Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH **Open Access**

Locality sensitive hashing-aware fruit fly optimization algorithm and its application in edge server placement

Qian Cao[1], Bo Liu[1,2] and Ying Jin[3*]

Abstract

As is well known that the global optimization ability of the Fruit fly Optimization Algorithm (FOA)is weak because it is easy to fall into local optimum. In this paper, a Fruit Fly Optimization Algorithm based on Locality Sensitive Hashing-aware (LSHFOA)was proposed. The locality sensitive hashing mechanism to optimize the generation mechanism for swarm population individuals was used, which can improve the individual diversity of the population. Meanwhile, when the fruit fly population falls into the local optimum, the locality sensitive hashing mechanism was adopted to change the population location, which is used for jumping out of local optimal limits. To verify the performance of LSHFOA, it was compared with FOA and its improvement algorithms CFOA, and IFFO with 8 representative benchmark functions. A large number of experimental results showed that LSHFOA has a faster convergence speed and higher precision of optimization for function optimization, especially in high-dimensional multi-peak functions. In addition to the theoretical evaluation, we also evaluate its performance in a real-world scenario. Generally, an edge computing environment, as an extension of cloud computing, can allow the users to access the network in a low-latency manner. In this way, to capture the high-speed convergence advantage, this paper makes the first attempt to tackle a classic research problem in the edge computing environment, i.e., the edge server placement problem. The experimental results show that the new algorithm has an excellent application effect.

Keywords: Fruit fly optimization algorithm, Locality sensitive hashing, Swarm intelligence algorithm, Global optimization capacity, Edge server placement problem, Edge computing

Introduction

Edge computing [1–3], as an extension of cloud computing, has been widely-investigated to reduce the latency of its users [4, 5]. However, it raises lot of research problems, where edge server placement problem [6, 7] is one of the fundamental studies for the edge computing environment. To achieve the low-latency goal, the swarm intelligent algorithms, based on its high-convergence speed, could be an efficient way. Recently, the fruit fly optimization algorithm (FOA) is an intelligent optimization algorithm based on the foraging characteristics of fruit fly populations proposed by Wen chao Pan in 2012 [8]. This algorithm has been widely used in numerous scientific and engineering fields with its advantages of fast optimizing speed, simple operation, and few parameters, as well as the traditional intelligent optimization algorithms such as PSO [9], GA [10], and ACO [11]. For example, the literature [12] applied the FOA algorithm to the field of service computing firstly, and the algorithm was verified through a large number of experiments that proved it can be used to solve the service combination optimization problem and was recognized by peers [13]; the literature [14] proposed a multi-scale collaborative variation-based fruit fly optimization algorithm by improving the FOA algorithm and applied the algorithm to function optimization; the literature [15] introduced

*Correspondence: jiny@hfuu.edu.cn

[3] School of Artifical Intelligence and Big Data, Hefei University, Hefei, China
Full list of author information is available at the end of the article

the FOA algorithm into a neural network for parameter optimization and conducting a feasibility study for this scheme. In addition, FOA is also used in the field of edge computing. In [16], a cache placement algorithm with lower complexity is proposed by utilizing FOA.

However, the fruit fly optimization algorithm suffers from the tendency to fall into local optima, the inability to traverse the problem domain and poor stability and so on [12, 14]. Literature [17] proposed that the original fruit fly optimization algorithm cannot access the negative problem domain, i.e., the position of an individual fruit fly cannot reach the negative domain, and therefore, it cannot be applied to some problems. For example, it cannot optimize for functions with a negative definition domain. In the literature [14], it is concluded from the theoretical analysis that the result of FOA depended on the initial position of the population and the initial position of the population is random, i.e., the stability of FOA was poor and the global optimization capacity was weak. Therefore, in order to solve these shortcomings of FOA, the research for the FOA algorithm is divided into two main aspects. On the one hand, it is necessary to reduce the dependence of the fruit fly optimization algorithm on the initial position of the population so as to improve the performance of this algorithm. On the other hand, dynamically adjusting the radius of population generation is useful to escape from the local optimum limitation during the fruit fly population. Among them, the literature [17] proposed a linear fruit fly optimization algorithm LGMS-FOA by improving the FOA population generation method to enhance the global optimization capacity of FOA. However, this method generates population individuals in an overly simple way and also suffers from the shortcomings such as weak global optimization capacity. To further remedy this drawback, a chaotic fruit fly optimization algorithm is proposed in literature [18]. The stability and optimization capacity of the algorithm are verified through extensive function tests based on the linear generation of population individuals. In addition, a fruit fly optimization algorithm with adaptive population size is proposed in [19] to solve the function optimization problem. In summary, although the fruit fly optimization algorithm has been widely used in many aspects, such as communication resources allocation and scheduling, computation offloading and caching schemes in mobile edge networks, it still has non-negligible shortcomings. In order to solve these shortcomings, some improvement algorithms have been widely studied in recent years. Although these improvement algorithms could solve the shortcomings of FOA to a certain extent, each improvement algorithm led to new shortcomings at the same time, such as the weak generalizability due to more parameters. In addition, many of the improved algorithms tend to focus on solving a certain class of problems, while neglecting other aspects of performance. For example, traditional FOA tends to be more suitable for optimization problems with positive problem domains and extreme value points close to the origin of the problem [8, 9].

In order to solve the weakness of the global optimization capacity of the fruit fly optimization algorithm from other aspects and to improve the solution accuracy of the fruit fly optimization algorithm in multi-polarity problems, this paper proposed a fruit fly optimization algorithm based on locality sensitive hashing, so that the algorithm can be better applied in distributed environment such as edge computing. The algorithm further improved the ability of fruit fly optimization algorithm to traverse the problem domain by introducing a locality sensitive hashing mechanism, establishing a locality sensitive hashing table of fruit fly population positions, and then using the table to reduce the candidate points of similar population positions. Thus, it can reduce the dependence of fruit fly optimization algorithm on the initial population positions and improve the global optimization capacity of FOA. In order to verify the performance of the algorithm in this paper, eight benchmark functions, covering single-dimensional and multi-dimensional, single-peak and multi-peak aspects, are selected for detailed comparison with the classical FOA improvement algorithms CFOA [18], IFFO [19].

This study detailed the implementation process of the traditional fruit fly optimization algorithm and the background knowledge of locality sensitive hashing, Locality sensitive hashing system and proposes algorithm were introduced and were proved through a large number of experiments to be efficient.

In the real-world scenario, edge computing is proposed to extend the cloud computing to overcome its high-latency difficulty issue [20–22]. Unline some AI techniques [3, 21, 23], to achieve the low-latency objective, the high-speed convergence of LSHFOA is most suitable for the edge computing. To evaluate its performance in the real-world scenario, a representative research problem in an edge computing environment has been investigated, i.e., edge server placement (ESP) problem [6, 7]. The edge server placement problem is a fundamental study for the edge computing environment. It is vital because without placing the edge servers properly, the edge computing environment will suffer from various kinds of challenges [24–26], such as network failures, huge latency, etc. Thus, in this paper, we first propose an improved Fruit fly optimization algorithm, namely LSHFOA. Then, we apply this new approach to solve the edge server placement (ESP) problem in the edge computing environment.

The rest of this paper is organized as follows. Section 2 provided the background information of this manuscript. Section 3 proposed the models of LSHESP and LSHFOA-ESP and its pseudo-code. Section 4 described the experimental implementation in detail and the results is discussed in Section 5. Final, Section 6 concluded the whole manuscript and pointed out the future directions.

Related Background

The key notations used in this paper are summarized in Table 1.

Fruit fly optimization algorithm

FOA, as a population intelligence optimization method, works by describing the problem on an n-dimensional space as each fruit fly, and the position of the fruit fly represents a feasible solution to the problem. Furthermore, the current position of each fruit fly is measured by the fitness function, and the population position is changed by the individual's fitness. Then, a new population of fruit flies is generated. In this way, the fruit fly population gradually approaches the optimal solution to the problem. The specific implementation process is shown in Fig. 1.

