International Journal of Innovative
Computing, Information and Control
Volume **16**, Number **1**, February **2020**

ICIC International ©2020 ISSN 1349-4198
pp. **283–299**

A TRANSFORMED SALP SWARM ALGORITHM ON CONTAINER DEPLOYMENT PROBLEM

Botao Ma[1,2], Hong Ni[1,2], Xiaoyong Zhu[1,*] and Zhao Wang[1,2]

[1]National Network New Media Engineering Research Center
Institute of Acoustics, Chinese Academy of Sciences
No. 21, North 4th Ring Road, Haidian District, Beijing 100190, P. R. China
{ mabt; nih }@dsp.ac.cn; *Corresponding author: zhuxy@dsp.ac.cn

[2]School of Electronic, Electrical and Communication Engineering
University of Chinese Academy of Sciences
No. 19(A), Yuquan Road, Shijingshan District, Beijing 100049, P. R. China

Received April 2019; revised August 2019

Abstract. *Salp Swarm Algorithm (SSA) is a meta-heuristic algorithm for solving single-objective optimization problems with satisfying performance in the exploration phase. However, the accuracy in the exploitation phase still remains a problem to be solved. The paper presented a Transformed SSA (TSSA). Firstly, the tent chaotic sequence was used to guarantee the uniformity of solution distribution. Secondly, the exploration mode was changed to improve the accuracy of solutions. Lastly, an improved exploitation mode was introduced to supplant the original way. The TSSA was compared with PSO algorithm and multiple newly-proposed heuristic algorithms in 29 widely-accepted test functions. We also applied the proposed TSSA to container deployment problem in a microservice architecture; in this scenario, each microservice ran in an isolated docker container. The results of these experiments demonstrated that TSSA performed better than other algorithms in most test functions and the container deployment problem.*

Keywords: Transformed salp swarm algorithm, Tent chaotic distribution, Nonlinear convergence factor, Microservice architecture, Container deployment

1. **Introduction.** In microservice architecture, an overall service is split into multiple microservices, which are deployed separately in virtual machines or devices [1]. These microservices have independent processes, and communicate with each other through lightweight mechanisms. To better utilize idle resources of terminal devices in edge computing, terminal devices with idle resources in the neighborhood are capable of integrating into a cluster to handle such microservices.

As a representative of lightweight virtualization, docker container can virtualize multiple isolated environments on a single device and costs less system resources [2]. Therefore, the usual practice in industry is to run each microservice in a separated docker container. When some of these microservices are upgraded or down, the other services will not be affected.

Recently, many studies on the container deployment problem have been developed on the basis of different evaluation indictors [3,14,15]. According to the above background, this article proposed a scheme to deploy multiple microservices in several terminal devices, which was based on the number of calls between different microservices and the communication overhead between different devices. On the premise that total resources of containers deployed in one device should be less than the total resources of this device,

DOI: 10.24507/ijicic.16.01.283

the quality of microservice deployment scheme was evaluated by total communication overhead between all microservices.

Salp Swarm Algorithm (SSA) is a population-based heuristic optimization algorithm proposed by Mirjalili et al. in 2017 [4], which is inspired by the nature behavior of salp swarm. It is easy to implement because of the only main controlling parameter. The excellent exploration ability makes the SSA be capable of solving optimization problems in complex situations.

With its good performance, SSA has been widely used in several areas [16,18]. However, the algorithm still has some limitations. The excellent exploration ability may lead to inaccuracy of solutions. To overcome the disadvantage and improve the accuracy of SSA, a Transformed Salp Swarm Algorithm (TSSA) was proposed in this article; at the same time, the proposed algorithm would be used to solve container deployment problem in a microservice architecture. The main contributions of this work are described as follows:

- Initializing the population of search agents by tent chaotic map and using revised main controlling parameter c_1 to reconstruct two-stage exploration mechanism;
- A position updating method based on spiral line was applied to replace the original weighting scheme;
- A container deployment model was proposed, which was evaluated by the overall service communication overhead.

The rest of this paper is organized as follows. The original SSA is briefly introduced in Section 2. Section 3 presents the Transformed Salp Swarm Algorithm (TSSA). Section 4 displays experimental results and performance of all test algorithms. Section 5 applies the proposed TSSA to solving container deployment problem; the corresponding experimental parameters and results are also shown in this section. In the end, conclusions and future expectations are shown in Section 6.

2. Salp Swarm Algorithm (SSA).

The mathematical model simulates the predatory behavior of salps, which belong to the family of Salpidae and have transparent barrel-shaped body. SSA divides the population of salps chain into two groups: leader and followers. The leader is the salp at the front of the chain which guides swarm and the other salps [4].

The motion trail of salp swarm is shown in Figure 1. Positions of these salps are defined in an n-dimensional search space and n is the number of dimensions of a given problem, so the positions are stored in a two-dimensional matrix called x. There is a food source called F in the search space as the target of swarm.

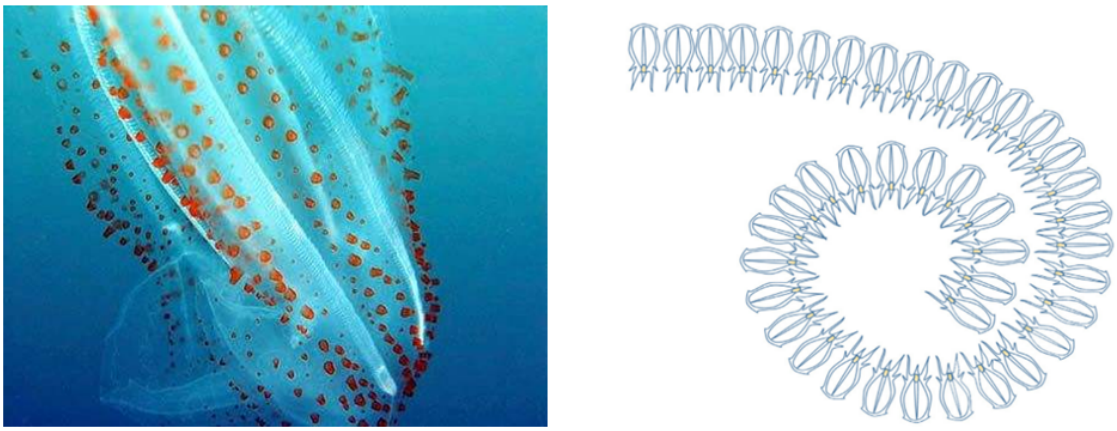

FIGURE 1. Individual salp and motion trail of salp swarm

The location update formula is proposed as Equation (1):

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

where X_j^1 is the position of the first salp in the j-th dimension, F_j is the position of food source in the j-th dimension, ub_j shows the upper bound of j-th dimension, lb_j indicates 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.

Equation (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 as follows:

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

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 which dictates if the next position in j-th dimension should be towards positive or negative infinity as well as the step size.

