc. IEEE/PES Power Syst. Conf. Expo.*, Mar. 2009, pp. 1–7.
[24] W. Xiao, H. Deng, Y. Sheng, and L. Hu, "Factored grey wolf optimizer with application to resource-constrained project to scheduling," *Int. J. Innocative Comput. Inf. Control*, vol. 14, no. 3, pp. 881–897, 2018.
[25] Q. Yue and S. Feng, "The statistical Analyses for computational performance of the genetic algorithms," *Chin. J. Comput.*, vol. 32, no. 12, pp. 2389–2392, 2009.
[26] M. Coffin and J. Matthew Saltzman, "Statistical analysis of computational tests of algorithms and heuristics," *Informs J. Comput.*, vol. 12, no. 1, pp. 24–44, Feb. 2000.
[27] F. Wilcoxon, S. K. Katti, and R. A. Wilcox, "Critical values and probability levels for the wilcoxon rank sum test and the wilcoxon signed rank test," *Sel. Tables Math. Statist.*, vol. 1, pp. 171–259, Jun. 1970.
[28] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 3–18, Mar. 2011.
[29] E. B. Page, "Ordered hypotheses for multiple treatments: A significance test for linear ranks," *J. Amer. Stat. Assoc.*, vol. 58, no. 301, pp. 216–230, 1963.
[30] J. Derrac, S. Garcíab, S. Hui, P. N. Suganthan, and F. Herrera, "Analyzing convergence performance of evolutionary algorithms: A statistical approach," *Inf. Sci.*, vol. 289, pp. 41–58, Dec. 2014.
[31] J. Derrac, S. García, S. Hui, F. Herrera, and P. N. Suganthan, "Statistical analysis of convergence performance throughout the evolutionary search: A case study with SaDE-MMTS and Sa-EPSDE-MMTS," in *Proc. IEEE Symp. Differ. Evolution (SDE)*, Singapore, Apr. 2013, pp. 151–156.
[32] B. I. Ismail, E. M. Goortani, M. B. A. Karim, W. M. Tat, S. Setapa, J. Y. Luke, and O. H. Hoe, "Evaluation of docker as edge computing platform," in *Proc. IEEE Conf. Open Syst. (ICOS)*, Bandar Melaka, Malaysia, Aug. 2015, pp. 130–135.
[33] C. Kaewkasi and K. Chuenmuneewong, "Improvement of container scheduling for docker using ant colony optimization," in *Proc. 9th Int. Conf. Knowl. Smart Technol.(KST)*, Chonburi, Thailand, Feb. 2017, pp. 254–259.
[34] M. Abdelbaky, J. Diaz-Montes, M. Parashar, M. Unuvar, and M. Steinder, "Docker containers across multiple clouds and data centers," in *Proc. IEEE/ACM 8th Int. Conf. Utility Cloud Comput. (UCC)*, Limassol, Cyprus, Dec. 2015, pp. 368–371.
[35] M. Sureshkumar and P. Rajesh, "Optimizing the docker container usage based on load scheduling," in *Proc. 2nd Int. Conf. Comput. Commun. Technol. (ICCCT)*, Chennai, India, Feb. 2017, pp. 165–168.
[36] A. Dusia, Y. Yang, and M. Taufer, "Network quality of service in docker containers," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2015, pp. 527–528.
[37] A. Azab, "Enabling docker containers for high-performance and many-task computing," in *Proc. IEEE Int. Conf. Cloud Eng. (IC2E)*, Vancouver, BC, Canada, Apr. 2017, pp. 279–285.
[38] J. Wu and T.-I. Yang, "Dynamic CPU allocation for Docker containerized mixed-criticality real-time systems," in *Proc. IEEE Int. Conf. Appl. Syst. Invention (ICASI)*, Chiba, Japan, Apr. 2018, pp. 279–282.
[39] S. K. Pentyala, "Emergency communication system with Docker containers, OSM and Rsync," in *Proc. Int. Conf. Smart Technol. Smart Nation (SmartTechCon)*, Bengaluru, India, Aug. 2017, pp. 1064–1069.
[40] G. Bhatia, A. Choudhary, and K. Dadheech, "Behavioral analysis of docker Swarm under DoS/ DDoS attack," in *Proc. 2nd Int. Conf. Inventive Commun. Comput. Technol. (ICICCT)*, Coimbatore, India, Apr. 2018, pp. 985–991.
[41] J. Shah and D. Dubaria, "Building modern clouds: Using docker, kubernetes & Google cloud platform," in *Proc. IEEE 9th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Las Vegas, NV, USA, Jan. 2019, pp. 184–189.
[42] A. Modak, S. D. Chaudhary, P. S. Paygude, and S. R. Ldate, "Techniques to secure data on cloud: Docker swarm or kubernetes?" in *Proc. 2nd Int. Conf. Inventive Commun. Comput. Technol. (ICICCT)*, Coimbatore, India, Apr. 2018, pp. 7–12.
[43] M. R. M. Bella, M. Data, and W. Yahya, "Web server load balancing based on memory utilization using docker Swarm," in *Proc. Int. Conf. Sustain. Inf. Eng. Technol. (SIET)*, Malang, Indonesia, Nov. 2018, pp. 220–223.
[44] N. Naik, "Building a virtual system of systems using docker swarm in multiple clouds," in *Proc. IEEE Int. Symp. Syst. Eng. (ISSE)*, Edinburgh, U.K., Oct. 2016, pp. 1–3.
[45] G. Xu and G. Yu, "On convergence analysis of particle swarm optimization algorithm," *J. Comput. Appl. Math.*, vol. 333, pp. 65–73, May 2018.
[46] B. Xie, G. Sun, and G. Ma, "Docker based overlay network performance evaluation in large scale streaming system," in *Proc. IEEE Adv. Inf. Manage., Commun., Electron. Autom. Control Conf. (IMCEC)*, Xi'an, China, Oct. 2016, pp. 366–369.
[47] Y. Li and Y. Xia, "Auto-scaling Web applications in hybrid cloud based on docker," in *Proc. 5th Int. Conf. Comput. Sci. Netw. Technol. (ICCSNT)*, Changchun, China, Dec. 2016, pp. 75–79.

BOTAO MA received the B.S. degree from the University of Electronic Science and Technology of China, in 2014. He is currently pursuing the Ph.D. degree with the University of Chinese Academy of Sciences. His current research interests include distributed computing, evolutionary algorithms, and virtualization technology.

HONG NI received the M.S. degree from the Institute of Acoustics, Chinese Academy of Sciences, in 1989, where he is currently a Researcher, a Doctoral Tutor, and the Deputy Director. His research interests include broadband multimedia communication, network new media technology and application, embedded systems, and middleware technology.

XIAOYONG ZHU received the Ph.D. degree in signal and information processing from the University of Chinese Academy of Sciences, in 2009. He is currently a Professor with the Chinese Academy of Sciences. His current research interests include embedded systems, virtualization technology, and new network communication.

RAN ZHAO received the B.S. degree in automation from Tsinghua University, in 2014, and the Ph.D. degree in signal and information processing from the University of Chinese Academy of Sciences, in 2019. His current research interests include evolutionary algorithms and cloud computing architecture.