The detailed optimizing process is as follows: Step 1: Initial population location

$$\begin{cases} x_{axis} = rand(LR) \\ y_{axis} = rand(LR) \end{cases} \tag{1}$$

Step 2: Generating individuals of the population

$$\begin{cases} x_i = x_{axis} + rand(V) \\ y_i = y_{axis} + rand(V) \end{cases} \tag{2}$$

Step 3: Calculate the individual fitness of the population

Table 1 Key Notations

Notation	Description
x_{axis}	the *x*-coordinate of the group location
y_{axis}	the *y*-coordinate of the group location
x_i	the *x*-coordinate of the *i*th fruit fly
y_i	the *y*-coordinate of the *i*th fruit fly
V	the radius of the fruit fly population
$Dist_i$	the Euclidean distance between *i*th fruit fly and the optimal solution
$Smell_i$	the Fitness of *i*th fruit fly to the optimal solution
S	a set of edge servers
s_i	edge server $s_i, i \in \{1, ..., n\}$
A	accessibility matrix between edge servers and the users
u_i	user $u_i, i \in \{1, ..., n\}$
U	set of users
$a_{j,i}$	accessibility between u_i and s_j

$$\begin{cases} Dist_i = \sqrt{x_i^2 + y_i^2} \\ s_i = \frac{1}{Dist_i} \\ Smell_i = Fitness(S_i) \end{cases} \tag{3}$$

Step 4: Preservation of optimal Drosophila individuals

$$\begin{cases} [bestSmellbeseIndex] = \max(Smell_i) \\ Smellbest = \max(bestSmell, Smellbest) \end{cases} \tag{4}$$

Step 5: Generate new population locations

$$\begin{cases} x_{axis} = x_{bestIndex} \\ y_{axis} = y_{bestIndex} \end{cases} \tag{5}$$

Step 6: Repeat Step 2 - Step 5 until the iteration conditions or accuracy requirements are met.

Where LR is the solving range, V is the population range radius, $Dist_i$, S_i, $Fitness$ are the formulas for the fitness function, $[bestSmellbestindex] = max?(Smell_i)$ which presents the individual optimal fitness value and individual position in the current population, $Smellbest$ is the global optimal fitness value, x_axis is the x coordinate of the fruit fly population location, y_axis is the y coordinate of the fruit fly population location.

Locality sensitive hashing

Hashing is an efficient method of data retrieval, mapping data to a hash table through a hash function can achieve a shift in search time from $O(n)$ to $O(1)$. Among them, locality sensitive hashing can further map high-dimensional massive data to approximate nearest neighbors to a locality sensitive hashing table. Then, it can quickly find similar data. There are basic ideas of locality sensitive hashing. One is that two adjacent data in high-dimensional data space will have a high probability to remain adjacent after being mapped to low-dimensional data space. The other is that two non-adjacent data will also have a high probability to be low-dimensional space. With this mapping, we can find the adjacent data points in the low-dimensional data space and avoid high-dimensional data space finding, which would be time-consuming. A hash mapping with such a property is said to be locality sensitive. Take Fig. 2 as an example, there are four data blocks $D1$, $D2$, $D3$ and $D4$, where $D1$ and $D2$ are similar or close to each other. The four data blocks can be mapped into a locality sensitive hashing table by locality sensitive hashing. The mapping usually results in D1 and D2 being in the same or similar region, and then the gap between $D3$ and $D4$ is further widened.

From Fig. 2, it can be seen that similar data are mapped to similar regions through locality sensitive hashing operations. Further, through the analysis of FOA, it is easy to know that FOA has a strong dependence on the initial

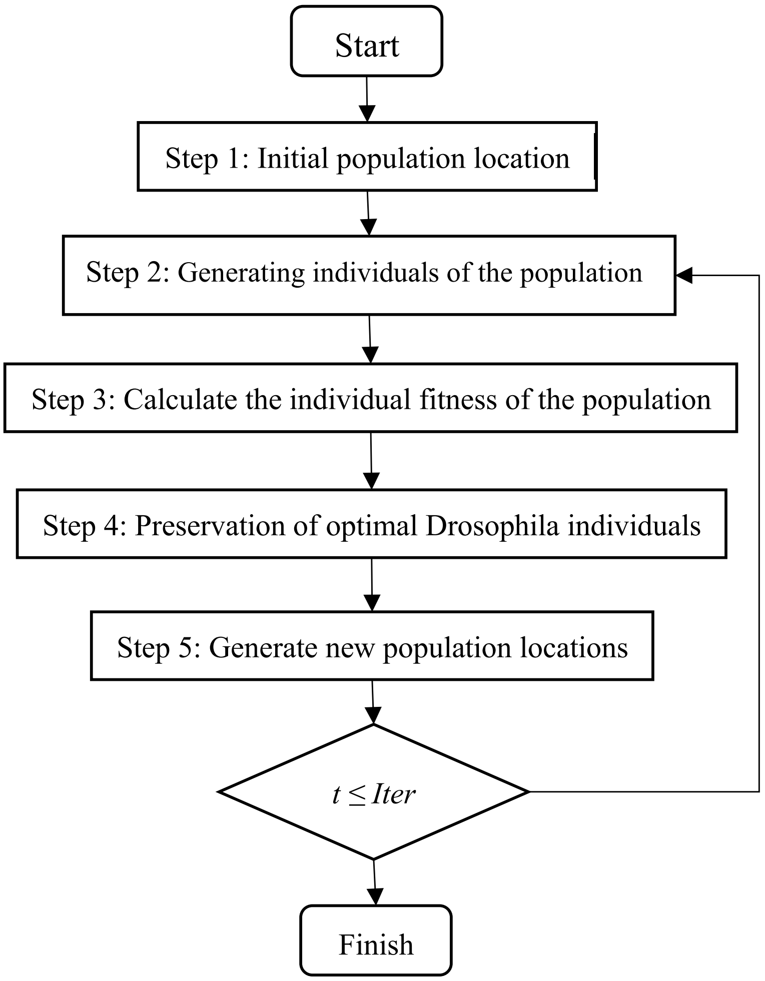
Start
Step 1: Initial population location
Step 2: Generating individuals of the population
Step 3: Calculate the individual fitness of the population
Step 4: Preservation of optimal Drosophila individuals
Step 5: Generate new population locations
$t \leq Iter$
Finish

Fig. 1 FOA flow chart

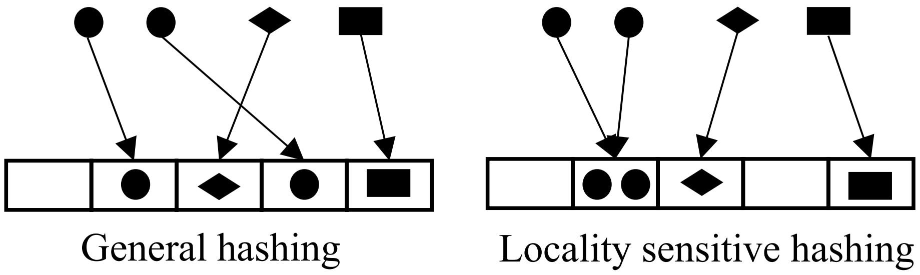

Fig. 2 Schematic diagram of locality sensitive hashing

position of the population [12]. By adjusting the initial location of the populations, it is beneficial to enhancing the global optimization capacity of FOA. At the same time, increasing the randomness of population positions or decreasing the similarity between population positions, can further enhance the global optimization capacity of FOA. The locality sensitive hashing operation of a set of population positions can obtain similar population positions quickly and effectively. Then the population positions with larger gaps can be more conveniently selected to enhance the global optimization capacity of FOA.

Edge Server Placement

For the ESP problem, edge servers are usually deployed on the base stations or access points. Therefore, in each ESP scenario, it usually includes n base stations $B = \{b_1, ..., b_n\}$, and m edge servers $S = \{s_1, ..., s_m\}$. The ESP problem aims to place these m edge servers on those n base stations to serve its most users $U = \{u_1, ..., u_c\}$. For each user $u_k \in U$, it can access a set of base stations, which is denoted by $a_{k,i}$, where $a_{k,i} = 1$ indicates the user u_k can access base station b_i, otherwise, $a_{k,i} = 0$. Thus, the user-base station accessibilities can be modelled as a matrix A.

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \tag{6}$$

Then, if an edge server has been placed on the base station b_i, denoted by $p_i = 1$, all its users can be served, i.e., $\forall u_k \in U, a_{k,i} = 1$. Thus, the objective of the ESP problem is to maximize the maximum number of served users,

$$O = \max(\sum_{u_k \in U} \min(\sum_{b_i \in B} a_{k,i} \cdot p_i, 1)) \tag{7}$$

Where $\sum_{b_i \in B} a_{k,i} \cdot p_i$ is used to calculate the served times of u_k by all edge servers. Then, $\min(\sum_{b_i \in B} a_{k,i} \cdot p_i, 1) = 0$ means that u_k cannot be served by any edge servers, otherwise, i.e., $\min(\sum_{b_i \in B} a_{k,i} \cdot p_i, 1) = 1$ indicates that u_k can be served.

Algorithm

Locality sensitive hashing mechanism based FOA

As seen from Section 1.2, the main idea of locality sensitive hashing is that if two data blocks in a high- dimensional space are close, the result of a locality sensitive hashing of that data block has a high probability of being similar. If two data blocks are farther apart, the hash result will have a small probability to be the same. Therefore, for the FOA algorithm, its core is to design the locality sensitive hashing function $h(x)$.