To simulate the trajectory of the swarm, basic SSA updates the position of the swarm as Equation (3):

$$X_j^i = \frac{1}{2} a t^2 + v_0 t \quad i \geq 2 \tag{3}$$

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

$$X_j^i = \frac{1}{2} \left(X_j^i + X_j^{i-1} \right) \quad i \geq 2 \tag{4}$$

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

3. Transformed Salp Swarm Algorithm.

SSA has excellent exploratory ability in search space, and it is easy to implement. However, excessive pursuit on exploratory ability could cause some disadvantages which prevent the algorithm from getting solutions with high accuracy. In some unimodal test functions, SSA is uncapable to find the optimal solutions. Meanwhile, the positions of follower salps are based on accelerated movement around food sources, considering in some respects, the accelerated movement in mathematical model is not suitable for the actual wandering way of salp swarm. To handle these deficiencies, four improvements are proposed to optimize the original model. The details of these improvements are explicitly in this part. At the end of this chapter, Algorithm 1 shows the pseudo-code of TSSA.

3.1. Initial distribution of tent chaotic map.

The meta-heuristic algorithm is sensitive to the initial distribution of search agents, and traditional meta-heuristic algorithms usually initialize population by random distribution. Different random maps would lead the population to different spatial distribution, which could directly affect the performance of the algorithm. Due to the homogeneity and pseudo-randomness of chaotic motion, initialization of population by chaotic map may achieve better ergodicity [5].

Comparing with several chaotic mapping initial distribution, such as Chebyshev map, logistic map, and tent map, we chose tent chaotic map to initialize the population because of its well-performed uniformity and randomness [5]. The normalized formula for tent map equation is given in Equation (5):

$$X_{k+1} = \begin{cases} uX_k & X_k < 0.5 \\ u\left(1 - X_k\right) & X_k \geq 0.5 \end{cases} \tag{5}$$

where u is set to 2 to make sure the range is $(0, 1)$, X_k is the value of i-th search agent.

In order to satisfy the initial distribution of each test function with different domain of definition, the normalized equation should be revised in Equation (6):

$$X_j^i = \begin{cases} 2\left(X_j^{i-1} - lb_j\right) + lb_j & 0 \leq \dfrac{X_j^{i-1} - lb_j}{ub_j - lb_j} < 0.5 \\ 2\left(ub_j - X_j^{i-1}\right) + lb_j & 0.5 \leq \dfrac{X_j^{i-1} - lb_j}{ub_j - lb_j} \leq 1 \end{cases} \tag{6}$$

where $i \geq 2$, X_j^i shows the position of i-th follower salp in j-th dimension, ub_j shows the upper bound of j-th dimension, lb_j indicates the lower bound of j-th dimension.

3.2. **Revised exploration mechanism.** The original SSA sets threshold to $0.5N$, which means the former $0.5N$ search agents will work for global search, and the other $0.5N$ search agents will follow the former position to optimize the solution, where N is number of search agents. According to a large number of experiments, we found threshold value $0.5N$ was not the best choice for most benchmark functions.

In the paper, we used two-stage exploration mechanism. In the first stage, the exploration mechanism of the original algorithm was maintained in order to ensure the powerful exploring ability of the original algorithm, based on a large number of test comparisons on different benchmark functions, we set the first threshold to $\lfloor 0.2N \rfloor$. 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 $\lfloor 2N/3 \rfloor$, and the original exploration step length could be set to a more refined value in Equation (7).

$$X_j^{i+1} = \begin{cases} X_j^1 + c_1\left(c_2 X_j^1 - X_j^i/2\right) & 0 \leq c_3 < 0.5 \\ X_j^1 - c_1\left(c_2 X_j^1 - X_j^i/2\right) & 0.5 \leq c_3 \leq 1 \end{cases} \tag{7}$$

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 the first 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 Equation (7) are able to find optimum solutions with higher accuracy.

3.3. **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 2 to 0 in the new domain of definition, c_1 is revised in Equation (8):

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

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 during the search iterations. To remedy this defect, damping motion is added in the convergence coefficient c_1 to increase perturbation [19]. After multiple tests on different test functions, the weight ratio of damping motion was set to 0.3.

3.4. **Spiral updating position.** Original algorithm updates the positions of follower salps by uniformly acceleration relation. However, we can see that the shape of the salp chain in Figure 1 is more similar to spiral line. To make the model more physical, Equation

(9) would be used to replace Equation (4) in our mathematical model when serial numbers of follower salps are between $\lfloor 2N/3 \rfloor + 1$ to N. These follower salps spin around the leader by using a shrinking circle or a spiral shaped path [7], and Equation (8) is calculated as follows:

$$X_j^i = \left| X_j^1 - X_j^i \right| e^l \cos(2\pi l) + F_j \tag{9}$$

where $\left| X_j^1 - X_j^i \right|$ indicates the distance of i-th follower salp to the 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 food source, the magnitude of the spiral line was determined by the distance between the follower salp and the leader salp.

Algorithm 1. Pseudo-code of Transformed Salp Swarm Algorithm

(1) Initialize the N salp swarm population X_i $(i = 1, 2, \ldots, N)$ with tent chaotic map by Equation (6)
(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 Equation (8)
(7) **for1** $0 < i < 0.2N$
(8) Update the position with longer step length in exploration phase by Equation (1)
(9) 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 Equation (7)
(13) Calculate the fitness of current search agent
(14) **end for2**
(15) **for3** $2N/3 < i < N$
(16) Update the position with spiral motion formulain exploitation phase by Equation (9)
(17) Calculate the fitness of current search agent
(18) **end for3**
(19) Choose the best fitness of all search agents, replace target fitness if it is a better solution
(20) Record position of target
(21) **end for**
(22) **end while**
(23) Check if any search agent goes beyond the search space and amend it
(24) current iteration + 1
(25) Return target fitness and target position

4. **Results on Benchmark Experiment.** In this work, a series of experiments was conducted to evaluate the performance of the proposed TSSA. The algorithm was benchmarked on 29 test functions with 7 meta-heuristic algorithms, including the original Salp Swarm Algorithm (SSA) [4], the Grasshopper Optimization Algorithm (GOA) [8], Whale

Optimization Algorithm (WOA) [7], Dragonfly Algorithm (DA) [9], Multi-Verse Optimization (MVO) [10], Ant Lion Optimizer (ALO) [11] and a classical heuristic algorithm, Particle Swarm Optimization (PSO) [12]. Each comparison algorithm performed well on several test functions; the parameters of these algorithms are set as the papers [7-12] describe.

These 29 well-known benchmark functions are divided into 4 types, which are used to test search ability of the proposed algorithm. These functions are listed in Table 1, Table 3, Table 4 and Table 7, where dim indicates dimension of each function. Search space of each function is limited by range parameters, 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.

4.1. **Evaluation of exploitation capability.** The first type of benchmark functions has only one global optimum. In Table 1, the unimodal functions F1-F7 can test the exploitation capability 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.

