*Research Article*

International Journal of Distributed Sensor Networks

*International Journal of Distributed Sensor Networks*
2017, Vol. 13(11)

DOI: 10.1177/1550147717739831
journals.sagepub.com/home/ijdsn
SAGE

# A fruit fly optimization algorithm with a traction mechanism and its applications

**Xing Guo[1,2], Jian Zhang[1], Wei Li[1] and Yiwen Zhang[1]**

**Abstract**

The original fruit fly optimization algorithm, as well as some of its improved versions, may fail to find the function extremum when it falls far from the origin point or in the negative range. To address this problem, in this article, we propose a new fruit fly optimization algorithm, named as the traction fruit fly optimization algorithm, which is mainly based on the combination of "traction population" and dynamic search radius. In traction fruit fly optimization algorithm, traction population consists of the worst individual recorded in the iterative process, the individual in the center of the interval, and the best fruit flies individual through different transformations, which is used to avoid the algorithm stopping at a local optimal. Moreover, our dynamic search radius strategy will ensure a wide search range in the early stage and enhance the local search capability in the latter part of the algorithm. Extensive experiment results show that traction fruit fly optimization algorithm is superior to fruit fly optimization algorithm and its other improved versions in the optimization of extreme values of continuous functions. In addition, through solving the service composition optimization problem, we prove that traction fruit fly optimization algorithm can also obtain a better performance in the discrete environment.

**Keywords**

Fruit fly optimization algorithm, traction mechanism, service composition, function extremum, swarm intelligence

Date received: 23 February 2017; accepted: 14 September 2017

Handling Editor: Daniel Gutierrez-Reina

## Introduction

Algorithms based on swarm intelligence (SI) serve as an effective way of solving complex optimization problems.[1] Compared with traditional intelligent algorithms, such as a genetic algorithm (GA), SI shows a better behavior in terms of simplicity and effectiveness. Many SI-based algorithms have been widely used in both scientific and production fields.[2] These algorithms mimic the social behavior of species or natural mechanisms to find the best possible result for the given problem. The most popular SI algorithms include ant colony optimization (ACO),[3] which is inspired by the information exchange mechanism of ants when foraging; particle swarm optimization (PSO),[4] which simulates the foraging behavior of bird swarms; locust swarm algorithm (LSA),[5] which was extended from PSO; and the fish swarm algorithm (FSA),[6] which is inspired by the natural aggregation behavior of fishes.

However, all the algorithms mentioned above have their shortcomings. For example, due to excessive dependence on the parameter settings, PSO is prone to

[1]School of Computer Science and Technology, Anhui University, Hefei, China
[2]Key Laboratory of Intelligent Computing & Signal Processing, Anhui University, Hefei, China

**Corresponding author:**
Yiwen Zhang, School of Computer Science and Technology, Anhui University, Hefei 230601, China.
Email: 64757837@qq.com

**Table 1.** Simple comparison of six intelligent algorithms.

| | Amount of calculation | Complexity | Stability | Precision |
|---|---|---|---|---|
| ACO | Medium | High | Medium | High |
| FOA | Low | Low | Low | High |
| FSA | High | High | Medium | High |
| LSA | High | Medium | Medium | Medium |
| DE | High | High | High | Medium |
| PSO | Medium | Medium | Low | Low |

ACO: ant colony optimization; FOA: fruit fly optimization algorithm; FSA: fish swarm algorithm; LSA: locust swarm algorithm; DE: differential evolution; PSO: particle swarm optimization.

converge to a local optimum, ACO weakens its performance due to slow convergence, LSA requires additional steps to solve the problem, while the intricacy of FSA hinders itself from practical applications.

Fruit fly optimization algorithm (FOA) is a novel global optimization algorithm presented by Pan in 2011, which is inspired by the foraging behavior of fruit flies. Fruit flies have an astute sense of smell and sight in comparison with other similar species. By identifying and collecting the odors floating in the air, fruit files are able to locate food sources or fellow creatures, and then with the help of acute vision, they can be directed toward their destination.[7] This technique has been widely applied in real-world problems, such as the solution to function extremum,[8] the enhancement of the accuracy of enterprises financial crisis warning,[9] the improvement of generalized regression neural network, and the PID parameter optimization.[10]

The performance of FOA was compared with ACO, FSA, LSA, differential evolution (DE) algorithm,[11,12] and PSO, as shown in Table 1.

In general, FOA features are simple to implement, require less parameters to be set, and have strong local searching ability.

However, FOA also has its limitations. Specifically, the diversity of an individual is easy to lose, which leads to the premature problem of the algorithm. In addition, the original FOA cannot find the function extremum when it falls either in the negative range or not around the origin point. Furthermore, the local search capability of the FOA is weak because of the improper osphresis operation and vision operation.[13,14] To tackle the above problems, the chaotic mechanism is introduced into the optimization algorithm named chaotic fruit fly optimization algorithm (CFOA).[7] The major limitation of CFOA is that it cannot find the extremum in some complex benchmark functions. QK Pan et al.[8] proposed an improved fruit fly optimization algorithm (IFFO) by improving the method of individual generation. The performance of IFFO was analyzed comprehensively using six benchmark functions. However, its convergence is still very slow. Dan Shan et al.[14] proposed a new improved fruit fly algorithm named linear generation mechanism of candidate solution (LGMS)-FOA based on a linear mechanism, which directly substitutes the individual into a fitness function and generates a new fruit fly individual by dynamically adjusting the parameter $\omega$. After testing this algorithm in some benchmark functions, its main defect is its poor stability. The proposed algorithms in Zhang and Cui[15] and Wu et al.[16] draw upon the idea of GA, utilizing Cross and Mutation in FOA, but feature poor stability as well. The collaborative multiobjective fruit fly optimization algorithm (CMFOA) in Wu et al.[17] obtains new fruit fly individual through the normal random distribution, which cannot either find the extremum exactly in complex functions. In the work by Wang et al.,[18] the proposed algorithm named LP-FOA is combined with level probability policy and a new mutation parameter. This algorithm performs better on the extreme value problem of high-dimensional functions, but it is not experimented in the case of low-dimensional multimodal. In the work by Lv et al.,[19] an improved FOA based on hybrid location information exchange mechanism named HFOA is presented. This algorithm performs well in a continuous environment, but it is not tested in discrete environments.

To tackle the limitations of FOA, this article presents an improved FOA algorithm, called the traction fruit fly optimization algorithm (TFOA). The main contributions are as follows:

1. The deficiency of the original algorithm is analyzed in detail. The reason why FOA has a tendency to fall into a local optimum is explained.
2. TFOA can escape from a local optimum and achieve a much better global optimum using a traction population and a dynamic search radius mechanism.
3. Extensive experiments were conducted to evaluate the effectiveness of TFOA and showed that TFOA outperformed the original FOA as well as its other extensions in both the continuous and discrete environments.

The rest of the article is organized as follows. In section "FOA," the basic FOA is presented. The TFOA is described in detail in section "The Improved

Algorithm—TFOA." TFOA is evaluated experimentally in section "Experimental evaluations." Section "Conclusion" delivers the conclusion and expectations of this study.