Given an n-dimensional problem $F(x_1, \cdots, x_n)$ with problem domain $x \in [R_min, R_max]^n$, given an existing solution as $X' = (x'_1, \cdots, x'_n)$, and any other solution is $X = (x_1, \cdots, x_n)$, then the problem $F(x_1, \cdots, x_n)$ relative to the solution X' the locality sensitive hashing function of is

$$h(X|X') = [\frac{\sqrt{(x_1 - x'_1)^2 + \cdots + (x_n - x'_n)^2}}{E_n}] \tag{8}$$

where $[value]$ denotes the integer part of the value, E_n denotes the solution overhead of the problem $F(x_1, \cdots, x_n)$, and the physical meaning is the range of each hashtable element. In short, the larger the value, the smaller the number of elements of the corresponding hash table. Meanwhile, the smaller the value, the more elements of the corresponding hash table. E_n is calculated as follows.

$$E_n = \frac{\sqrt{(R^1_{max} - R^1_{min})^2 + \cdots + (R^n_{max} - R^n_{min})^2}}{\omega} \tag{9}$$

Where ω denotes the accuracy of the solved problem. It is used to control the range of each element of the hash table. For example, if given a two-dimensional ($n = 2$) problem $F(x_1, x_2)$, its problem domain is $x \in [0,5]^2$, $x_i \in [0,5]$, $i = 1,2$, then the solution range of the problem $F(x_1, x_2)$ is $(0,0), \cdots, (5,5)$, if given the parameter $\omega = 5$, then $E_n = \sqrt{50}/5$, i.e., the range of elements in each sensitive hash table is $[[(\sqrt{50}/5) * (j-1)], [(\sqrt{50}/5) * j]]$, where j denotes the jth elements in the sensitive hash table, and if given the parameter $\omega = 10$, then $E_n = \sqrt{50}/10$, similarly, the range of elements in each sensitive hash table is $[[(\sqrt{50}/10) * j], [(\sqrt{50}/10) * (j+1)]]$. Then, assuming that the current solution (i.e., for the population location in the FOA) is $X = (0,0)$ and the target solution (i.e., the population location to be selected in the FOA) is $X_1 = (0,1)$, $X_2 = (1,0)$, $X_3 = (1,1)$, $X_4 = (0,2)$, $X_5 = (2,0)$, $X6 = (2,2)$, then the sensitive hash value corresponding to the target solution($\omega = 5$) is $h(X_1|X) = [5/\sqrt{50}] = 0$, $h(X_2|X) = [5/\sqrt{50}] = 0$, $h(X_3|X) = [10/\sqrt{50}] = 1$, $h(X_4|X) = [20/\sqrt{50}] = 2$, $h(X_5|X) = [20/\sqrt{50}] = 2$, $h(X_6|X) = [40/\sqrt{50}] = 5$, Thus, the target solutions X_1 and X_2 are mapped into the first element of the locality sensitive hashing table, X_3 is mapped into the second element of the locality sensitive hashing table, the target solutions X_4 and X_5 are mapped into the third element of the locality sensitive hashing table, and the target solution X_6 is mapped into the sixth element of the locality sensitive hashing table. The graphical representation is shown in the following figure.

From Fig. 3, the locality sensitive hashing result of the target solution (the population location to be selected in the FOA) in this example is divided into four sensitive hashing table elements, i.e., elements $j \in J = \{1,2,3,6\}$. Then, how to migrate the FOA population location is another key problem. In this study, the roulette technique for the target solution selection and migrates the selected target solution were used as the new FOA population location.

Roulette as a commonly used selection method is also known as the proportional selection method. The basic idea of Roulette is that the probability of each population location point being selected is related to its corresponding edge weight value, which is performed in the following steps.

Step1: First determine the selection probability of each locally sensitive hash table element.

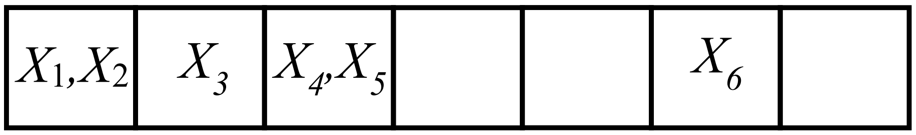

Fig. 3 Locality sensitive hashing table

$$\alpha_j = e^j, j \in J \tag{10}$$

Step2: Calculate the sum of the weights between all FOA population location points to be selected and the current fruit fly population location $X_i : \sum_{j\in J} \alpha_j$

Step 3: Calculate the probability of each FOA population location point to be selected.

$$p_k = \frac{\alpha_k}{\sum_{j\in J} \alpha_j} \tag{11}$$

Step 4: Calculate the cumulative probability of each FOA population location point to be selected.

$$p_k' = \frac{\sum_{j=1}^{k} p_j}{\sum_{j\in J} \alpha_j} \tag{12}$$

Step 5: Generate a random number θ with uniform distribution in the interval [0, 1].

Step5: If $\theta \le p_k'$ and $p_(k-1)' \le \theta$, then element k of the locality sensitive hashing table is selected, and then, any FOA to be selected population location X_i in element k is randomly selected.

At this point, the new FOA population location has been selected.

Algorithm 1 LSHFOA

```
Require: benchmark function and definition $D$
Ensure: the optimal solution in the definition domain $D$
 1: Initialize a set of $(n)$ population location points $V = \{X_1, X_2, \ldots, X_n\}$
 2: Select a population location point as the initial: $\forall X_i \in V$
 3: for each $t \le iter$ do
 4:     Generate population individuals $(Pop)$ for point $X_i$ according to the population
 5:     Calculate the benchmark function value for each individual fruit fly
 6:     Preservation of optimal fruit fly individuals
 7:     if trapped in local optimum then
 8:         Calculate the probability of selecting the location point of the population to be selected in
            FOA and the point $X_i$: $\{\alpha_1, ..., \alpha_n\}$
 9:         Roulette calculates the selection probability of each population location point: $P =
            \{p_1, \cdots, p_n\}$
10:         Randomly generated selection probabilities $p_0$
11:         Selecting new population locations based on locality sensitive hashing table
12:     end if
13:     Update population location point $X_i$
14: end for
15: return the optimal solution
```

By means of roulette, the weight relationship between population location points can be mapped to the selection probability, and then random values are used for population location selection. This roulette selection method can, on the one hand, improve the selection probability of dominant population location points, i.e., satisfy the principle of optimization selection; on the other hand, roulette also has a chance to select other slightly inferior population location points, and this operation helps to enhance the diversity of fruit fly population locations and further ensures the global optimization capacity of the fruit fly optimization algorithm.

LSHFOA

In the improved fruit fly optimization algorithm (LSHFOA) proposed in this paper, locality sensitive hashing tables are used for the selection of Drosophila population locations, i.e., when the fruit fly optimization algorithm falls into a local optimum (usually measured by multiple population location invariance), the roulette mechanism is used for population location selection (according to locality sensitive hashing table). the pseudo-code representation of LSHFOA is shown in Algorithm 1 [9, 12, 13].

Algorithm 1 shows that the time complexity of the algorithm is $O(m_1 m_2)$, i.e., the time consumption of the algorithm is related to the population size m_1 and the number of iterations m_2. Therefore, in order to reduce the time-consuming of the algorithm, the population size in the experiments of this paper is 50 and the number of iterations is 300. In order to cover more initial location points, the size of the initial population location set V is 50, i.e., the fruit fly population location locality sensitive hashing table contains a total of 50 points [12, 15]. According to the execution flow of Algorithm 1, the flow chart of LSHFOA is shown in Fig. 4.

Where V in Fig. 4 denotes a set of fruit fly population location points, i.e., each vertex in the locality sensitive hashing table of fruit fly population location, X_i denotes any point of population location points in V, Pop denotes the set of population individuals generated according to the population location X_i with a scale of 50, and Fitness denotes the fitness value of each fruit fly individual on the benchmark function, i.e., the Step6- Step7 denotes the new population location selection scheme, i.e., the locality sensitive hashing table model with roulette wheel selection method [10].

LSHFOA-ESP

In order to solve the ESP problem [6, 7] by LSHFOA, we employ a new perspective to model the ESP problem.