Table 1. 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
$F_3(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$	30	$[-100, 100]$	0
$F_4(x) = \max_i \{ \lvert x_i \rvert,\ 1 \le i \le n \}$	30	$[-100, 100]$	0
$F_5(x) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \right]$	30	$[-30, 30]$	0
$F_6(x) = \sum_{i=1}^{n} ([x_i + 0.5])^2$	30	$[-100, 100]$	0
$F_7(x) = \sum_{i=1}^{n} i x_i^4 + random[0, 1)$	30	$[-1.28, 1.28]$	0

Comparison data are shown in Table 2. In the table, TSSA works best in 4 of the 7 unimodal benchmark tests, and reaches second place twice. As for standard deviation and worst value, TSSA also performs better than the other algorithms in most cases, which means the proposed algorithm is able to produce an optimal solution with stability.

4.2. **Evaluation of exploration capability.** The second and third types of benchmark functions own several local optimums. In Table 3 and Table 4, these multimodal functions F8-F23 are designed for evaluating the exploration capability of the algorithms. The search may fall into local optimums when dealing with multimodal functions if the algorithm performs poorly in exploration phase. Even if the algorithm owns strong exploitation ability, lacking of the exploration ability would make all efforts in vain.

According to the results in Table 5 and Table 6, TSSA outperforms all the other comparison algorithms in terms of average values in 10 of 16 benchmark tests and achieves 11 best values. At the same time, TSSA behaves better than the original algorithm in

TABLE 2. Results of unimodal benchmark functions

Func	Type	TSSA	SSA	PSO	GOA	ALO	DA	MVO	WOA
F_1	Ave	6.199e-15	3.7766e-07	6.1248e-06	34.2098	1.2848e-03	1915.2408	1.2487	**1.4466e-69**
	Std	1.1871e-14	9.1872e-07	6.8643e-06	22.2996	1.4178e-03	889.5906	0.42782	**7.9236e-69**
	Best	3.1031e-17	2.3299e-08	1.2524e-07	5.166	2.5476e-04	578.7575	0.64391	**1.0679e-86**
	Worst	5.7986e-14	4.9376e-06	1.7687e-04	99.4415	3.5214e-03	3766.9521	2.4783	**4.3399e-68**
F_2	Ave	6.6289e-10	2.4871	0.045204	19.7742	41.0818	14.7403	7.9824	**5.6001e-51**
	Std	5.4101e-10	1.5315	0.16441	21.7689	45.2608	4.8384	27.3407	**1.636e-50**
	Best	1.3969e-10	0.20054	4.1337e-04	3.043	2.4559	3.4118	0.54848	**6.8292e-58**
	Worst	2.3361e-09	6.6585	0.90182	90.9762	131.566	21.1672	114.4957	**6.8296e-50**
F_3	Ave	**0.42514**	1516.9863	211.3245	3275.2429	4815.8221	13334.8431	232.8326	45034.4731
	Std	**0.54515**	908.092	111.0891	2022.8163	1671.1507	8585.9586	83.488	16289.2284
	Best	**3.9417e-08**	274.3046	50.9278	883.9015	1777.2923	2254.7522	91.5906	19915.204
	Worst	**2.9679**	3926.7125	490.471	10083.0382	7418.3747	15298.9712	437.4359	84881.83
F_4	Ave	**1.325**	12.45	3.9256	14.444	16.9612	31.7089	2.1666	53.3474
	Std	**1.0542**	3.0442	1.2755	4.1961	5.1606	8.5852	0.6177	27.789
	Best	**6.2319e-04**	6.1313	1.3752	7.394	8.4173	12.6756	1.172231758	9.0348e-03
	Worst	**4.4785**	17.981	5.949	28.2597	30.283	49.0818	3.9959	88.9081
F_5	Ave	**27.9971**	213.0121	56.4356	3143.3388	326.6604	412824.1884	493.895	28.0338
	Std	**0.49358**	447.4194	82.3257	7634.4715	415.049	772217.7579	749.3712	0.81284
	Best	26.2224	27.3993	**10.7441**	233.8927	28.2927	3678.1152	34.7908	27.0564
	Worst	**28.8422**	1906.033	464.9722	32122.3686	1481.0289	3118631.5425	3033.4649	28.7801
F_6	Ave	0.17165	**1.5955e-07**	3.1769e-03	36.6188	1.3783e-03	1997.718	1.2366	0.38121
	Std	0.050128	**1.9136e-07**	0.016625	20.3804	1.1352e-03	1183.0783	0.31474	0.24548
	Best	0.075675	**2.2864e-08**	5.6006e-07	10.85	9.9949e-05	702.5253	0.64421	0.082401
	Worst	0.26343	**1.0345e-06**	0.091134	98.0702	4.9721e-03	6157.6156	1.7719	1.1348
F_7	Ave	**2.7643e-03**	0.18311	0.034269	0.040015	0.28724	0.60676	0.03441	6.1448e-03
	Std	**2.6899e-03**	0.067974	0.017026	0.018028	0.099622	0.56741	0.01108	5.253e-03
	Best	**2.7557e-05**	0.04359	0.015513	8.1886e-03	0.13218	0.17388	0.010949	8.4567e-04
	Worst	**0.011791**	0.3812	0.092794	0.082053	0.58462	3.3547	0.050907	0.011791

TABLE 3. Multimodal benchmark functions

Function	Dim	Range	$f_{\min}$
$F_8(x) = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	30	$[-500, 500]$	418.9829*Dim
$F_9(x) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	30	$[-5.12, 5.12]$	0
$F_{10}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$	30	$[-32, 32]$	0
$F_{11}(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	30	$[-600, 600]$	0
$F_{12}(x) = \frac{\pi}{n}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\right\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$ $y_i = 1 + \frac{x_i + 1}{4}$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a < x_i < a \\ k(-x_i - a)^m & x_i < -a \end{cases}$	30	$[-50, 50]$	0
$F_{13}(x) = 0.1\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{n}(x_i - 1)^2\left[1 + \sin^2(3\pi x_i + 1)\right] + (x_n - 1)^2\left[1 + \sin^2(2\pi x_n)\right]\right\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	30	$[-50, 50]$	0

TABLE 4. Fixed-dimension multimodal benchmark functions

Function	Dim	Range	$f_{\min}$
$F_{14}(x) = \left(\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}\right)^{-1}$	2	$[-65, 65]$	0
$F_{15}(x) = \sum_{i=1}^{11} \left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2$	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
$F_{17}(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$	2	$[-5, 5]$	0.398
$F_{18}(x) = \left[1 + (x_1 + x_2 + 1)^2\left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right] \times \left[30 + (2x_1 - 3x_2)^2 \times \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right]$	2	$[-2, 2]$	3
$F_{19}(x) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2\right)$	3	$[1, 3]$	-3.86
$F_{20}(x) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\right)$	6	$[0, 1]$	-3.32
$F_{21}(x) = -\sum_{i=1}^{5} \left[(X - a_i)(X - a_i)^T + c_i\right]^{-1}$	4	$[0, 10]$	-10.1532
$F_{22}(x) = -\sum_{i=1}^{7} \left[(X - a_i)(X - a_i)^T + c_i\right]^{-1}$	4	$[0, 10]$	-10.4028
$F_{23}(x) = -\sum_{i=1}^{10} \left[(X - a_i)(X - a_i)^T + c_i\right]^{-1}$	4	$[0, 10]$	-10.5363

