An Adaptive Bacterial Foraging Optimization Algorithm with Lifecycle and Social Learning

Xiaohui Yan[1,2*], Yunlong Zhu[1], Hao zhang[1,2], Hanning Chen[1], Ben Niu[3]

[1]*Key Laboratory of Industrial Informatics, Shenyang Institute of Automation, Chinese Academy of Sciences, 110016, Shenyang, China*

[2]*Graduate School of the Chinese Academy of Sciences, 100039, Beijing, China*

[3]*College of Management, Shenzhen University, 518060, Shenzhen, China*

* *yanxiaohui@sia.cn*

Abstract: Bacterial Foraging Algorithm (BFO) is a recently proposed swarm intelligence algorithm inspired by the foraging and chemotactic phenomenon of bacteria. However, its optimization ability is not so good compared with other classic algorithms as it has several shortages. This paper presents an improved BFO Algorithm. In the new algorithm, a lifecycle model of bacteria is founded. The bacteria could split, die or migrate dynamically in the foraging processes and population size varies as the algorithm runs. Social learning is also introduced so that the bacteria will tumble towards better directions in the chemotactic steps. Besides, adaptive step lengths are employed in chemotaxis. The new algorithm is named BFOLS and it is tested on a set of benchmark functions with dimensions of 2 and 20. Canonical BFO, PSO and GA algorithms are employed for comparison. Experiment results and statistic analysis show that the BFOLS algorithm offers significant improvements than original BFO algorithm. Especially with dimension of 20, it has the best performances among the four algorithms.

Keywords: Bacterial Foraging Optimization; Lifecycle model; Social learning; Adaptive search strategies

1. INTRODUCTION

Swarm intelligence is an innovative optimization technique inspired by the social behaviors of animal swarms in nature. Though the individuals have only simple behaviors and are without centralized control, complex collective intelligence could emerge on the level of swarm by their interaction and cooperation. Recent years, several swarm intelligence algorithms have been proposed, such as Ant Colony Optimization (ACO) [1], Particle Swarm Optimization (PSO) [2], Artificial Bee Colony (ABC) [3] and Bacterial Foraging Optimization (BFO) [4] et al. BFO algorithm is first proposed by Passino in 2002. It is inspired by the foraging and chemotactic behaviors of bacteria, especially the Escherichia coli (*E. coil*). By smooth running and tumbling, The *E. coil* can move to the nutrient area and escape from poison area in the environment. The chemotactic is the most attractive behavior of bacteria. It has been studied by many researchers [5] [6]. By simulating the problem as the foraging environment, BFO algorithm and its variants are used for many numerical optimization [7] [8] or engineering optimization problems, such as distributed optimization [10], job shop scheduling [11], image processing[12] and stock market prediction [13] et al.

However, the original BFO has some shortages: (1) Dispersal, reproduction and elimination happens each after a certain amount of chemotaxis operations. The appropriate time and method for dispersal and reproduction are important. Otherwise, the stability of the population may be destroyed. (2) The tumble angles in the chemotactic phase are generated randomly. As a result, the algorithm is more like a random searching algorithm except it will try further in better directions. The bacteria swarm lacks of interaction between individuals. Good information carried by those individuals in higher nutritional areas can't be shared with and utilized by other bacteria. (3) The swim step length in the original BFO algorithm is a constant. In most cases, the bacterium will run one more step if the position is better than its last position. If the swim step is large at the end stage (for example, larger than the distance between its current position and the optimal point), it will skip the optimal point repeatedly. This will make the bacteria hard to converge to the optimal point.

In this paper, several adaptive strategies are used to improve the original BFO algorithm. First, a lifecycle model of bacteria is proposed. Bacteria will split or die depending on the nutrition obtained in their foraging processes. Then, social leaning is introduced to enhance the information sharing between bacteria. The tumble angles are no longer generated randomly but directed by the swarm's memory. Last, adaptive search strategy is employed, which makes the bacteria could use different search step lengths in different situations.

The rest of the paper is organized as followed. In Section 2, we will introduce the original BFO algorithm. Its features and pseudo code are given. In section 3, the adaptive strategies of BFOLS algorithm is described in detail. In Section 4, the BFOLS algorithm is tested on a set of benchmark functions compared with several other algorithms. Results are presented and discussed. The test of its new parameters setting and the simulation of its varying population size are also done in this section. Finally, conclusions are drawn in Section 5.