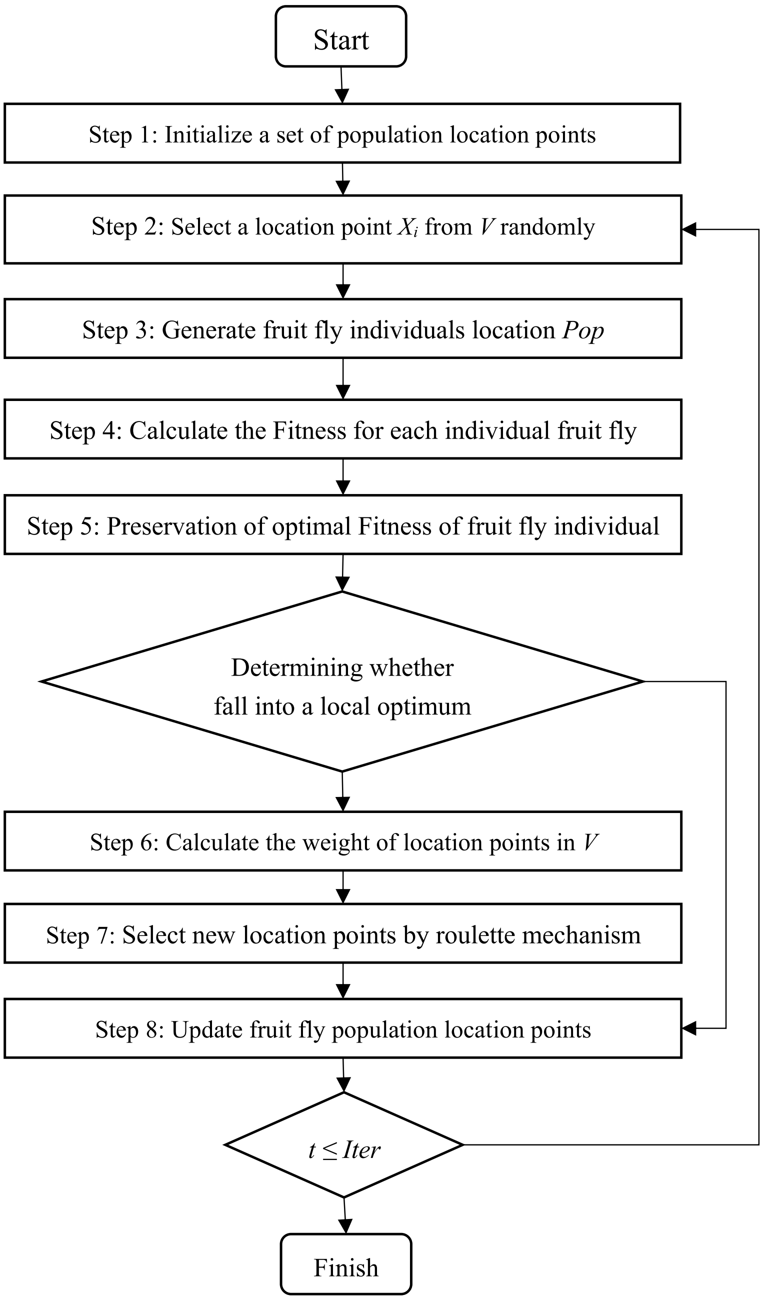
Start
Step 1: Initialize a set of population location points
Step 2: Select a location point X_i from V randomly
Step 3: Generate fruit fly individuals location *Pop*
Step 4: Calculate the Fitness for each individual fruit fly
Step 5: Preservation of optimal Fitness of fruit fly individual
Determining whether fall into a local optimum
Step 6: Calculate the weight of location points in V
Step 7: Select new location points by roulette mechanism
Step 8: Update fruit fly population location points
$t \leq Iter$
Finish

Fig. 4 LSHFOA flow chart

Definition 1 Coding of Fruit Flies: Given a set of base stations B with a set of users and a set of edge server S, the coding scheme of each individual fruit fly dl is a tuple with m elements, denoted as $d_l = [d_l^1, \cdots, d_l^m]$, where $d_l^m \in 0 \bigcup i|b_i \in B$, i.e., d_l^j is s_j's placement decision with a value in $0 \bigcup i|b_i \in B$.

Figure 5 provides an example of such a coding scheme for a fruit fly d_l.

As shown in Fig. 5, each fruit fly is coded by m elements, which indicate m edge servers. And, the value of each element varies among the base stations, i.e., $1, \cdots, n$. In this way, each fruit fly represents one feasible solution to the edge server placement problem. Then, each fruit fly changes based on the schemes of LSHFOA. In terms of the fitness function of the ESP problem, it can be modelled as below to pursue the objective of the ESP problem, as shown in Eq. (13).

$$Fit(d_l) = \sum_{u_k \in U} \min\left(\sum_{d_l^j \in d_l} a_{k,d_l^j}, 1\right) \tag{13}$$

Experimental analysis

In order to verify the performance of the proposed algorithm in this paper for optimization, this section conducts a comprehensive comparison with the improved algorithms of FOA, CFOA and IFFO, in eight commonly used benchmark functions. Firstly, this section lists the experimental environment and parameter settings of this paper; secondly, the eight benchmark functions are analyzed and demonstrated in detail; finally, a graphical presentation and detailed analysis are made based on the experimental results [12, 14].

Experimental environment and parameter settings

The experiments in this paper are based on Windows 10, 64-bit operating system, 16G memory, 2.4GHZ desktop computer, the experimental programming language is C#, the compiler is Visual Studio 2010. where the population size is 50, the number of iterations iter is 300, the initialized population location points are 50, each experiment is repeated 50 times, and the mean value is taken as the experimental result and plotted as the experimental performance graph. The other parameters set in the experiment are shown in Table 2.

Benchmark functions

In order to analyze the performance of the algorithm more comprehensively, the eight benchmark functions are divided into the following categories: single-dimensional single-peak function (*F*1), single-dimensional multi-peak function (*F*2, *F*3), multi-dimensional single-peak function (*F*4, *F*5), multi-dimensional multi-peak function (*F*6, *F*7) and two-dimensional combined

Table 2 Parameter Settings

Algorithm	Parameters	Numerical values	Meaning
IFFO	V	1	population radius
CFOA	$\cos i \cos x_i^{-1}$	Chebyshev	Chaotic map function
LSHFOA	ω	10	Question accuracy

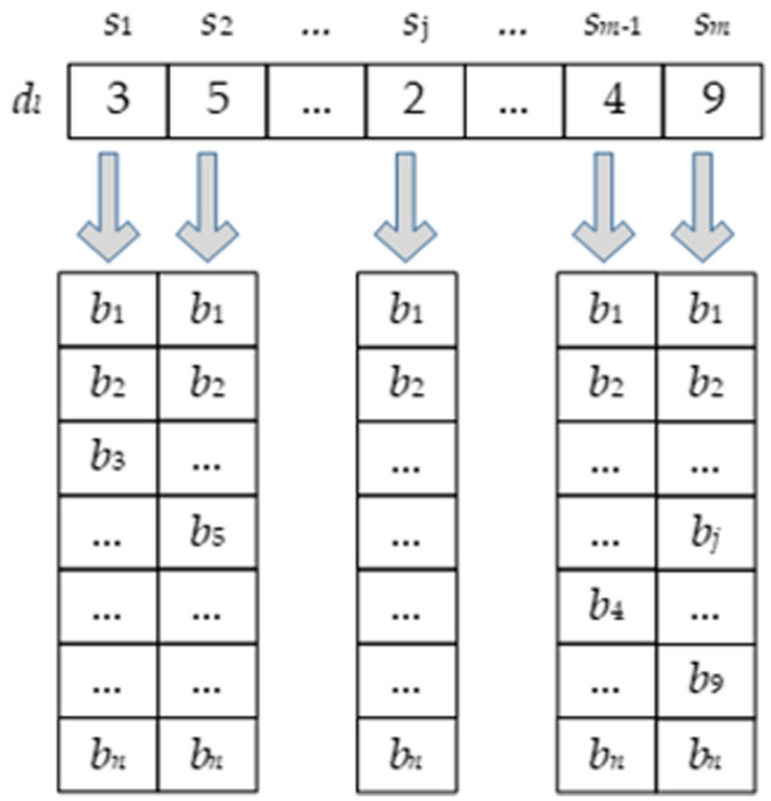

Fig. 5 Example of the coding of individual fruit fly

function ($F8$). At the same time, the selected benchmark functions have both 0 ($F1$, $F4$, $F5$, $F6$, $F7$, $F8$) and non-0 ($F2$, $F3$, $F8$) extreme points in order to verify the global optimization capability of the algorithm in a more comprehensive way. The detailed benchmark functions are shown in Fig. 6. The dimension n indicates that the benchmark function has n variables, and is denoted as $x_1, \cdots, x_n$, and the definition domain $[-10, 10]^n$ indicate that each dimension in the benchmark function takes values in the range [-10,10], and the minimum value indicates the minimum value of the function in the current definition domain.

Result and Discussion

LSHFOA

In order to comprehensively analyze the experimental performance of the algorithm LSHFOA in this paper, comparison tests with IFFO and CFOA on the basis of Fig. 6 are performed, and the experimental results are shown in the following figures.