TABLE 5. Results of multimodal benchmark functions

Func	Type	TSSA	SSA	PSO	GOA	ALO	DA	MVO	WOA
F_8	Ave	−7561.5331	−7076.6538	−6438.2845	−7673.025	−5456.8822	−5488.7838	−7768.2216	**−10527.8401**
	Std	695.6631	756.7834	755.4979	759.0983	**64.9291**	573.8238	745.6872	1860.2069
	Best	−9310.6934	−8719.8079	−7646.6541	−8721.8403	−5644.4458	−6734.7262	−9817.0442	**−12569.0502**
	Worst	−6211.9193	−5263.4383	−5079.3903	−6024.9831	−5417.6748	−4465.005	−5921.351	**−7560.1486**
F_9	Ave	6.0112	55.7508	46.2655	96.3403	79.498	172.9918	120.224	**3.7896e-15**
	Std	8.6965	17.435	14.2677	30.2648	26.8635	33.9423	24.9474	**2.0756e-14**
	Best	**0**	15.9193	26.8639	32.6543	43.7803	96.6606	72.3776	0
	Worst	39.8458	97.5057	76.6116	145.3632	134.32	234.227	160.9621	**1.4211e-14**
F_{10}	Ave	**2.552e-14**	2.6177	1.4885	5.2136	4.5583	10.459	2.4771	4.4409e-14
	Std	**1.0757e-14**	1.2605	1.0406	1.2706	3.0144	1.8064	3.3268	2.4685e-14
	Best	7.9936e-15	0.1206	6.5335e-04	2.9315	1.5018	6.5435	0.64865	**8.8818e-16**
	Worst	**6.839e-14**	2.3168	3.367e-11	2.8144	3.0271	6.8494	2.0227	8.5365e-14
F_{11}	Ave	**0.010631**	0.016693	0.026539	1.1268	0.06859	21.0259	0.84863	0.012065
	Std	**0.016976**	0.13643	0.026612	0.082979	0.041115	8.8756	0.084787	0.1913
	Best	**0**	6.8799e-04	1.7634e-06	0.97705	0.011877	8.0817	0.5968	0
	Worst	**0.064986**	0.66938	0.091082	1.3311	0.18437	40.6642	1.0121	0.6155
F_{12}	Ave	**0.019718**	6.3063	0.44613	8.6445	13.3065	7114.8527	2.0444	0.025995
	Std	**0.011643**	3.368	0.26435	4.9626	4.5415	133642.169	1.1651	0.033859
	Best	2.0753e-03	0.71861	**3.6431e-07**	3.3265	4.6545	14.7392	0.032992	5.56629e-03
	Worst	**0.069062**	16.7087	0.93606	28.5395	22.4794	624140.540	4.2569	0.15246
F_{13}	Ave	1.814	18.7259	**0.10213**	37.5607	28.7336	388686.5496	0.16458	0.55203
	Std	0.52572	14.2972	**0.40083**	21.173	18.6913	848008.8765	0.1165	0.35429
	Best	0.11599	1.8269e-03	**2.063e-07**	4.1725	0.49933	28.5256	0.061736	0.041001
	Worst	2.5557	41.7829	2.1804	80.1317	66.3801	3633323.450	**0.63024**	1.6382

TABLE 6. Results of fixed-dimension multimodal benchmark functions

Func	Type	TSSA	SSA	PSO	GOA	ALO	DA	MVO	WOA
F_{14}	Ave	1.5264	1.0974	3.1028	**0.998**	3.2579	1.1303	0.998	3.3519
	Std	1.0617	0.30331	2.3768	**4.1233e-16**	3.2256	0.50338	4.4741e-11	3.578
	Best	**0.998**	0.998	0.998	0.998	0.998	0.998	0.998	0.998
	Worst	5.9288	1.992	10.7632	0.998	11.7187	2.9821	**0.998**	10.7632
F_{15}	Ave	2.44762e-03	2.802e-03	**5.4427e-04**	8.5259e-03	3.6075e-03	6.6138e-03	5.3579e-03	8.8915e-04
	Std	**4.4421e-04**	8.8915e-04	5.4201e-03	8.0946e-04	9.0811e-03	6.7373e-03	0.012812	8.4231e-03
	Best	**3.0749e-04**	3.7476e-04	3.075e-04	3.0823e-04	5.5747e-04	5.57563e-04	4.0755e-04	3.4297e-04
	Worst	0.02046	0.047048	0.020363	0.020375	0.020721	0.063374.07	0.020364	**2.2519e-03**
F_{16}	Ave	**−1.0316**	−1.0316	−1.0316	−1.0316	−1.0316	−1.0316	−1.0316	−1.0316
	Std	2.6981e-10	3.1652e-14	**6.7752e-16**	2.7496e-13	4.293e-14	3.8812e-07	3.2115e-07	1.7242e-09
	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	9.318e-08	9.2461e-14	**0**	9.7362e-13	1.1094e-13	9.4572e-07	5.0891e-07	2.1327e-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	0.39789	0.39789
F_{18}	Ave	**3**	3	3.9	8.4	3	3	3	3.0001
	Std	1.1271e-06	**2.2739e-13**	4.9295	20.5504	3.4727e-13	3.4039e-06	3.4032e-06	1.3957e-04
	Best	**3**	3	3	3	3	3	3	3
	Worst	**3**	3	3	84	3	3	3	3.0006
F_{19}	Ave	**−3.8628**	−3.8628	−3.8628	−3.7148	−3.8628	−3.8627	−3.8628	−3.855
	Std	7.13e-06	7.4765e-12	2.6823e-15	0.29919	1.2004e-12	2.4251e-04	1.3044e-06	9.6245e-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	−3.8628	−3.8611
F_{20}	Ave	−3.2647	−3.24	−3.2721	**−3.2805**	−3.2702	−3.2588	−3.253	−3.2589
	Std	0.067881	0.064265	0.06318	**0.059748**	0.060259	0.079759	0.079759	0.095563
	Best	**−3.322**	−3.322	−3.322	−3.322	−3.322	−3.322	−3.322	−3.3216
	Worst	−3.1376	−3.1519	−3.1376	−3.1859	**−3.2003**	−3.0769	−3.1901	−3.0401
F_{21}	Ave	**−8.4699**	−7.8876	−5.3951	−5.1395	−6.8718	−6.8648	−7.2882	−7.8627
	Std	**2.6748**	3.5369	3.1741	3.0711	3.2345	2.8218	3.0155	2.9149
	Best	**−10.1532**	−10.1532	−10.1532	−10.1532	−10.1532	−10.1532	−10.153	−10.1528
	Worst	**−2.6305**	−2.6305	−2.6305	−2.6305	−2.6305	−2.6304	−2.6304	−2.6292
F_{22}	Ave	**−9.1711**	−7.3413	−6.0843	−5.9811	−6.6017	−7.0961	−8.6784	−7.7381
	Std	**2.4644**	3.1716	3.4042	3.505	3.5337	2.918	3.526	2.9626
	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	−1.8376	−1.8371
F_{23}	Ave	**−8.8276**	−7.7676	−6.0114	−4.437	−6.6708	−7.1719	−8.7747	−7.1365
	Std	2.9413	3.5269	3.8243	3.4299	3.4562	3.0701	3.0392	3.5446
	Best	**−10.5364**	−10.5364	−10.5364	−10.5364	−10.5364	−10.5364	−10.5363	−10.5359
	Worst	**−2.8711**	−1.6766	−2.4217	−1.8595	−1.6766	−2.8066	−2.4273	−1.6741

14 sets of data, which means the proposed TSSA behaves better in exploration phase compared with the original SSA.