## FOA

FOA can be summarized thoroughly with the following seven steps:

*Step 1*. Initialize related parameters, including population size, the maximum number of iterations, and the location of fruit flies ($X\_axis$, $Y\_axis$)

$$\begin{cases} X\_axis = rand(LR) \\ Y\_axis = rand(LR) \end{cases} \quad (1)$$

*Step 2*. Generate the location of an individual fruit fly in the swarm

$$\begin{cases} x_i = X\_axis + randValue \\ y_i = Y\_axis + randValue \end{cases} \quad (2)$$

where $randValue$ represents the range parameter.

*Step 3*. Calculate the distance between each fruit fly and the origin ($Dist_i$) and the odor concentration judgment value of each individual fruit fly ($S_i$)

$$Dist_i = \sqrt{x_i^2 + y_i^2} \quad (3)$$

$$S_i = \frac{1}{Dist_i} \quad (4)$$

*Step 4*. Calculate the odor concentration of each individual fruit fly

$$Smell_i = Function(S_i) \quad (5)$$

*Step 5*. Identify the optimal odor concentration value in the swarm (the maximum value is adopted here)

$$[bestSmell, bestIndex] = \max(Smell) \quad (6)$$

*Step 6*. Reserve maximal concentration value and $x$, $y$ coordinate. Then, the fruit fly swarm flies toward that location using vision

$$\begin{cases} Smellbest = bestSmell \\ X\_axis = x_{bestIndex} \\ Y\_axis = y_{bestIndex} \end{cases} \quad (7)$$

*Step 7*. Finish the algorithm if the number of generations or the required precision is reached; otherwise, go to step 2.

## The improved algorithm—TFOA

This section presents an improved FOA algorithm by introducing a traction mechanism and a dynamic search radius. The algorithm for the traction mechanism is stated in detail first, followed by an elaboration of the TFOA algorithm.

### *The generation of the traction population*

Figure 1 displays the basic optimization process of the algorithm. The algorithm will be trapped in a local optimum without finding the global minimum. TFOA records the worst individual of each iteration round. When the algorithm cannot find a better solution, using the recorded individuals to explore a larger solution space in the opposite direction of the evolution, it will find a better solution, as shown in Figure 2. When the algorithm finds a better solution, it will change the location of the fruit fly population and restart the basic

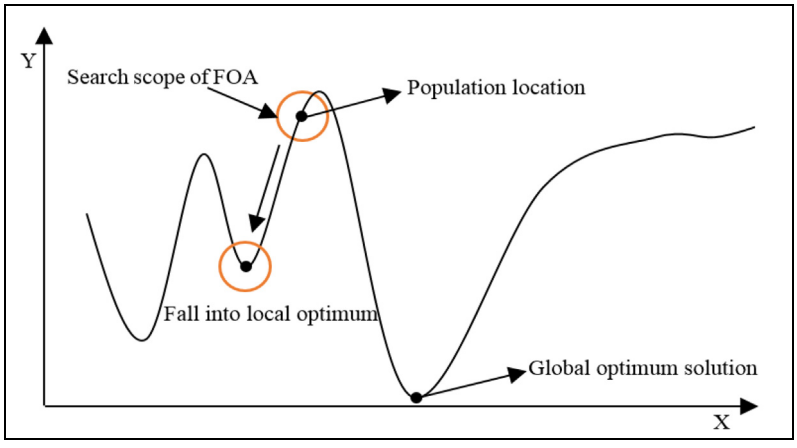


**Figure 1.** FOA optimization process.

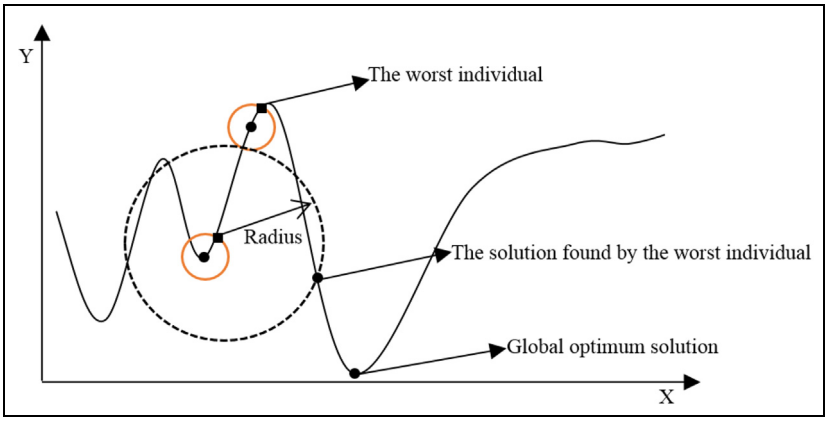


**Figure 2.** The method used in TFOA to escape a local optimum.

search in the FOA algorithm, thus finding the real minimum value in the end.

This article applies the above to the generation of a traction population. The purpose of the traction population is to guide the algorithm to jump out of the local optimum. To further enhance the diversity of the traction population and improve the convergence rate, the algorithm also creates some individuals around the center solution space. The entire generation process of the traction population is demonstrated in Algorithm 1.

*searchRadius* represents the search radius of the traction population. *bestInd* represents the best individual when the algorithm falls into a local optimum, *badPop* is the worst individual in each iteration, *upper* and *lower* are the upper and lower bounds of each dimension, *sizepop* represents the size of the fruit fly population, *numCenter* refers to the number of individuals produced by center searching. $Tpop_i^j$ represents the *i*th individual's *j*th dimension in the traction population, *rand* represents a random number within the range [0, 1], and *RandomDirection* is a random integer in $\{-1, 0, 1\}$.

The above algorithm first calculates the direction of evolution according to equation (8) and then follows the transformation rule in line 5 to generate a new individual. After finding the center position (line 11), multiple individuals are randomly generated around it. The final output is the population (*Tpop*) which is used to guide the FOA algorithm to jump out of the local optimum

$$direction = \begin{cases} 1 \; bestInd^j < badPop_i^j \\ -1 \; bestInd^j > badPop_i^j \\ RandomDirection \; bestInd^j = badPop_i^j \end{cases} \quad (8)$$

## The dynamic search radius strategy

In the original FOA, the search radius is fixed and always searches in one direction. This leads to a slow convergence speed and a tendency to fall into a local optimum. An appropriate search radius is located in the early stages of the algorithm to ensure that the algorithm finds as many solutions as possible. In the latter stages of the algorithm, we can find new solutions and an optimal solution. Therefore, the search radius should be closely related to the number of iterations and the size of the search interval.

A new method for calculating the search radius presented in this article is shown in equation (9) in the process of generating new individuals by adding a random direction to avoid single direction search

$$radius = radiusMax * e^{\lg\left(\frac{radiusMin}{radiusMax}\right) * \frac{g}{maxgen}} \quad (9)$$

where *g* denotes the number of iterations while the maximum and minimum radii are represented as *radiusMin* and *radiusMax*, respectively. The values of *radiusMin* and *radiusMax* are determined by the size of the search interval.