Figure 7 represents the experimental results of the single-dimensional single-peak function F1. Since the single-dimensional single-peak function is relatively simple, a set of classical functions was used randomly to test the performance of the algorithm. As can be seen from the figure, the algorithm in this paper is significantly better than IFFO in both the optimization accuracy and optimization efficiency. Compared to CFOA, the optimization efficiency is slightly lower while the final experimental results are similar. The experimental results of this group show that the algorithm of this paper also has good experimental results under the one-dimensional single-peak function test.

Figures 8 and 9 represent benchmark functions F2 and F3, i.e., single-dimensional multi-peak function tests. The experimental accuracy and efficiency of LSHFOA are much better than CFOA and slightly better than IFFO. While the experimental accuracy and efficiency of this algorithm are significantly better than CFOA and IFFO. Meanwhile, it can be seen from the changing trend of each algorithm in Figs. 7 and 8 that the fruit fly optimization algorithm has a strong dependence on the initial position of the population, i.e., after the initial position is determined, the algorithm's optimization results are limited. However, it can be slightly seen from Fig. 9 that the algorithm in this paper can reduce the influence of the initial position on the FOA optimization results, i.e., the optimization results will fluctuate slightly which will suitable to find better results. Such fluctuations are more obvious in Figs. 11 and 12, and the impact of such fluctuations on the experimental accuracy is similar.

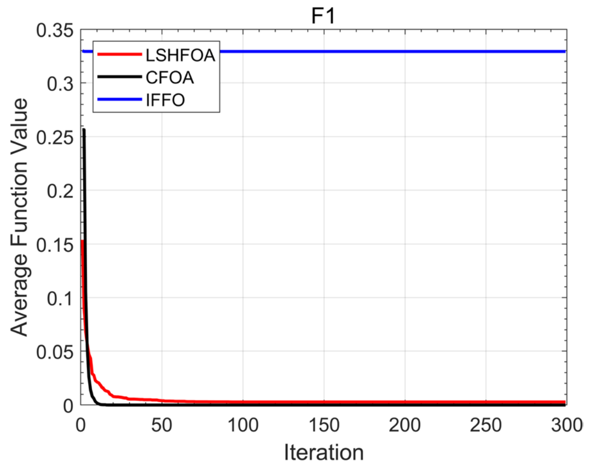

Fig. 7 Performance of the algorithm under F1 function

Function	Dimensionality	Definition Domain	Minimun Value
$F1 = \lvert x\rvert$	$n = 1$	$-10 \le x \le 10$	0
$F2 = sin(x)$	$n = 1$	$-10 \le x \le 10$	-1
$F3 = xsin(x)$	$n = 1$	$0 \le x \le 20$	-17.3076
$F4 = \sum_{i=1}^{n} x_i^2$	$n = 30$	$[-10,10]^n$	0
$F5 = \sum_{i=1}^{n} \lvert x_i\rvert + \prod_{i=1}^{n} \lvert x_i\rvert$	$n = 30$	$[-10,10]^n$	0
$F6 = \sum_{i=1}^{n}(x_i^2 - 10cos2\pi x_i + 10)$	$n = 30$	$[-5.12, 5.12]^n$	0
$F7 = -20\exp\left(-0.2\sqrt[2]{\frac{\sum_{i=1}^{n} x_i^2}{n}}\right) - \exp\left(\frac{\sum_{i=1}^{n} cos2\pi x_i}{n}\right) + 20 + e$	$n = 30$	$[-32,32]^n$	0
$F8 = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(2 - \frac{1}{8\pi}\right) - cosx_1 + 10$	$n = 2$	$[-1,1]^2$	-2

Fig. 6 Benchmark functions

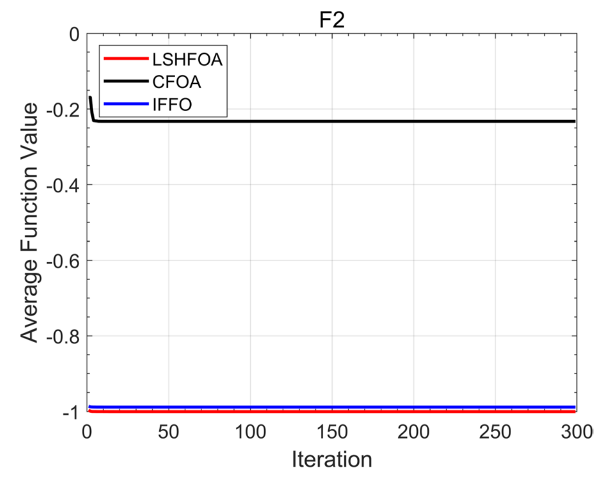

Fig. 8 Performance of the algorithm under F2 function

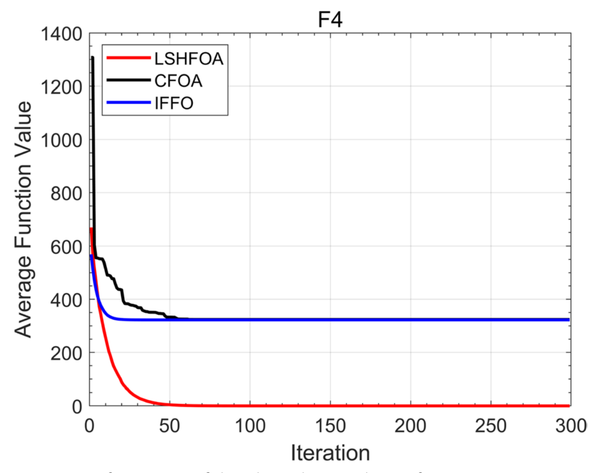

Fig. 10 Performance of the algorithm under F4 function

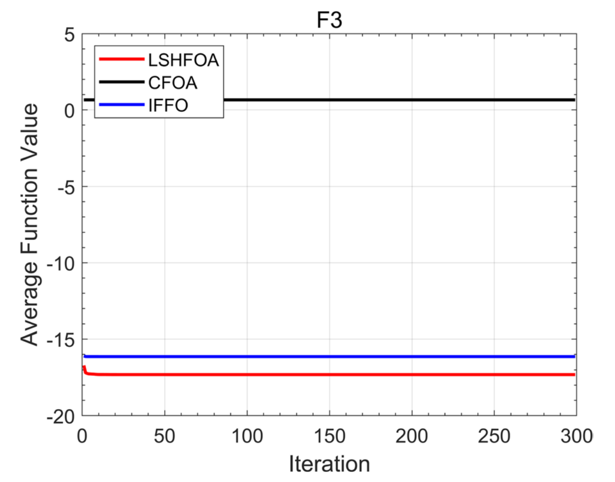

Fig. 9 Performance of the algorithm under F3 function

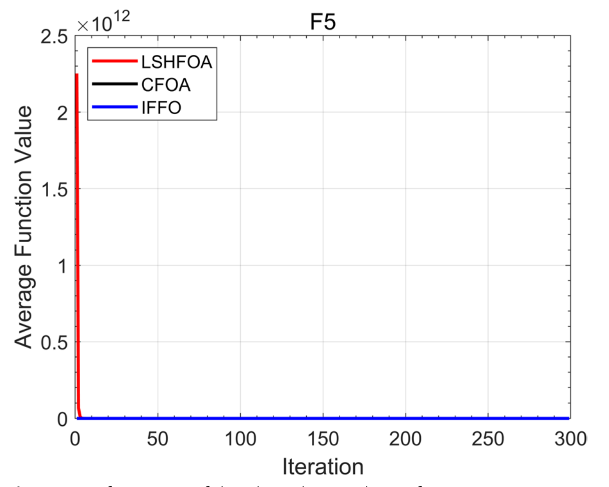

Fig. 11 Performance of the algorithm under F5 function

Figures 10 and 11 represent the multidimensional multi-peak test functions (F4 and F5). from Fig. 10, it can be seen that with the increase of iterations, the algorithm in this paper can rapidly reduce the experimental search accuracy. The benchmark function value rapidly decreases and the results are significantly smaller than CFOA and IFFO. Therefore, for the benchmark function F4, the algorithm in this paper has better experimental results with convergence speed. From Fig. 10, it can be seen that in the multi-dimensional multi-peak test function, although the initial Fitness value is larger than LSHFOA, the algorithm in this paper can quickly approach the optimal solution through the local sensitive hashing mechanism, so as to achieve similar optimization-seeking accuracy and

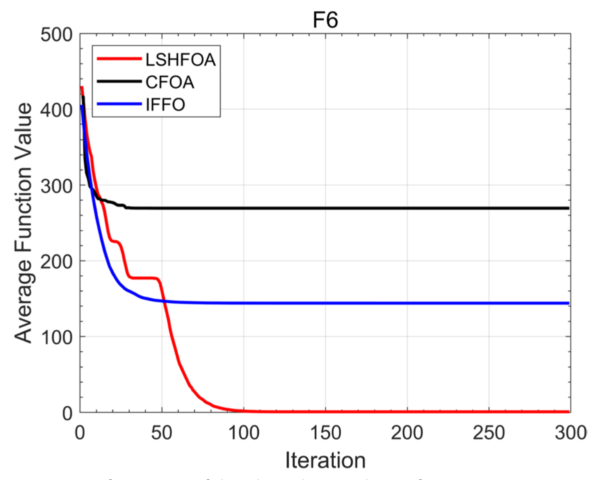

Fig. 12 Performance of the algorithm under F6 function

optimization-seeking efficiency comparing to algorithms IFFO and CFOA. In summary, it can be seen from Figs. 10 and 11 that with the increase of iterations, the algorithm in this paper has excellent optimization-seeking accuracy and convergence speed in the multidimensional single-peak function test.