4.3. **Evaluation of avoiding local optimums.** Composite functions are the combination of several some basic benchmark functions with same domain of definition. These benchmark functions own quite a number of local optimums and they can test the exploitation abilities and exploration abilities at the same time. Only when the test algorithms strike the balance between these two abilities can they find solutions closer to the global optimal solutions.

The results in Table 8 show that compared with other 7 meta-heuristic algorithms, TSSA still outperforms other algorithms on F24, F27, F28, which means the proposed algorithm provides very competitive performance in the composite functions.

4.4. **Evaluation of averaged convergence curves.** The averaged convergence curves in this chapter indicate the tendency and rate of convergences, and these curves display the optimum of each algorithm at the same time. Some of the averaged convergence curves of algorithms on unimodal functions are shown in Figure 2. As these curves indicate, TSSA is capable of finding the optimal solution with competitive convergence rate as well. Figure 3 show the curves of multimodal benchmark functions. In these cases, it is

TABLE 7. Composite benchmark functions

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

clear that the compared algorithms with faster convergence speed usually cannot reach the optimal solutions; on the contrary, TSSA sacrifices partial convergence rate in exchange for better searching ability.

As for the curves of composite benchmark functions, Figure 4 shows that the convergence rate of TSSA is satisfactory; moreover, these curves also reveal the well-behaved overall optimization capacity of the proposed algorithm.

4.5. **Significance of the results.** The comparison based on average value could not express difference between results. In order to judge whether the results of TSSA are significantly different from results of other algorithms, the Wilcoxon rank-sum test is carried out, which is a nonparametric test of null hypothesis [18]. The result calculated in this way is marked as p-value. In this work, p-values were calculated based on statistic data between TSSA and each of the other algorithms. By definition, when p-value is less than 0.05, it could be considered that difference between two samples is significant. The

TABLE 8. Results of composite benchmark functions

Func	Type	TSSA	SSA	PSO	GOA	ALO	DA	MVO	WOA
F_{24}	Ave	**50.0041**	80.0073	136.0351	91.0334	123.5287	206.7901	70.1104	160.8798
	Std	**70.7092**	113.5245	95.76	137.7856	111.6866	93.8733	82.3061	106.7659
	Best	1.2814e-03	**5.1258e-05**	1.6245e-03	3.7812e-04	3.9533e-04	21.7657	2.3956e-03	50.2348
	Worst	**300.0003**	400.0048	300.0046	500.0027	380.8259	403.1323	500.0027	430.115
F_{25}	Ave	136.7543	**113.7946**	197.4287	224.6313	161.0097	237.4597	182.9257	237.0832
	Std	84.2916	96.2217	162.4752	115.5219	112.0313	141.1126	160.2984	**66.3385**
	Best	**15.148**	15.4722	32.0426	130.7801	19.0267	84.7149	15.042	119.3591
	Worst	214.5828	**209.8059**	400.3332	425.931	293.2121	403.0255	417.1762	287.0019
F_{26}	Ave	328.8339	353.7246	388.244	390.591	298.4989	475.4832	**238.5968**	527.5308
	Std	109.4225	117.0363	109.6057	163.9833	148.3748	142.5933	**61.7703**	129.4443
	Best	166.8463	206.8256	205.6139	182.9626	188.8034	285.1006	**161.2621**	313.6263
	Worst	532.3176	535.8106	646.4158	774.1267	553.5444	700.3358	**371.6087**	721.8944
F_{27}	Ave	**397.8475**	469.8181	535.279	496.8436	496.365	537.1297	446.8109	705.9838
	Std	**90.6146**	130.7774	151.9004	139.1317	130.1846	124.7955	136.2914	98.598
	Best	**285.3821**	304.9143	306.5712	314.3576	329.3154	336.554	289.9556	389.9549
	Worst	**720.2044**	900.0023	819.3079	816.8678	771.6583	831.8431	730.0914	836.8155
F_{28}	Ave	**94.497**	136.8129	176.0039	188.8622	198.7807	168.6928	112.1789	256.2377
	Std	**137.6279**	201.3604	193.5497	175.543	123.3339	223.2135	150.6825	168.1784
	Best	**2.7818**	9.0706	5.5174	11.2834	73.9141	40.6371	4.4713	89.7476
	Worst	**506.2416**	524.7751	538.5924	508.4277	535.852	821.3798	504.9618	548.6243
F_{29}	Ave	816.4387	818.0079	824.4995	**799.1566**	881.7421	850.825	836.1283	860.3455
	Std	219.9683	220.9136	174.9161	158.1368	**105.3799**	131.5775	139.7219	149.0895
	Best	501.5774	501.5196	511.6002	501.5294	**500.9302**	532.7938	502.0492	526.3388
	Worst	906.1241	907.4463	**903.7813**	909.3123	914.5163	915.7554	904.202	939.9401

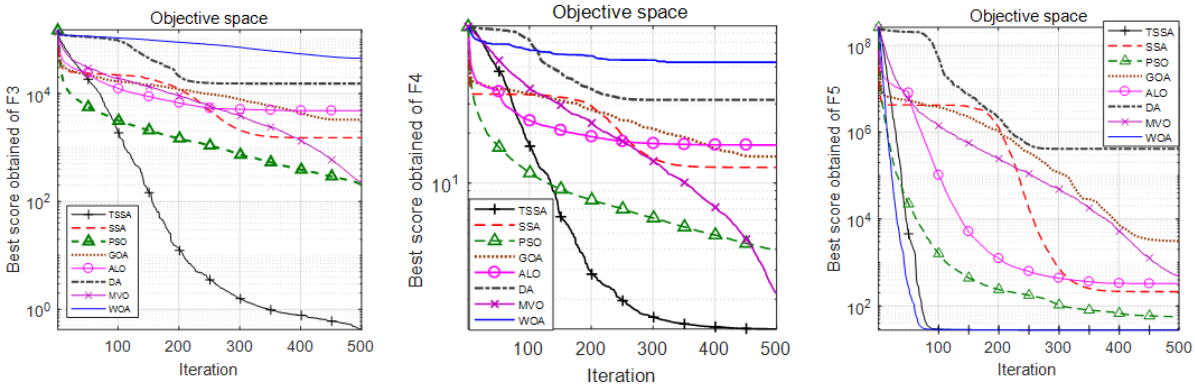

FIGURE 2. Convergence curves for algorithms over some unimodal benchmark functions