New individuals are generated according to equation (10), where $individual_i^m$ represents the *m*th dimension for the *i*th individual and *bestInd* represents the fruit fly population position

$$individual_i^m = bestInd^m + (radius * RandDirection) \quad (10)$$

## Implementation of the TFOA algorithm

By combining the two strategies mentioned above with the basic FOA algorithm, Algorithm 2 illustrates the entire process of the TFOA via an example of finding the minimum value of a function.

Since the convergence speed is greatly affected by the initial position, a new method for locating the initial population position is applied in TFOA. According to equations (11) and (12), a chaotic mechanism is used to generate the first fruit fly population. The best individual is then chosen as the group's initial position

**Algorithm 1.** Generation algorithm of traction population (GATP).

```
Input:
  searchRadius, badPop, upper, lower, numCenter, bestInd
Output:
  The traction population: TPop
 1 [rows, cols] ← size(badPop)
 2 for each i<=rows do
 3    for each j<=cols do
 4       Get direction according to equation (8)
 5       Tpop_i^j←badPop_i^j + searchRadius*direction;
 6       if Tpop_i^j out bound then
 7          Tpop_i^j←lower + (upper-lower)*rand;
 8       end if
 9    end for
10 end for
11 midposition = getCenter ();
12 for each i<=numCenter do
13    for each j<=cols
14       midPop_i^j←midPosition^i + searchRadius*Random
         Direction;
15       if midPop_i^j out bound then
16          midPop_i^j←lower + (upper-lower)*rand;
17       end if
18    end for
19 end for
20 Tpop = [Tpop;midPop];
```

$$\begin{cases} individual_0 = rand(1, dimension) \\ individual_i = individual_{i-1}*(\boldsymbol{E} - individual_{i-1})*4 \end{cases} \tag{11}$$

$$individual_i = lower + individual_i*(upper - lower) \tag{12}$$

where $\boldsymbol{E}$ is a vector whose element is 1.

The execution sequence of TFOA is performed as follows:

*Step 1*. Produce fruit fly population according to equation (10), calculate fitness of fruit fly individual and record the worst individual into *badPop* (lines 3–6).

*Step 2*. Update global best fitness value (*GbestFitness*) and population position (*bestInd*). Update *count* instead if *GbestFitness* does not require updating (lines 7–12).

*Step 3*. If the algorithm ends prematurely, execute Algorithm 1 to jump out of the local optimum. Once a better individual is found, update the population position and search around it, otherwise, change the search radius to continue to look for a better solution. Re-record the worst individual in every iteration when the traction operation finishes (lines 13–24).

*Step 4*. Repeat steps 1–3 until the number of generations or the required precision is reached.

**Algorithm 2.** Traction fruit fly optimization algorithm (TFOA).

```
Input:
  maxgen, sizepop, dimension, upper, lower, countMax,
radiusMax, radiusMin
Output:
  Function minimum value: GbestFitness
 1 Init ();
 2 for each g<=maxgen do
 3    Create individual according to equation (10)
 4    fitness←CalFitness(individual);
 5    [bestIndex, bestFitness]←Min(fitness);
 6    badPop←RecordBad ();
 7    if GbestFitness > bestFitness then
 8       Update global optimum fitness (GbestFitness) and
         Population location (bestInd);
 9       count = 0;
10    else
11       count←count + 1;
12    end if
13    if count>=countMax then
14       for i = maxgen-g: maxgen
15          radius←radiusMax*exp (log (radiusMin/
            radiusMax)*((i-maxgen + g + 1)/g));
16       TPop←GATP (radius, badPop, upper, lower, numCenter,
         bestInd);
17       fitness←CalFitness(Tpop);
18       [bestIndex, bestFitness]←Min(fitness);
19       if GbestFitness > bestFitness then
20          Update global optimum fitness and Population
            location, Stop iteration.
21       end if
22    end for
23    Set variable (count) to zero and empty worst individual set
      (badPop).
24    end if
25 end for
```

## Time complexity analysis of TFOA

The following is a brief analysis of the time complexity of TFOA, mainly focusing on the effect of traction on the performance of the algorithm. When the algorithm falls into a local optimum, it is necessary to carry out a traction operation; this operation is performed up to $maxgen/countMax$ times. Traction operation requires the generation of a traction population. The time complexity is $O(TPopSize * maxtow)$, where $TPopSize$ is the size of the traction population and $maxtow$ is the maximum iteration number of the traction population. Therefore, the time complexity of the algorithm can be defined as follows

$$O\left(\frac{TPopSize * maxgen * maxtow}{countMax}\right)$$

The time complexity of TFOA is higher than the original algorithm due to the time taken to jump out of a local optimum. As the traction mechanism is used, the algorithm needs to generate traction population to search again. For instance, if the algorithm converges

at the $n$th iteration, TFOA will perform traction operations at most *maxtow* times. In other words, TFOA will try to spend more time finding better solutions. In order to minimize the influence of the traction operation on the search speed of the algorithm, the dynamic step strategy and the chaotic mechanism are introduced into the TFOA to improve the search speed. Compared with the original algorithm, the overall searching speed is a little slower, yet both the accuracy and global search ability are improved.

## Experimental evaluations

Experiments in continuous and discrete environments were carried out to evaluate the performance of the TFOA. In continuous environments, we evaluated the stability and global optimization ability of TFOA in comparison with the most recent improved versions of FOA, including IFFO, CFOA, and LGMS-FOA, based on 10 benchmark functions. In discrete environments, through the solution of the web service composition problem, the optimization ability of the algorithm is proven.

### *Algorithm performance test in continuous environments*

This section compares TFOA with FOA and three existing improved versions of FOA, including IFFO, CFOA, and LGMS-FOA, based on 10 benchmark functions. The searching ability of TFOA in continuous environments is evaluated.

*Experimental setup and parameter configuration.* The experiments were conducted on a machine running Windows 7 (64 bit) with Intel® Core™ i5-2430M CPU 2.40 GHz and 4G of RAM. MATLAB R2014a was implemented. The parameters of each algorithm are listed in Table 2. The experimental parameters of the other four algorithms are set according to the corresponding papers. The population size of each algorithm was set to 50, and the number of iterations was set to 500 times.

*Benchmark functions.* To evaluate the ability of TFOA in solving function extrema, 10 test functions whose local extremum are diversely distributed were selected, which is demanding for algorithm's performance of optimization. The 10 test functions are listed in Table 3.

*Experimental results.* To evaluate the performance in finding the extreme value of the above 10 functions, means and variance yielded after the algorithms had run 50 times were, respectively, recorded and are shown in Table 4. The iterative optimization curves were drawn accordingly, as shown in Figures 3–12.

The experimental results of F1, F2, F3, and F4 show that all algorithms except for TFOA cannot accurately find the extreme value of the functions. The reason is that these test functions have many "local extremum points," which cause the algorithms to easily fall into a local optimum. Additionally, these function extrema are hard to find because they are far from the origin. On the contrary, the results prove that TFOA can find the function extrema values with small variances, which demonstrates its performance in terms of finding low-dimensional function extrema.