Figures 12 and 13 represent the multi-dimensional multi-peak test functions (test functions F6 and F7). Overall, it can be seen from the experimental results in Figs. 12 and 13 that the proposed algorithm in this paper can achieve significant advantages in the multi-dimensional multi-peak situation compared with the CFOA and IFFO. For example, as can be seen from Fig. 11, even though the initial population position LSHFOA is slightly worse than that of CFOA and IFFO, with the increase in the number of iterations, the optimization accuracy of this algorithm is gradually improved. After 50 iterations, the experimental results of this algorithm significantly outperform the comparative algorithms CFOA and IFFO. Therefore, for the benchmark function F5, the accuracy and efficiency of the experimental results are significantly better than the classical algorithms CFOA and IFFO, although the initial position of the algorithm is slightly worse. As can be seen from Fig. 13, for benchmark function F7, the algorithm in this paper outperforms CFOA and IFFO in terms of optimization results close to 0 (the most value of the function in the domain). In terms of the optimization efficiency, the accuracy of the feasible solution of this paper is higher than that of the comparison algorithm in about 20 iterations, which indicates that LSHFOA can achieve excellent optimization results and efficiency in high-dimensional multi-peak functions. In summary, for the multi-dimensional multi-peak problem, the algorithm in this paper can get better results. It can be seen that LSHFOA is more suitable for solving the high-dimensional multi-peak optimization problem.

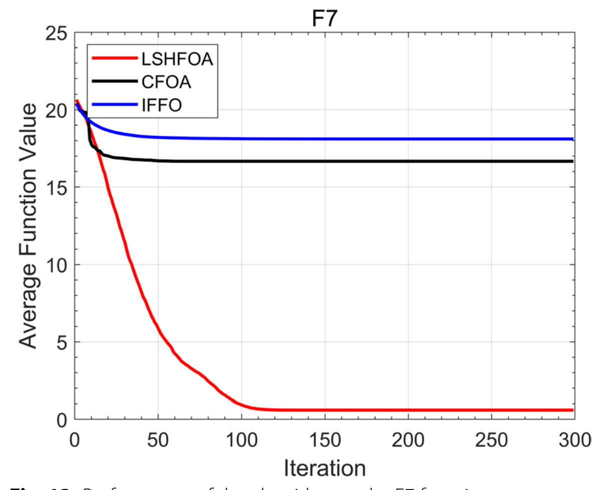

Fig. 13 Performance of the algorithm under F7 function

Figure 13 represents the combination function of two-dimensional variables, and it can be seen from the figure that although the algorithm in this paper has a slightly worse optimization accuracy than IFFO, it significantly outperforms CFOA, that is LSHFOA can be applied in such problems. Combining the above several test functions, it can be seen that the algorithm in this paper can achieve significantly better experimental results than CFOA and IFFO in multi-peak situations, especially in high-dimensional multi-peak situations.

LSHFOA-ESP

To extensively evaluate LSHFOA-ESP's performance, we simulate a set of ESP scenarios in the experiments. We employ a Windows machine equipped with an Intel Core i7-7500 processor, and 16G RAM to perform the experiments. At the same time, a real-world dataset is applied to conduct the experiments. It has been widely used in edge computing environments [2, 6, 22, 26, 27]. Overall, this dataset includes a large number of real-world users and base stations in Melbourne Metropolis, Australia, including the geographical information of users and base stations, and the coverage of base stations.

Performance Metrics. Two metrics are employed to measure the effectiveness and efficiency of LSHFOA-ESP, including 1) the number of served users, and 2) the time consumption.

Comparison Approaches. In this paper, to evaluate the performance comprehensively, two state-of-the-art approaches and one baseline approach are employed as comparison approaches in this paper.

RESP [6]: This is a representative approach proposed very recently. It makes the first attempt to solve the robustness-oriented edge server placement problem, with the aim to maximize the overall robustness.

CRESP [7]: This approach is an extension of, which focuses on the tradeoff between robustness and coverage. This is because maximizing the overall robustness only usually leads to a decrease in user coverage.

FOA-ESP: This is a baseline approach that tries to solve the edge server placement problem by using the classical FOA only [12, 14].

Parameter Settings. Similar to many studies in edge computing environment [2, 6, 7, 22, 26–29], in each experiment, n base stations are randomly selected from the dataset and c users are selected from the data set [2, 22, 27] randomly as well, where base stations include the geographical locations and the coverage radiuses, users include the geographical locations. Then, based on those geographical locations of base stations and users and the radiuses of base stations, the user-base station accessibilities matrix can be built. Next, to test the performance of LSHFOA-ESP comprehensively, three parameters are varied, including 1) number of base stations (n); 2) number of edge servers (m) and 3) number of users (c). Accordingly, the experimental settings of those parameters are summarized in Table 3. LSHFOA-ESP iterates 300 times before giving out the solution. The number of fruit flies in each iteration is 50. Each time we vary one parameter and repeat the experiment 100 times, then the results are averaged.

Effectiveness Generally, Figs. 15, 16 and 17 show the effectiveness, measured by the number of served users, of all the approaches in Set 1, Set 2 and Set 3, respectively. From those figures, it is easy to see that the proposed approach, LSHFOA-ESP can serve the most users compared to other approaches. First, LSHFOA-ESP can find a solution to cover the most users, which is significantly greater than the classic FOA and its application to the ESP problem. This is because, as stated above, LSHFOA, as an extension of FOA, is designed to overcome the difficulties of FOA and aims to find the optimal solution. Thus, LSHFOA-ESP's performance is better than FOA-ESP's, by 15.32%. Second, RESP serves the least number of users. The background reason is straightforward. That is, RESP is designed to maximize the overall robustness of edge servers, i.e., maximizing the overall served times of users instead of serving more users. Thus, edge servers are usually driven to be placed on a small group of base stations that have covered the greatest number of users. In this way, the overall robustness will be maximized. Obviously, LSHFOA-ESP outperforms RESP significantly, by 25.48%. Lastly, as an extension of RESP, CRESP is designed to balance the overall robustness and the number of served users. As a result, its number of served users achieves the second-highest performance. But it is still lower than LSHFOA-ESP by 8.32%.

Specifically, Fig. 15 shows that when the number of base stations increases in Set 1, the number of served users achieved by all the four approaches decreases. The background reason is analyzed as follows. As shown in Table 3, when the number of base stations varies, the number of edge servers and the number of users are fixed. In this case, a larger number of base stations will lead to a decrease in the number of users covered by each base station on averagely. As a consequence, selecting the same number of base stations, i.e., placing a fixed number of edge servers, usually results in a lower number of served users. But the results, as shown in Fig. 14, are gradually stabilized. This is because, the locations of base stations and users come from a real-world, and they are fixed. In this case, when nearly all the base stations have been selected, the geographical distributions of users are unchanged. Thus, the number of served users decreases first and then becomes stabilized. Figure 16 demonstrates that LSHFOA-ESP is capable of serving the most edge users when the number of edge servers varies. Compared to FOA-ESP, RESP and CRESP, LSHFOA-ESP outperforms them with significant advantages. Especially, when more and more edge servers are placed in a specific edge computing environment, the performance gaps between LSHFOA-ESP and FOA-ESP, RESP and CRESP increase gradually. This is because, given a fixed

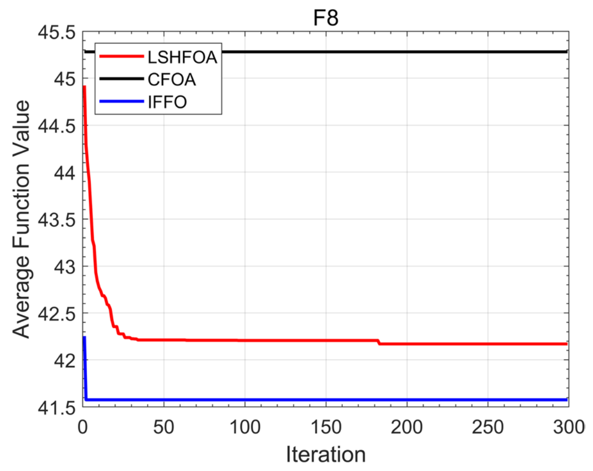

Fig. 14 Performance of the algorithm under F8 function

Table 3 Experimental Settings

	n	*m*	*c*
Set 1	100, 200, ..., 800	40	4000
Set 2	400	10, 20, ..., 80	4000
Set 3	400	40	1000, 2000, ...,8000