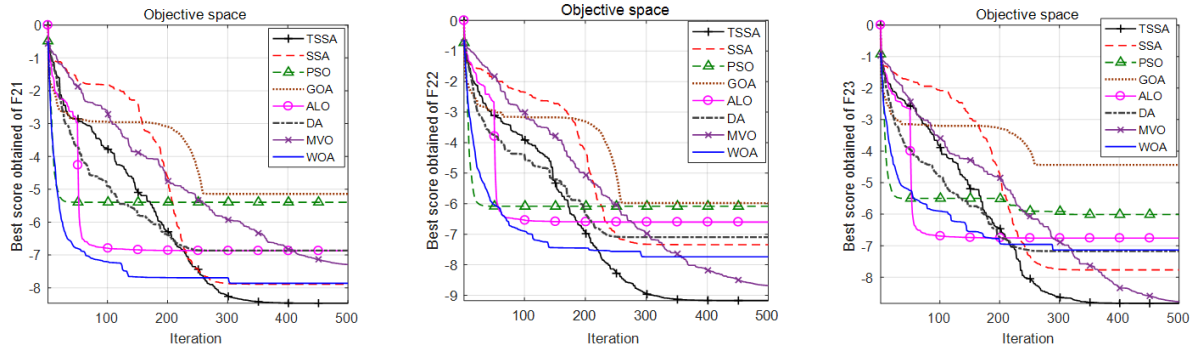

FIGURE 3. Convergence curves for algorithms over some multimodal benchmark functions

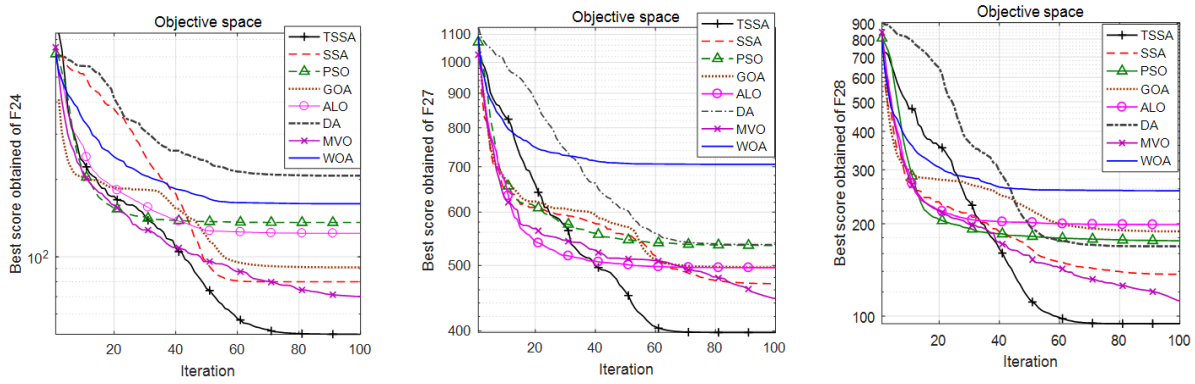

FIGURE 4. Convergence curves for algorithms over some composite benchmark functions

TABLE 9. p-values of the Wilcoxon rank-sum test over unimodal benchmark functions

Func	TSSA	SSA	PSO	GOA	ALO	DA	MVO	WOA
F_1	N/A	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11
F_2	N/A	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11
F_3	N/A	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11
F_4	N/A	3.0199e-11	1.8567e-09	3.0199e-11	3.0199e-11	3.0199e-11	1.0666e-07	4.1997e-10
F_5	N/A	2.3715e-10	0.20095	3.0199e-11	1.0937e-10	3.0199e-11	3.0199e-11	0.028378
F_6	N/A	3.0199e-11	3.3384e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	1.2493e-05
F_7	N/A	3.0199e-11	8.1527e-11	9.9186e-11	3.0199e-11	3.0199e-11	8.9934e-11	4.4592e-04
F_8	N/A	0.018368	1.9527e-03	2.7548e-03	2.31e-10	2.4386e-09	4.9818e-04	1.3289e-10
F_9	N/A	6.0584e-11	9.9068e-11	3.3342e-11	3.0161e-11	3.0161e-11	3.0161e-11	1.7175e-12
F_{10}	N/A	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	3.0199e-11	6.387e-12
F_{11}	N/A	0.016953	3.6704e-03	3.018e-11	3.3505e-08	3.018e-11	3.018e-11	1.2108e-12
F_{12}	N/A	3.0199e-11	0.30418	3.0199e-11	3.0199e-11	3.0199e-11	4.9752e-11	6.2828e-06
F_{13}	N/A	2.7726e-05	4.1997e-10	3.0199e-11	1.698e-08	3.0199e-11	2.1544e-10	1.411e-09
F_{14}	N/A	5.6865e-09	0.30351	1.9814e-11	0.6723	1.066e-05	0.036439	3.1753e-05
F_{15}	N/A	0.29047	3.0059e-04	8.6844e-03	0.046756	5.5699e-03	0.36322	0.4804
F_{16}	N/A	3.018e-11	1.2118e-12	3.0199e-11	3.0142e-11	2.6586e-06	3.0199e-11	0.070617
F_{17}	N/A	2.9916e-11	1.2118e-12	5.0769e-10	3.0142e-11	7.8759e-07	7.0881e-08	9.5207e-04
F_{18}	N/A	3.0199e-11	4.2911e-10	8.4848e-09	3.018e-11	1.8453e-09	1.1953e-03	0.016285
F_{19}	N/A	3.018e-11	2.3638e-12	2.0023e-06	3.0199e-11	5.8282e-03	0.28378	9.9186e-11
F_{20}	N/A	0.08418	6.8562e-04	8.5641e-04	2.4994e-03	0.082357	0.036439	0.019112
F_{21}	N/A	0.26433	0.058263	0.011228	0.59969	0.42344	2.8389e-04	2.278e-05
F_{22}	N/A	8.6359e-05	0.024103	2.9329e-03	0.14483	8.1706e-04	1.3936e-09	2.1224e-11
F_{23}	N/A	0.0238	4.1926e-03	3.4029e-04	6.6273e-03	0.083025	7.6973e-04	4.9426e-05
F_{24}	N/A	1.8368e-06	6.735e-03	2.6433e-03	0.66273	7.2446e-05	0.78446	7.7272e-06
F_{25}	N/A	8.3146e-08	2.1156e-05	0.29047	3.4783e-03	9.3519e-08	7.394e-05	0.12967
F_{26}	N/A	1.5079e-07	6.9048e-09	9.5238e-11	2.2463e-10	0.015873	6.9048e-03	7.9365e-03
F_{27}	N/A	0.44642	1.6687e-07	5.106e-05	5.9428e-07	1.5367e-03	2.1156e-06	1.6062e-11
F_{28}	N/A	9.0972e-05	6.2318e-07	1.0036e-03	0.073373	0.096985	9.8451e-04	0.017257
F_{29}	N/A	5.4762e-03	8.4127e-05	7.9365e-04	2.2674e-07	7.861e-03	5.8263e-06	2.2133e-05

results shown in Table 9 indicate that p-values between TSSA and other algorithms are less than 0.05 in most cases, which means the solutions of TSSA and the others could be regarded to be uncorrelated.