As for F5, F6, F7, F8, F9, and F10, six high-dimensional functions, the experimental results show that the algorithms IFFO and CFOA exhibited the premature phenomenon but that TFOA succeeded in finding the extremum. Relatively speaking, TFOA outperforms the original FOA, IFFO, and CFOA in

**Table 2.** Parameter settings.

| Algorithm | Parameter | Value | Significance | Reference |
|---|---|---|---|---|
| FOA | V | 1 | The range size of fly swarm | Mitić et al.[7] |
| TFOA | radiusMax | (upper-lower)/6 | Maximum search radius | – |
| | radiusMin | 0.00001 | Minimum search radius | |
| | countMax | 40 | Number of not updated | |
| | limitBadSize | 40 | Size of bad population | |
| IFFO | $\lambda_{max}$, $\lambda_{min}$ | (UB-LB)/2, $10^{-6}$ | Maximum radius<br>Minimum radius | Pan[9] |
| CFOA | $\cos\left(i \cdot \frac{1}{\cos(x_i)}\right)$ | Chebyshev | Chaotic map function | Pan et al.[8] |
| LGMS_FOA | $n$, $\omega_0$, $\alpha$ | 0.005, 1, 0.95 | Search coefficient<br>Initial weight<br>Weight coefficient | Shan et al.[14] |

FOA: fruit fly optimization algorithm; TFOA: traction fruit fly optimization algorithm; IFFO: improved fruit fly optimization algorithm; CFOA: chaotic fruit fly optimization algorithm; LGMS_FOA: linear generation mechanism of candidate solution_FOA.

**Table 3.** Benchmark functions.

| Function | Range and dimension | Minimum |
|---|---|---|
| $f_1(x,y) = -x\cos(2\pi y) - y\sin(2\pi x)$ | $x \in [-2,2]$, $y \in [-2,2]$, dimension = 2 | −3.75 |
| $f_2(x,y) = x^2 + y^2 - 0.3*\cos(3\pi x) + 0.3*\cos(4\pi y) + 0.3$ (Bohachevsky) | $x \in [-10,10]$, $y \in [-10,10]$, dimension = 2 | −0.24 |
| $f_3(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$ $\times[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$ (Goldstein and Price) | [−2, 2], dimension = 2 | 3 |
| $f_4(x) = -\cos x_1 \times \cos x_2 \times e^{-(x_1-\pi)^2-(x_2-\pi)^2}$ (Easom) | [−10, 10], dimension = 2 | −1 |
| $f_5(x) = \sum_{i=1}^{n} x_i^2$ (Sphere) | [−10, 10],dimension = 30 | 0 |
| $f_6(x) = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10)$ (Rastrigin) | [−5.12, 5.12], dimension = 30 | 0 |
| $f_7(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$ (Ackley) | [−32, 32], dimension = 30 | 0 |
| $f_8(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$ (Griewank) | [−600, 600], dimension = 30 | 0 |
| $f_9(x) = \sum_{i=1}^{n} ix_i^4 + rand(\,)$ (Quartic) | [−1.28, 1.28], dimension = 50 | 0 |
| $f_{10}(x) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2$ (Step) | [−100, 100], dimension = 50 | 0 |

**Table 4.** Mean and variance of algorithm.

| | TFOA | FOA | IFFO | LGMS_FOA | CFOA |
|---|---|---|---|---|---|
| $F_1$ | Std: **3.3292e−04** | Std: 0.7366 | Std: 0.2972 | Std: 0.0241 | Std: 0.4458 |
| | Mean: **−3.7561** | Mean: −1.1095 | Mean: −2.8748 | Mean: −2.0074 | Mean: −2.8349 |
| $F_2$ | Std: **5.3477e−04** | Std: 0.1236 | Std: 2.1055 | Std: 0.1128 | Std: 0.4552 |
| | Mean: **−0.2397** | Mean: −0.0392 | Mean: 0.6283 | Mean: 0.0442 | Mean: 0.4311 |
| $F_3$ | Std: **9.4182e−11** | Std: 0.0124 | Std: 18.8023 | Std: 0.2402 | Std: 9.3036 |
| | Mean: **3.0000** | Mean: 602.4381 | Mean: 20.5914 | Mean: 32.6989 | Mean: 12.8737 |
| $F_4$ | Std: **2.5742e−11** | Std: 0.4075 | Std: 0.4437 | Std: 0.0049 | Std: 0.2715 |
| | Mean: **−1.0000** | Mean: −0.2566 | Mean: −0.4045 | Mean: −0.9941 | Mean: −0.5682 |
| $F_5$ | Std: 5.5238e−07 | Std: **2.1958e−07** | Std: 103.0907 | Std: 0.0212 | Std: 30.6464 |
| | Mean: **1.0987e−05** | Mean: 3.6316e−05 | Mean: 341.4249 | Mean: 0.1646 | Mean: 189.5328 |
| $F_6$ | Std: **1.3261e−07** | Std: 4.6339e−05 | Std: 29.8381 | Std: 1.0689 | Std: 29.6298 |
| | Mean: **9.7135e−08** | Mean: 0.0072 | Mean: 192.1436 | Mean: 8.6716 | Mean: 460.7571 |
| $F_7$ | Std: 0.9072 | Std: **1.4641e−05** | Std: 0.8158 | Std: 0.1751 | Std: 0.3593 |
| | Mean: 1.0308 | Mean: **0.0034** | Mean: 17.6819 | Mean: 1.8399 | Mean: 17.4034 |
| $F_8$ | Std: 0.0068 | Std: **2.9413e−07** | Std: 108.4951 | Std: 0.0141 | Std: 19.9989 |
| | Mean: 0.0123 | Mean: **1.5639e−05** | Mean: 316.9595 | Mean: 1.0768 | Mean: 174.8681 |
| $F_9$ | Std: **1.2378e−06** | Std: 0.0010 | Std: 30.9859 | Std: 8.2925e−05 | Std: 3.1481 |
| | Mean: **1.5555e−06** | Mean: 0.0030 | Mean: 64.3927 | Mean: 7.8192e−05 | Mean: 11.2370 |
| $F_{10}$ | Std: **0** | Std: 0 | Std: 7.9307e + 03 | Std: 2.2113 | Std: 2.7059e + 03 |
| | Mean: **0** | Mean: 0 | Mean: 3.8166e + 04 | Mean: 10.9000 | Mean: 1.8517e + 04 |

FOA: fruit fly optimization algorithm; TFOA: traction fruit fly optimization algorithm; IFFO: improved fruit fly optimization algorithm; CFOA: chaotic fruit fly optimization algorithm; LGMS_FOA: linear generation mechanism of candidate solution_FOA.
Bold numbers indicates that the corresponding algorithm has the best experimental result.

terms of not only the accuracy of the solution but also the variance.