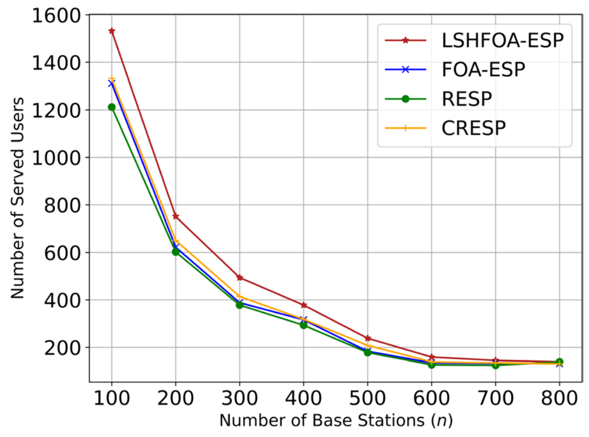

Fig. 15 Number of Served Users (Set 1)

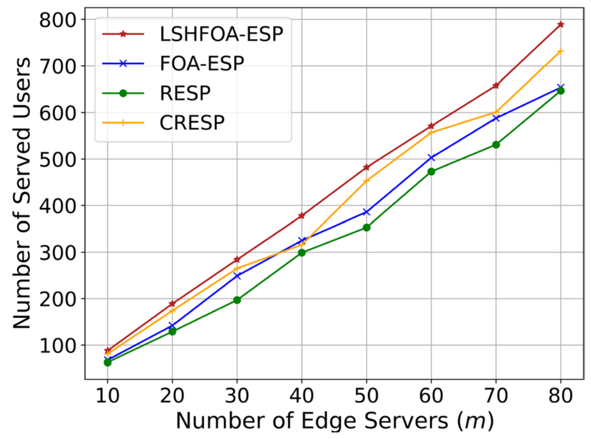

Fig. 16 Number of Served Users (Set 2)

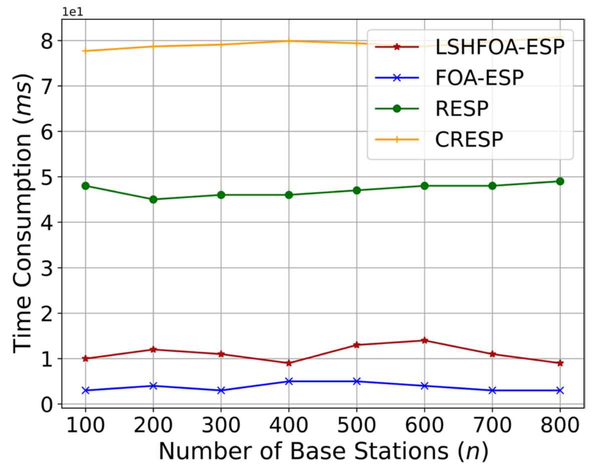

Fig. 18 Overall Time Consumption (Set 1)

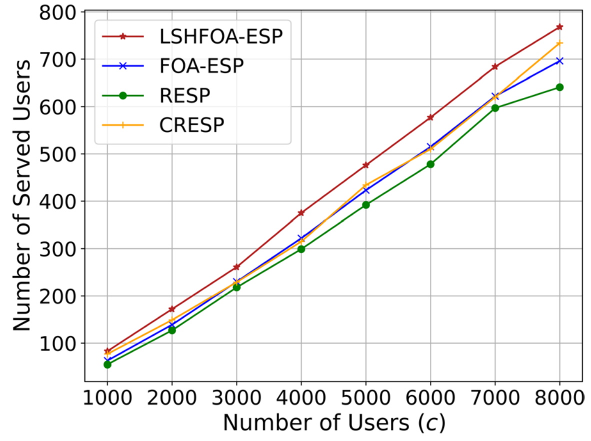

Fig. 17 Number of Served Users (Set 3)

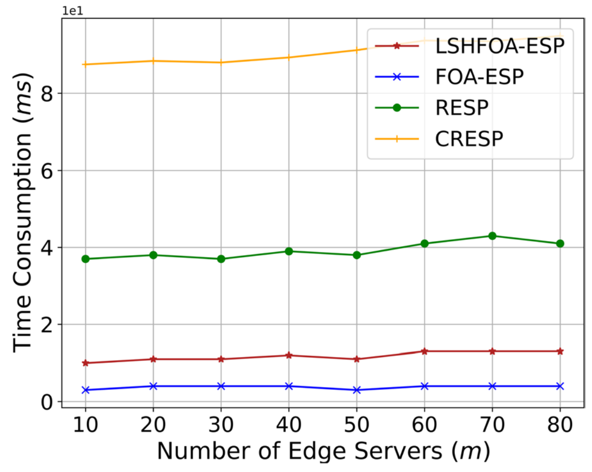

Fig. 19 Overall Time Consumption (Set 2)

number of base stations, placing more edge servers will cover more base stations to serve more users. When the number of users increases in Set 3, the number of served users increases in all approaches, as shown in Fig. 17 The underlay reason is similar to that in Set 2. That is, more users are extracted from the real-world data, and each base station will cover more users in general. Thus, placing a fixed number of edge servers usually leads to an increase in the overall number of served users, as shown in Fig. 16. As shown in Fig. 16, our approach, LSHFOA-ESP can still find a solution to serve the maximum number of users. Therefore, as demonstrated in Figs. 15, 16 and 17, the proposed approach, LSHFOA-ESP can be used to solve the edge server placement effectively.

Efficiency Figures 18, 19 and 20 demonstrate the time consumption of all approaches in Set 1, Set 2 and Set 3. In general, we can find that LSHFOA-ESP takes much less time than RESP and CRESP to find the solutions, and is slightly higher than FOA-ESP. Those phenomena are acceptable. First, as an improved FOA approach, LSHFOA-ESP takes a slightly higher time to find a solution. It is straightforward. But as shown in Figs. 15, 16 and 17, LSHFOA-ESP can serve much more users than FOA-ESP. Second, LSHFOA-ESP takes a smaller time to find a better solution than RESP and CRESP, as shown in Figs. 15, 16 and 17 and Figs. 18, 19 and 20. This shows that LSHFOA-ESP can be used to solve the edge server placement problem efficiently. Last, in terms of the time consumption of RESP and CRESP, CRESP is an extension of RESP by considering more metrics, such as robustness and user coverage. Thus, CRESP takes more time than RESP, obviously, and achieves a better result than RESP, as well, as shown in Figs. 15, 16 and 17.

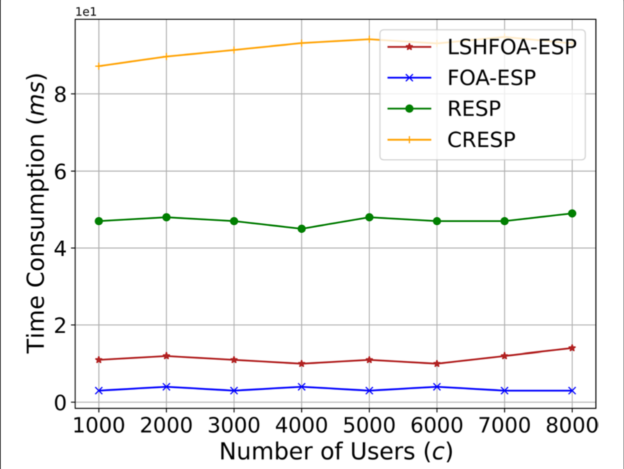

Fig. 20 Overall Time Consumption (Set 3)

Specifically, as shown in Figs. 18, 19 and 20, we can find that LSHFOA-ESP's time consumption increases as long as the number of edge servers or the number of users increases. The reason is straightforward - a larger number of edge servers means a longer coding of each fruit fly, as shown in Fig. 5, and a larger number of users means the more complicated fitness function calculations of each fruit fly. However, in Set 1, the increase in the number of base stations does not significantly affect LSHFOA-ESP's time consumption, as shown in Fig. 18. This is because, for the coding of each fruit fly, i.e., Fig. 5, each code element will select one of the base stations only without traversing the entire base stations. Thus, the time consumption of Set 1 fluctuate as the number of base stations varies. As shown in Figs. 18, 19 and 20, the increase in time consumption usually follows a linear trend when the number of edge servers and the number of users increase. This indicates the LSHFOA-ESP can handle the large-scale ESP problem efficiently, i.e., LSHFOA-ESP can converge quickly in large-scale ESP scenarios.