5. **Application on Container Deployment in Microservice Architecture.** In microservice architecture, microservices call each other through HTTP request or RPC communication. A simple example diagram is shown in Figure 5; in the diagram, microservice

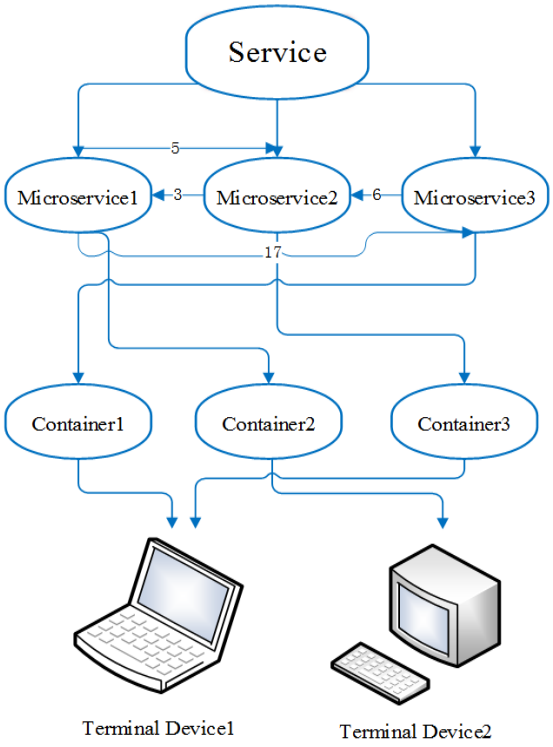

FIGURE 5. Diagram of container deployment relationships in microservice architecture

1 calls microservice 3 17 times and microservice 1 runs in container 2, which is deployed in terminal device 2. Since each microservice runs in separate containers, the number of calls between microservices is equivalent to the number of calls between containers. Therefore, when a container is deployed, number of calls between each container is a vital factor that must be considered for container deployment problem.

At the same time, considering the limited resources of terminal devices, the resource capacity of a single device is generally less than the sum of resources that all containers need to be pre-allocated [6]. Multiple containers are inevitably deployed on different terminal devices, and the communication overhead between devices has an impact on service completion time [13]. In the actual situation, containers in the same device communicate directly through network bridge; in this way, the communication overhead will be much smaller than the communication overhead between devices. Ideally, we tend to deploy containers with frequent call relationships to the same device or to devices with low communication overhead, and satisfying the resource constraints of containers and the terminal devices [14].

It can be seen that the container deployment problem in terminal device is NPC problem, and there are multiple solving algorithms [17]. Meanwhile, the dimensions of the solution sets become larger as the number of containers increases, which will enlarge the search spaces and make the optimization problem much more complex. For the single-objective problem which has multi-dimension, several meta-heuristic algorithms are proposed in recent years. These meta-heuristic algorithms are more effective than the traditional methods. In this work, the container deployment problem is a composite model with multimodal, which has multiple local optimal solutions. Based on the above benchmark functions, TSSA is good at solving these kinds of problems.

In the next chapter, TSSA described above and the recently proposed meta-heuristic algorithms are used to solve the container deployment problem in terminal devices. Fitness

of the problem is based on the number of calls between containers and communication overhead of terminal devices.

5.1. **Container deployment model.** In this chapter, we assume that there are N containers and M terminal devices for deploying containers. The container deployment model is described as follows.

1) The set of N resources is $\{R_1, R_2, \ldots, R_N\}$ and R_i represents the pre-allocated resources of the i-th container, the normalized resource of each container is as shown in Table 10.

Table 10. Pre-allocated resource of each container

Container	1	2	3	4	5	6	7	8	9	10
R: 1-10	1	2	3	4	5	6	5	4	3	2

2) We use $X(i)$ to represent the device, in which i-th container is deployed and $1 \leq X(i) \leq M$.
3) 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 CPU resource to reduce the dimension of resources, the normalized resource of each device is as shown in Table 11.

Table 11. Resource of each terminal device

Device	1	2	3	4	5	6	7
S: 1-7	8	7	9	10	11	6	12

4) A container can be operated at any device as long as the device can provide enough resource, which means $\sum_{i=1}^{N} R_i^x \leq Sx$ and R_i^x represents the i-th container is deployed in the x-th device.
5) The paper uses $F(X(i))$ to represent whether $\sum_{i=1}^{N} R_i^x \leq Sx$, when the condition is met, $F(X(i)) = 1$, otherwise $F(X(i)) = \text{inf}$.
6) The paper uses matrix $P[N, M]$ to represent the uniqueness of containers, $P_i^x = 1$ when i-th container is deployed in the x-th device, otherwise $P_i^x = 0$.
7) 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. To simplify the model, every microservice would be operated at the same speed.
8) Microservices running in container could call each other via HTTP request or RPC remote call, the matrix of calling times are $Calls[N, N]$ and $Calls_i^j$ represents the times of i-th container calls j-th container, the calling relation are as shown in Table 12, for example, number of calls between 6-th container and 5-th container is 14 in Table 12.
9) The communication overhead between each device is not negligible, the matrix of communication overhead is $Dealys[M, M]$, in which $Delays_x^y$ represents the communication overhead between x-th device and y-th device, for example, the normalization communication overhead between 3-rd device and 5-th device is 2 in Table 13.
10) Based on these assumptions, the completion time of all microservices mainly depends on the number of calls between containers and the communication overhead between devices. In this model, the fitness is calculated as follows:

$$fitness = \sum_{i=1}^{N}\sum_{j=1}^{N} P_i^{X(i)} * P_j^{X(j)} * Calls_i^j * Delays_{X(i)}^{X(j)} * F(X(i)) * F(X(j)) \tag{10}$$

where $P_i^{X(i)}$ is used to judge whether i-th container is deployed in the $x(i)$-th terminal device, $F(X(i))$ is applied to distinguish whether the resource $x(i)$-th terminal device is available.