By observing the optimization curve of TFOA, it can be seen that when converging, the number of iterations is higher than other algorithms due to the traction mechanism. Specifically speaking, TFOA needs a certain number of iterations to perform traction operations to find a better solution. The other three algorithms lack proper measures to deal with the problem that the algorithm falls into the local optimum, so

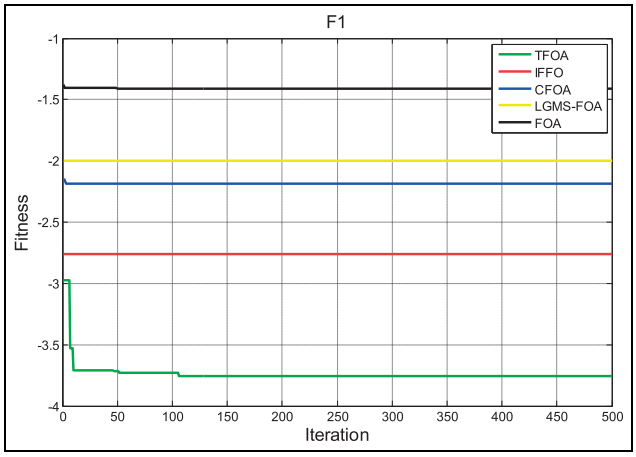


**Figure 3.** Function $F_1$ optimization curve.

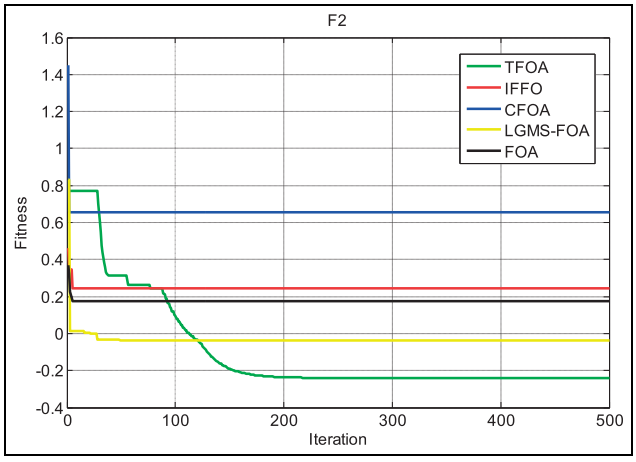


**Figure 4.** Function $F_2$ optimization curve.

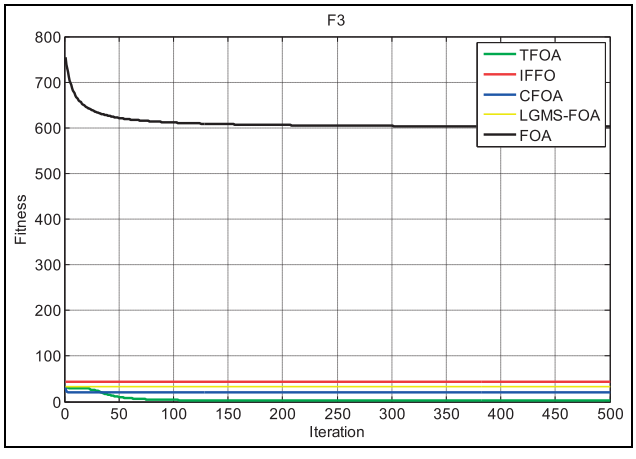


**Figure 5.** Function $F_3$ optimization curve.

they converge to a local extremum and fail to find the global optimal solution.

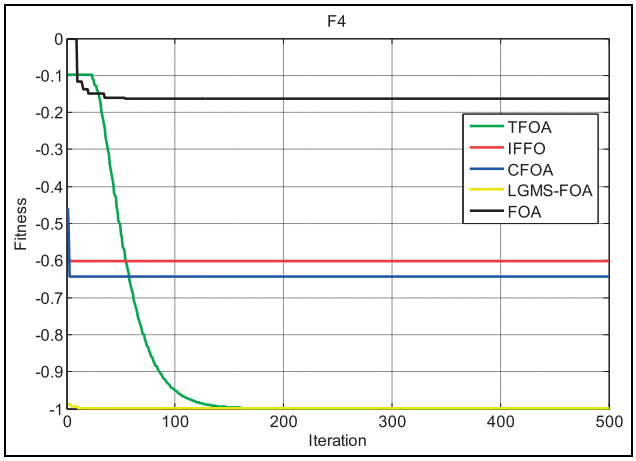


**Figure 6.** Function $F_4$ optimization curve.

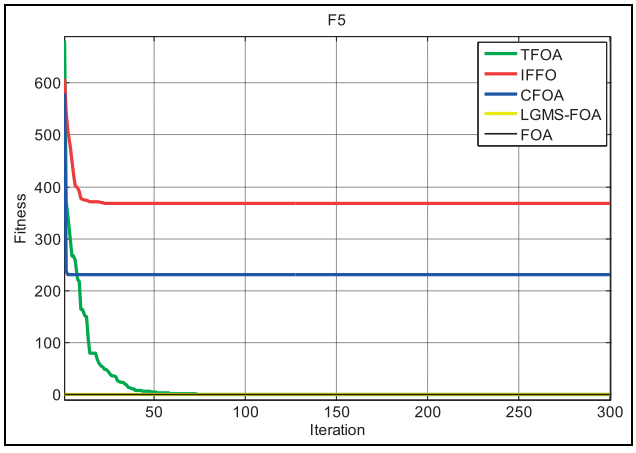


**Figure 7.** Function $F_5$ optimization curve.

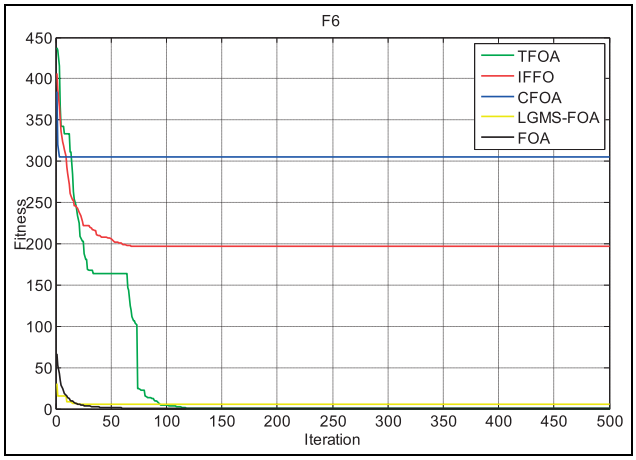


**Figure 8** Function $F_6$ optimization curve.

In conclusion, it can be seen that for a multimodal, low-dimensional function, TFOA can perform both