Conclusion

In this paper, through the study of the fruit fly optimization algorithm (FOA), the optimization results of the FOA depend on the initial position highly, which leads to the reduction of the global optimization capacity of the fruit fly optimization algorithm. To further optimize the algorithm, this paper introduces a locality sensitive hashing mechanism to get rid of the influence of the initial position of the fruit fly population on the optimization result by constructing a locality sensitive hashing table and making the selection of the fruit fly population position when the FOA falls into a local optimum according to the roulette approach. To verify the performance of the algorithm LSHFOA proposed in this paper, a comparative study is performed with eight classical benchmark functions (covering single-dimensional and multi-dimensional, single-peak and multi-peak characteristics) and the improved algorithms CFOA, IFFO, and MSFOA of FOA. The experimental results show that the algorithm in this paper has better convergence speed and better optimization accuracy in the multi-dimensional multi-peak case compared with the comparison algorithm.

Although the algorithm in this paper can obtain high experimental results in multi-polar situations, there are some problems that can be further optimized. For example, when falling into local optimum the process of measurement of the FOA, the judgement of the position of the fruit fly population proposed in this study remains unchanged many times. The judgement tends to improve the accuracy and efficiency of the fruit fly optimization algorithm in finding the best result. Therefore, in the future study, it is necessary to improve the judgement for the problem of falling into local optimum, so as to further improve the performance of this algorithm, and apply this algorithm to the edge computing environment.

Acknowledgements
The authors are grateful to the Faculty of Innovation Engineering at Macau University of Science and Technology, and the School of Artificial Intelligence and Big Data at Hefei University.

Authors' contributions
All authors take part in the discussion of the work described in this paper. Qian Cao designed all the experiments, Qian Cao and Bo Liu wrote the main manuscript text, and Ying Jin prepared all the figures and tables. All authors reviewed the manuscript. The author(s) read and approved the final manuscript.

Funding
This work is supported by the Key Project of Nature Science Research for Universities of Anhui Province of China (No. KJ2020A0657), 2020 Quality Improvement Project of Chaohu University on Discipline Construction (No. kj20xqyx03) and The Provincial Natural Science Research Program of Higher Education Institutions of Anhui province (No. KJ2021A1030).

Availability of data and materials
Not applicable.

Declarations

Ethics approval and consent to participate
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]College of Information Engineering, Chaohu University, Hefei, China. [2]Faculty of Innovation Engineering, Macau University of Science and Technology, Macao, China. [3]School of Artifical Intelligence and Big Data, Hefei University, Hefei, China.

Received: 11 June 2022 Accepted: 17 August 2022
Published online: 30 August 2022

References

1. Chen Y, Gu W, Li K Dynamic task offloading for Internet of Things in mobile edge computing via deep reinforcement learning. Int J Commun Syst. https://doi.org/10.1002/dac.5154
2. Cui G, He Q, Xia X, Chen F, Gu T, Jin H et al (2021) Demand response in NOMA-based mobile edge computing: a two-phase game-theoretical approach. IEEE Trans Mob Comput. https://doi.org/10.1109/TMC.2021.3108581
3. Chen Y, Zhao F, Lu Y, Chen X Dynamic task offloading for mobile edge computing with hybrid energy supply. Tsinghua Sci Technol. https://doi.org/10.26599/TST.2021.9010050
4. Chen Y, Liu Z, Zhang Y et al (2021) Deep reinforcement learning-based dynamic resource management for mobile edge computing in industrial internet of things. IEEE Trans Ind Inform 17(7):4925–4934
5. Huang J, Lv B, Wu Y et al (2022) Dynamic Admission Control and Resource Allocation for Mobile Edge Computing Enabled Small Cell Network. IEEE Trans Veh Technol 71(2):1964–1973
6. Cui G, He Q, Xia X, Chen F, Jin H, Yang Y (2020) Robustness-oriented k edge server placement. In: 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID). IEEE, Melbourne, VIC, Australia, May 11, pp 81–90
7. Cui G, He Q, Chen F, Jin H, Yang Y (2020) Trading off between user coverage and network robustness for edge server placement. IEEE Trans Cloud Comput. https://doi.org/10.1109/TCC.2020.3008440
8. Pan WT (2012) A new Fruit Fly Optimization Algorithm: Taking the financial distress model as an example. Knowl Based Syst 26:69–74
9. Zhou J, Yang J, Lin L, Zhu Z, Ji Z (2018) Local Best Particle Swarm Optimization Using Crown Jewel Defense Strategy. Critical developments and applications of swarm intelligence. IGI Global, 27-52
10. wei Gong D, Sun J, Miao Z (2018) A Set-Based Genetic Algorithm for Interval Many-Objective Optimization Problems. IEEE Trans Evol Comput 22:47–60
11. Dorigo M, Stützle T (2019) Ant Colony Optimization: Overview and Recent Advances. Handb Metaheuristics. Springer, 311-351
12. Zhang Y, Cui G, Wang Y, Guo X, Zhao S (2015) An optimization algorithm for service composition based on an improved FOA. Tsinghua Sci Technol 20:90–99
13. Jatoth C, Gangadharan GR, Buyya R (2017) Computational Intelligence Based QoS-Aware Web Service Composition: A Systematic Literature Review. IEEE Trans Serv Comput 10:475–492
14. Zhang Y, Cui G, Wu J, Pan WT, He Q (2016) A novel multi-scale cooperative mutation Fruit Fly Optimization Algorithm. Knowl Based Syst 114:24–35
15. Zhang Q, Li C, Yin C, Zhang H, Su F (2022) A Hybrid Framework Model Based on Wavelet Neural Network with Improved Fruit Fly Optimization Algorithm for Traffic Flow Prediction. Symmetry 14(7):1333
16. Timo DH, Andy DP, Jarmo T, Stamatis V (2005) Embedded Computer Systems: Architectures, Modeling, and Simulation 5th International Workshop, SAMOS 2005, Samos, Greece, July 18-20, 2005, proceedings. In: SAMOS. Springer Science \& Business Media
17. Shan D, Cao G, Dong H (2013) LGMS-FOA: An Improved Fruit Fly Optimization Algorithm for Solving Optimization Problems. Math Probl Eng 2013:1–9
18. Mitic M, Vukovic N, Petrovic M, Miljković Z (2015) Chaotic fruit fly optimization algorithm. Knowl Based Syst 89:446–458
19. ke Pan Q, Sang H, Duan JH, Gao L (2014) An improved fruit fly optimization algorithm for continuous function optimization problems. Knowl Based Syst 62:69–83
20. Xu J, Li D, Gu W, Chen Y (2022) UAV-assisted task offloading for IoT in smart buildings and environment via deep reinforcement learning. Build Environ. Elsevier, 109218
21. Huang J, Tong Z, Feng Z (2022) Geographical POI recommendation for Internet of Things: A federated learning approach using matrix factorization. Int J Commun Syst. https://doi.org/10.1002/dac.5161
22. Cui G, He Q, Chen F, Zhang Y, Jin H, Yang Y (2021) Interference-aware game-theoretic device allocation for mobile edge computing. IEEE Trans Mob Comput. https://doi.org/10.1109/TMC.2021.3064063
23. Chen Y, Liu Z, Zhang Y, Wu Y, Chen X, Zhao L (2021) Deep Reinforcement Learning-Based Dynamic Resource Management for Mobile Edge Computing in Industrial Internet of Things. IEEE Trans Ind Inform 17:4925–4934
24. Chen Y, Zhao F, Chen X, Wu Y (2022) Efficient Multi-Vehicle Task Offloading for Mobile Edge Computing in 6G Networks. IEEE Trans Veh Technol 71(5):4584–4595
25. Ying C, Hua X, Zhuo M, et al (2022) Cost-Efficient Edge Caching for NOMA-enabled IoT Services. China Commun
26. Zhang Y, Pan J, Qi L, He Q (2021) Privacy-preserving quality prediction for edge-based IoT services. Future Gener Comput Syst 114:336–348
27. Cui G, He Q, Xia X, Chen F, Dong F, Jin H et al (2021) Ol-eua: Online user allocation for noma-based mobile edge computing. IEEE Trans Mob Comput. https://doi.org/10.1109/TMC.2021.3112941
28. Chen Y, Zhao F, Chen X, Wu Y (2022) Efficient Multi-Vehicle Task Offloading for Mobile Edge Computing in 6G Networks. IEEE Trans Veh Technol 71:4584–4595
29. Huang J, Lv B, Wu Y, Chen Y, Shen XS (2022) Dynamic Admission Control and Resource Allocation for Mobile Edge Computing Enabled Small Cell Network. IEEE Trans Veh Technol 71:1964–1973

Publisher's Note