TABLE 12. Number of calls to each container

$Calls_i^j$	**i: 1-10**									
j: 1-10	0	28	1	13	29	35	168	0	15	10
	28	0	17	4	8	1	20	0	0	12
	1	17	0	5	9	21	1	25	33	7
	13	4	5	0	7	3	6	8	14	9
	29	8	9	7	0	14	23	27	0	8
	35	1	21	3	14	0	9	11	13	17
	168	20	1	6	23	9	0	5	5	8
	0	0	25	8	27	11	5	0	23	29
	15	0	33	14	0	13	5	23	0	8
	10	12	7	9	8	17	8	29	8	0

TABLE 13. Normalization communication overhead relations

$Calls_i^j$	**i: 1-7**						
j: 1-7	0	1	2	1	2	2	1
	1	0	1	1	1	2	1
	2	1	0	1	2	1	2
	1	1	1	0	2	1	1
	2	1	2	2	0	1	1
	2	2	1	1	1	0	2
	1	1	2	1	1	2	0

5.2. **Result of TSSA on container deployment model problem.** In this work, 10 containers would be deployed in 7 terminal devices which have restricted resources. The number of calls to each container is as shown in Table 12 and the normalization communication overhead relations are as shown in Table 13.

In this work, 7 algorithms are compared with the proposed TSSA and each algorithm runs 500 iterations, 30 search agents are employed in these algorithms. In order to reduce the accidental factors, each algorithm is tested for 50 times and we record the statistical data such as average values, standard deviations and best values. The result of the experiment is listed in Table 14.

The result of the experiment demonstrates that the proposed TSSA outperforms the other algorithms in all respects. TSSA is capable of finding better solutions comparing with all the other algorithms and it has the best stability as well. Based on the best

TABLE 14. Result of container deployment

Algorithm	*Average*	Std	*Best*	*p-value*
TSSA	**427.8250**	**20.2014**	**361**	N/A
SSA	452.8000	23.7284	396	0.0086674
PSO	484.7250	36.3353	399	2.5364e-12
GOA	516.8750	26.4616	443	1.9205e-12
ALO	755.8500	62.4083	588	1.4285e-14
DA	510.7500	37.4068	420	3.9794e-13
MVO	437.8500	28.7354	380	0.068602
WOA	453.4000	32.5149	388	0.0031291

value of TSSA, we can find the corresponding terminal devices number of containers are $(1, 1, 7, 4, 4, 2, 1, 7, 7, 7)$.

6. **Conclusions.** The paper proposed a Transformed Salp Swarm Algorithm (TSSA), in which tent chaotic initialization distribution was used and we optimized the exploration mechanism in search space as well as exploitation mechanism. The performance of TSSA was tested on 29 benchmark functions and compared to 7 algorithms. The p-values of Wilcoxon rank-sum statistic tests were also used to compare relevance of solutions. When TSSA was applied to container deployment problem in microservice architecture, it could promote the performance of the original SSA and outperformed all the other comparison algorithms.

For future work, we intend to further improve the performance of TSSA, and our work will also focus on a larger scale of container cluster. In order to meet the increasing industrial demand, we expect that the optimized algorithm could be used in hundreds of containers and terminal devices. Besides, we will also modify model constraints to reduce the specification of the scene.

Acknowledgment. The work is partially supported by these projects:

- Strategic Leadership Project of Chinese Academy of Sciences: SEANET Technology Standardization Research System Development (Project No. XDC02010701);
- Innovation Youth Talent Project of Institute of Acoustics of Chinese Academy of Sciences: Research on Key Technologies of Embedded Container File System (Project No. Y754061601).

REFERENCES

[1] R. Zhao and X. Zhu, A review of the microservice architecture, *Journal of Network New Media*, vol.8, no.43(01), pp.58-61+65, 2019.

[2] D. Bernstein, Containers and cloud: From LXC to Docker to Kubernetes, *IEEE Cloud Computing*, vol.1, no.3, pp.81-84, 2014.

[3] R. Morabito, A performance evaluation of container technologies on Internet of Things devices, *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, San Francisco, CA, pp.999-1000, 2016.

[4] S. Mirjalili, A. H. Gandomi, S. Z. Mirjalili et al., Salp swarm algorithm: A bio-inspired optimizer for engineering design problems, *Advances in Engineering Software*, vol.114, pp.163-191, 2017.

[5] S. Arora and P. Anand, Chaotic grasshopper optimization algorithm for global optimization, *Neural Computing and Applications*, pp.1-21, 2018.

[6] J. Ha et al., A web-based service deployment method to edge devices in smart factory exploiting Docker, *International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju, pp.708-710, 2017.

[7] S. Mirjalili and A. Lewis, The whale optimization algorithm, *Advances in Engineering Software*, vol.95, pp.51-67, 2016.
[8] S. Saremi, S. Mirjalili and A. Lewis, Grasshopper optimization algorithm, *Advances in Engineering Software*, vol.105, pp.30-47, 2017.
[9] S. Mirjalili, Dragonfly algorithm: A new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems, *Neural Computing and Applications*, vol.27, no.4, pp.1053-1073, 2016.
[10] S. Mirjalili, S. M. Mirjalili and A. Hatamlou, Multi-verse optimizer: A nature-inspired algorithm for global optimization, *Neural Computing and Applications*, vol.27, no.2, pp.495-513, 2016.
[11] S. Mirjalili, The ant lion optimizer, *Advances in Engineering Software*, vol.83, pp.80-98, 2015.
[12] M. R. Asadi and S. M. Kouhsari, Optimal overcurrent relays coordination using particle-swarm-optimization methodology, *Proc. of IEEE Power Syst. Conf.*, pp.1-7, 2009.
[13] B. I. Ismail et al., Evaluation of docker as edge computing platform, *2015 IEEE Conference on Open Systems (ICOS)*, Bandar Melaka, pp.130-135, 2015.
[14] C. Kaewkasi and K. Chuenmuneewong, Improvement of container scheduling for docker using ant colony optimization, *The 9th International Conference on Knowledge and Smart Technology (KST)*, Chonburi, pp.254-259, 2017.
[15] M. Abdelbaky, J. Diaz-Montes, M. Parashar, M. Unuvar and M. Steinder, Docker containers across multiple clouds and data centers, *IEEE/ACM the 8th International Conference on Utility and Cloud Computing (UCC)*, Limassol, pp.368-371, 2015.
[16] M. Sureshkumar and P. Rajesh, Optimizing the docker container usage based on load scheduling, *The 2nd International Conference on Computing and Communications Technologies (ICCCT)*, Chennai, pp.165-168, 2017.
[17] C. Li, H. Zhang, H. Zhang and Y. Liu, Short-term traffic flow prediction algorithm by support vector regression based on artificial bee colony optimization, *ICIC Express Letters*, vol.13, no.6, pp.475-482, 2019.
[18] F. Wilcoxon, S. Katti and R. A. Wilcox, Critical values and probability levels for the Wilcoxon rank sum test and the Wilcoxon signed rank test, *Sel Tables Mathematical Statistics*, vol.1, pp.171-259, 1970.
[19] W. Xiao, H. Deng, Y. Sheng and L. Hu, Factored grey wolf optimizer with application to resource-constrained project scheduling, *International Journal of Innovative Computing, Information and Control*, vol.14, no.3, pp.881-897, 2018.