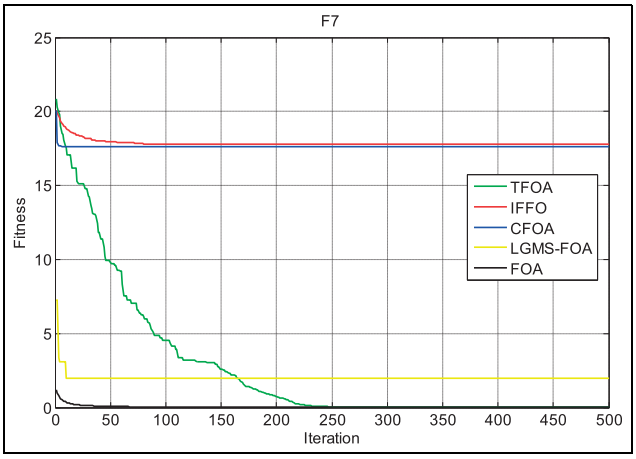


**Figure 9.** Function $F_7$ optimization curve.

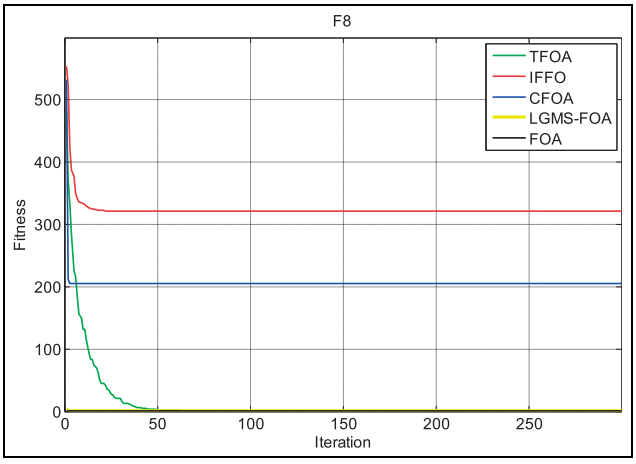


**Figure 10.** Function $F_8$ optimization curve.

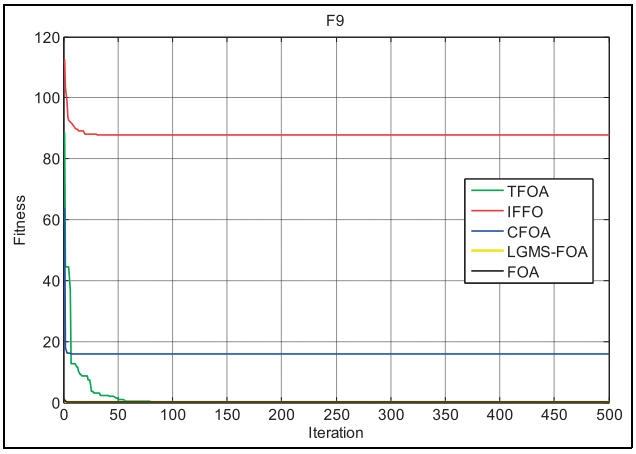


**Figure 11.** Function $F_9$ optimization curve.

accurately and stably. As for high-dimensional functions, TFOA displays a comparatively better performance as well. Therefore, the effectiveness of the improved FOA algorithm, TFOA, has been proven.

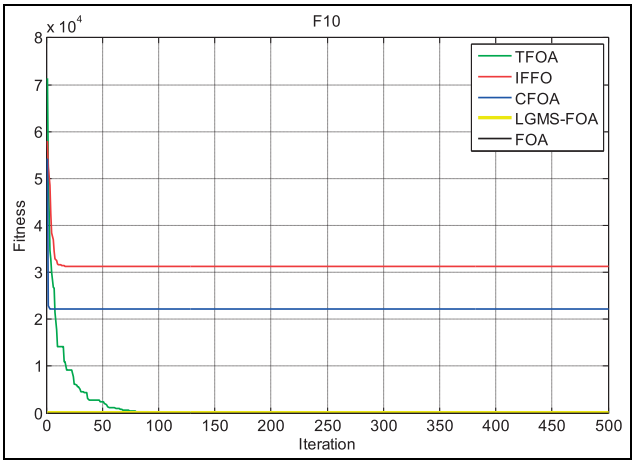


**Figure 12.** Function $F_{10}$ optimization curve.

## *Algorithm performance test in discrete environments*

Web services are software components designed to support interoperable machine-to-machine interaction over a network.[20] With the number of web services increasing, the selection of a web service for a composition is not an easy problem because many web services are equal in functionality but different in Quality of Service (QoS).[21–30] Web service composition aims to choose a combination service which meets the needs of a user's service quality. Currently, special attention has been given to the solution of the web service composition problem through SI algorithms, with the proposal of algorithms such as PSO, ACO, and firework. However, the convergence rate and optimization ability of the above algorithms can be further improved. In this article, the effectiveness of TFOA is evaluated and compared with the results of dynamic discrete particle swarm optimization (DDPSO) proposed by Zhang et al.[26]

*Modeling service composition algorithm.* Assume that task $T = \{ST^1, ST^2, \ldots, ST^i, \ldots, ST^N\}$ denotes that the task $T$ is composed of $N$ subtasks, where $ST^i$ represents the $i$th subtask. $WSS^i = \{ws_1^i, ws_2^i, \ldots, ws_j^i, \ldots, ws_{Ti}^i\}$ denotes the candidate service set for the $i$th subtask,[23] where $ws_j^i$ represents the $j$th candidate service for the $i$th subtask. Each candidate service has $M$ performance metrics which can be illustrated as follows

$$Q^{wss^i} = \begin{bmatrix} q_1^{ws_1^i} & q_2^{ws_1^i} & \cdots & q_r^{ws_1^i} & \cdots & q_M^{ws_1^i} \\ q_1^{ws_2^i} & q_2^{ws_2^i} & \cdots & q_r^{ws_2^i} & \cdots & q_M^{ws_2^i} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ q_1^{ws_l^i} & q_2^{ws_l^i} & \cdots & q_r^{ws_l^i} & \cdots & q_M^{ws_l^i} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ q_1^{ws_{Ti}^i} & q_2^{ws_{Ti}^i} & \cdots & q_r^{ws_{Ti}^i} & \cdots & q_M^{ws_{Ti}^i} \end{bmatrix}$$

where $q_r^{ws_l^i}$ represents the $r$th metric of the $l$th candidate service for the $i$th subtask. However, the performance requirements of the candidate services vary with the situation. For example, the response time should be as small as possible, while the reliability should be as large as possible. Therefore, the performance indicators for the candidate services need to be further normalized. In this article, the performance metrics of candidate services are normalized according to equation (13)

$$S^{wss_i} = \begin{cases} (q_r^{\max} - q_r^{ws_l^i})/(q_r^{\max} - q_r^{\min}) \ negative \\ (q_r^{ws_l^i} - q_r^{\min})/(q_r^{\max} - q_r^{\min}) \ positive \end{cases} \quad (13)$$

In TFOA, individual coding is expressed as $\{x_1, x_2, x_3 \ldots x_N\}$, in which each element is an integer ranging from 1 to $T_i$, where $x_i$ represents that the $i$th subtask uses the $x_i$th candidate service.

A fitness function is employed to evaluate the usefulness of web service composition. As shown in equation (14), the fitness function used in this article is as follows

$$fitness = \sum_{r=1}^{mq} \omega_r \times Q_{cs}^r \quad (14)$$

where $\omega_r$ is the weight of the $r$th QoS metric, $mq$ is the total number of the QoS metrics, and $Q_{cs}^r$ is the $r$th QoS metric.

In a discrete environment, using a dynamic radius will generate a decimal and so a crossover operation is applied instead. The operation process is shown in Figure 13, where $r1$ and $r2$ are two different random numbers whose values range from 1 to $N$. If the code of an individual equal to the best individual exists, then we will randomly select two different locations of the individual for re-encoding.

A dynamic radius strategy will be used continually in the generation of a traction population. If a decimal number is generated in the calculation process, we will round the decimal number to a positive integer.

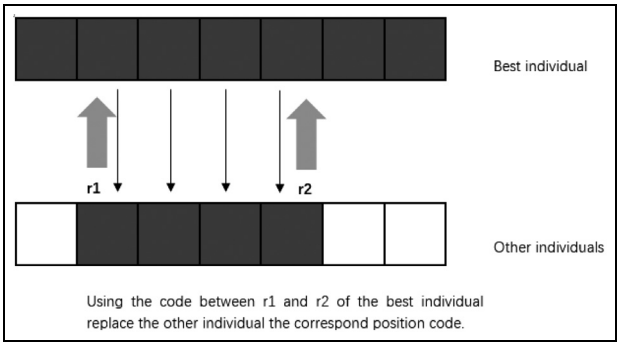


**Figure 13.** Learning from the best individual in TFOA.

*Experimental simulations.* *Response time*, *availability*, *throughput*, and *reliability* are the four metrics selected for the candidate service in this article. To conduct the experiments in a fair manner, the same set of parameters is required. The weights of these four metrics are 0.4, 0.1, 0.3, and 0.2, and the number of candidate services for each subtask is the same. The model of the web service composition is a sequential model. The performance indexes of the candidate services were normalized in advance. The service data were collected from a real Internet environment.

The experiment will mainly focus on the performance of the algorithm, including the influence of the traction group, the optimization ability, the convergence speed, and the stability.

To verify how much the algorithm can be improved by a traction group, a comparative experiment was carried out. The number of candidate services was set to 500. The number of subtasks was set to 7 and the iteration number was 800. As shown in Figure 14, TFOA with a traction group significantly outperforms TFOA without a traction population, which proves that the performance of TFOA has been greatly improved.

In the second controlled experiment, TFOA was compared with DDPSO proposed by Zhang and Cui[15] and the original FOA algorithm with 50 independent runs. The number of iterations (800) and population size (50) were fixed under an environment with different candidate service numbers (50, 100, 150, 200, 250, 300, 350, 400, 450, and 500). The results are presented in Figure 15.

It can be seen that for a different number of candidate service, the final fitness values of TFOA are noticeably higher than that of the other two algorithms.

The third experiment mainly focused on the volatility of the algorithm. SI algorithms often have the problem of large fluctuations in the results. In an

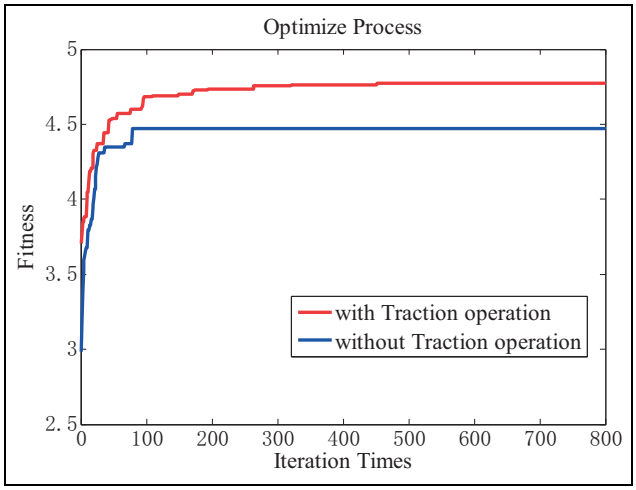


**Figure 14.** Effect of traction on the performance of TFOA.

**Table 5.** Variance of the fitness of the algorithm under different candidate service numbers.

| Service number | 50 | 100 | 150 | 200 | 250 | 300 | 350 | 400 | 450 | 500 |
|---|---|---|---|---|---|---|---|---|---|---|
| TFOA | 3.5527e−15 | 0.0012 | 0.0034 | 0.0056 | 0.0075 | 0.0085 | 0.0128 | 0.0098 | 0.0162 | 0.0132 |
| DDPSO | 0.1520 | 0.1592 | 0.1769 | 0.1525 | 0.1516 | 0.1744 | 0.1487 | 0.1508 | 0.1480 | 0.1420 |
| FOA | 0.1822 | 0.1872 | 0.1521 | 0.2104 | 0.1376 | 0.1770 | 0.1987 | 0.1678 | 0.2218 | 0.1439 |

FOA: fruit fly optimization algorithm; TFOA: traction fruit fly optimization algorithm; DDPSO: dynamic discrete particle swarm optimization.

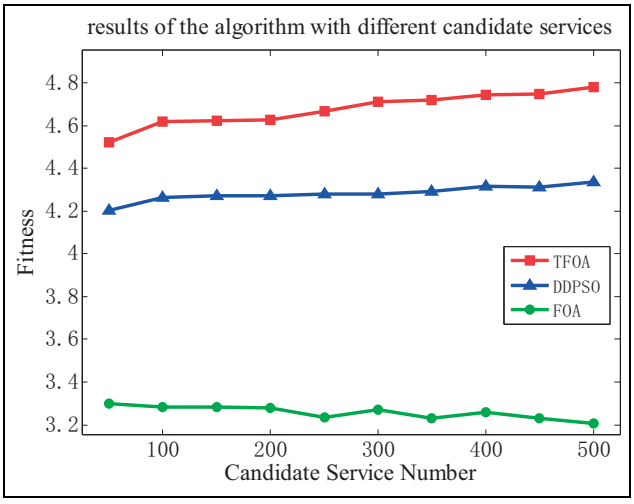


**Figure 15.** Optimization results of the algorithm with different candidate services.

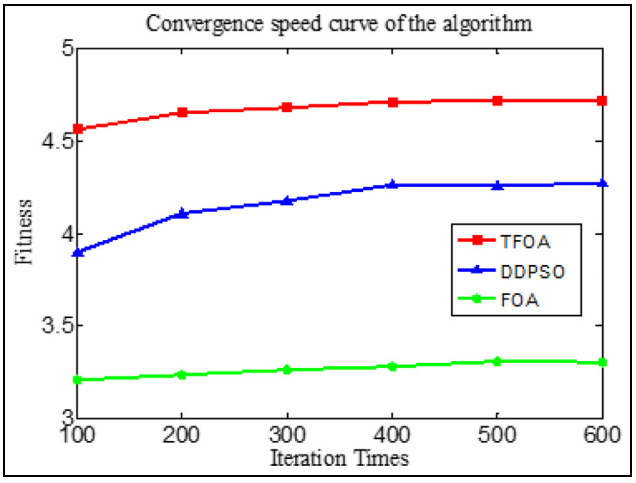


**Figure 16.** Optimization results of the algorithm with different iteration times.

environment that was the same as that for the second experiment, the fitness variances obtained with 50 independent runs are shown in Table 5. It can be seen that TFOA, with a smaller variance, is more stable than the other two algorithms.

The fourth set of experiments was used to measure the convergence speed of the algorithm. The number of candidate services (350) was fixed. The experimental results are shown in Figure 16. By examining the experimental results shown in Figure 16, it can be concluded that the convergence rate of TFOA is higher than that of the DDPSO algorithm and the original FOA algorithm.

The conclusion derived from the above four experiments is that in terms of the solution stability, convergence speed, and solution result, TFOA is superior to DDPSO and the original FOA algorithm. Therefore, the usefulness of TFOA in solving the web service composition problem is proven.

## Conclusion

In this article, a new improved fruit fly optimization algorithm named TFOA is proposed. TFOA uses a "traction population" to guide the algorithm to jump out of the local optimum. In addition, the convergence speed and the solving ability of the algorithm are also improved by combining dynamic step size and chaotic mechanism. The performance of the proposed algorithm is tested in both continuous and discrete environments. By solving the extreme value of various functions and through comparisons with three other improved optimization algorithms, that is, LGMS-FOA, IFFO, and CFOA, it can be seen that TFOA features a high accuracy, wide application range, and high stability. Compared with the DDPSO algorithm when solving the problem of web service composition, TFOA is better in terms of accuracy and speed of convergence.

In the case of service composition, the result yielded by TFOA algorithm is unqualified when the number of candidate services exceeds 10,000. Therefore, in addition to reducing TFOA's time complexity, future work will focus on improving its performance especially when it comes to a large data set.

### Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

## Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the National Key Technology R&D Program (2015BAK24B01).

## References

1. Yang XS. Swarm intelligence based algorithms: a critical analysis. *Evol Intell* 2014; 7(1): 17–28.
2. Molina D and Herrera F. Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems. *Soft Comput* 2011; 15(11): 2085–2087.
3. Reed M, Yiannakou A and Evering R. An ant colony algorithm for the multi-compartment vehicle routing problem. *Appl Soft Comput* 2014; 15(2): 169–176.
4. Liu Z, Mao C and Luo J. A three-domain fuzzy wavelet network filter using fuzzy PSO for robotic assisted minimally invasive surgery. *Knowl Bas Syst* 2014; 66(1): 13–27.
5. Cui L and Deng J. A novel locust swarm algorithm for the joint replenishment problem considering multiple discounts simultaneously. *Knowl Bas Syst* 2016; 111(6): 51–62.
6. Wang CR, Zhou CL and Ma JW. An improved artificial fish-swarm algorithm and its application in feed-forward neural networks. In: *Proceedings of the international conference on machine learning and cybernetics*, Guangzhou, China, 18–21 November 2005, pp.2890–2894. New York: IEEE.
7. Mitić M, Vuković N, Etrović M, et al. Chaotic fruit fly optimization algorithm. *Knowl Bas Syst* 2015; 89(3): 446–458.
8. Pan QK, Sang HY, Duan JH, et al. An improved fruit fly optimization algorithm for continuous function optimization problems. *Knowl Bas Syst* 2014; 62(5): 69–83.
9. Pan WT. A new fruit fly optimization algorithm: taking the financial distress model as an example. *Knowl Bas Syst* 2012; 26(2): 69–74.
10. Han J, Wang P and Yang X. Tuning of PID controller based on fruit fly optimization algorithm. In: *Proceedings of the international conference on mechatronics and automation*, Chengdu, China, 5–8 August 2012, pp.409–413. New York: IEEE.
11. Cui L, Li G and Lin Q. Adaptive differential evolution algorithm with novel mutation strategies in multiple sub-populations. *Comput Oper Res* 2015; 67: 155–173.
12. Zeng YR, Peng L and Zhang J. An effective hybrid differential evolution algorithm incorporating simulated annealing for joint replenishment and delivery problem with trade credit. *Int J Comput Int Syst* 2016; 9(6): 1001–1015.
13. Niu J, Zhong W, Liang Y, et al. Fruit fly optimization algorithm based on differential evolution and its application on gasification process operation optimization. *Knowl Bas Syst* 2015; 88(3): 253–263.
14. Shan D, Cao GH and Dong HJ. LGMS-FOA an improved fruit fly optimization algorithm for solving optimization problems. *Math Probl Eng* 2013; 2013(7): 1256–1271.
15. Zhang YW and Cui G. A novel multi-scale cooperative mutation fruit fly optimization algorithm. *Knowl Bas Syst* 2016; 114: 24–35.
16. Wu L, Xiao W, Zhang L, et al. An improved fruit fly optimization algorithm based on selecting evolutionary direction intelligently. *Int J Comput Intell Syst* 2016; 9(1): 80–90.
17. Wu L, Zuo C and Zhang H. A cloud model based fruit fly optimization algorithm. *Knowl Bas Syst* 2015; 89(3): 603–617.
18. Wang L, Liu R and Liu S. An effective and efficient fruit fly optimization algorithm with level probability policy and its applications. *Knowl Bas Syst* 2016; 97(3): 158–174.
19. Lv SX, Zeng YR and Wang L. An effective fruit fly optimization algorithm with hybrid information exchange and its applications. *Int J Mach Learn Cyb* 2017; 1–26, https://doi.org/10.1007/s13042-017-0669-5
20. Zhang YW, Cui G, Wang Y, et al. An optimization algorithm for service composition based on an improved FOA. *Tsinghua Sci Technol* 2015; 20(1): 90–99.
21. Gamble R. Introducing replaceability into web service composition. *IEEE T Serv Comput* 2014; 7(2): 198–209.
22. Zhang YW, Wu JT and Guo X. Optimising web service composition based on differential fruit fly optimisation algorithm. *Int J Comput Sci Math* 2016; 7(1): 87–101.
23. Deng SG, Yin JW, Yin JW, et al. Technical framework for Web service composition and its progress. *Comput Integr Manuf* 2011; 17(2): 404–412.
24. Ardagna D and Pernici B. Adaptive service composition in flexible processes. *IEEE T Software Eng* 2007; 33(6): 369–384.
25. Wen T, Sheng GJ, Guo Q, et al. Web service composition based on modified particle swarm optimization. *Chin J Comput* 2013; 36(5): 1031–1046.
26. Zhang YP, Jing Z, Zhang Y, et al. Dynamic web service composition based on discrete particle swarm optimization. *Computer Science* 2015; 42(6): 71–75.
27. Cardoso J, Sheth A, Miller J, et al. Quality of service for workflows and web service processes. *Web Sem* 2004; 1(3): 281–308.
28. Zhang T. QoS-aware web service selection based on particle swarm optimization. *J Netw* 2014; 9(3): 17–20.
29. Zhang YW, Wu JT, Zhao S, et al. Optimization service composition based on improved firework algorithm. *Comput Integr Manuf* 2016; 22(2): 422–432.
30. Yu Q, Chen L and Li B. Ant colony optimization applied to web service compositions in cloud computing. *Comput Electr Eng* 2015; 41(3): 18–27.