2. ORIGINAL BACTERIAL FORAGING OPTIMIZATION

The *E. coli* bacterium is one of the earliest bacterium which has been researched. It has a plasma membrane, cell wall, and capsule that contains the cytoplasm and nucleoid. Besides, it has several flagella which are randomly distributed around its cell wall. The flagella rotate in the same direction at about100-200 revolutions per second [14]. If the flagella rotate clockwise, they will pull on the cell to make a "tumble". And if they rotate counterclockwise, their effects accumulate by forming a bundle which makes the bacterium "run" in one direction [4]. As shown in Figure 1.

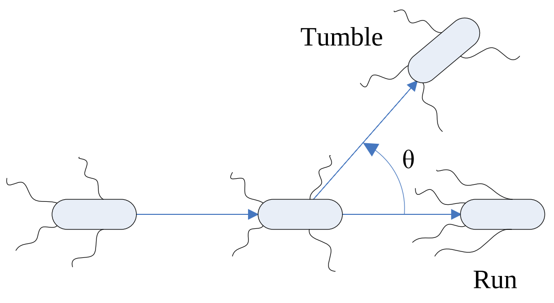


Figure 1: Chemotactic behavior of E. coli: Run and Tumble

The bacteria can sense the nutrient concentration in the environment. By tumbling and running, the bacteria will search for nutrient area and keep away from the poisonous area. Simulating the foraging process of bacteria, Passino proposed the Bacterial Foraging Optimization (BFO) algorithm. The main mechanisms of BFO are illustrated as followed.

2.1 Chemotaxis

Chemotaxis is the main motivation of the bacteria's foraging process [15]. It is consist of a tumble with several runs. In BFO, the position updating which simulates the chemotaxis procedure is used the Eq. (1) as followed. θ_i^t presents the position of the *i*th bacterium in the *t*th chemotaxis step. $C(i)$ is the step length during the *i*th chemotaxis. $\phi(i)$ is a unit vector which stands for the swimming direction after a tumble. It can be generated by Eq. (2), where Δ_i is a randomly produced vector with the same dimension of the problem.

$$\theta_i^{t+1} = \theta_i^t + C(i)\phi(i) \tag{1}$$

$$\phi(i) = \frac{\Delta_i}{\sqrt{\Delta_i^T \Delta_i}} \tag{2}$$

In each chemotactic step, the bacterium generated a tumble direction firstly. Then the bacterium moves in the direction using Eq. (1). If the nutrient concentration in the new position is higher than the last position, it will run one more step in the same direction. This procedure continues until the nutrient get worse or the maximum run step is reached. The maximum run step is controlled by a parameter called N_s.

2.2 Reproduction

For every N_c times of chemotactic steps, a reproduction step is taken in the bacteria population. The bacteria are sorted in descending order by their nutrient obtained in the previous chemotactic processes. Bacteria in the first half of the population are regarded as having obtained sufficient nutrients so that they will reproduce. Each of them splits into two (duplicate one copy in the same location). Bacteria in the residual half of the population die and they are removed out from the population. The population size remains the same after this procedure. Reproduction is the simulation of the natural reproduction phenomenon. By this operator, individuals with higher nutrient are survived and duplicated, which guarantees that the potential optimal areas are searched more carefully.

2.3 Eliminate and dispersal

In nature, the changes of environment where population lives may affect the behaviors of the population. For example, the sudden change of temperature or nutrient concentration, the flow of water, all these may cause bacteria in the population die or move to another place [16]. To simulate this phenomenon, eliminate-dispersal is added in the BFO algorithm. After every N_{re} times of reproduction steps, an eliminate-dispersal event happens. For each bacterium, a random number is generated between 0 and 1. If the random number is less than a pre-determined parameter, known as P_e, the bacterium will be eliminated and a new bacterium is generated in the environment. The operator can be also regarded as moving the bacterium to a randomly produced position. The eliminate-dispersal events may destroy the chemotactic progress. But they may also promote the solutions since dispersal might place the bacteria in better positions. Overall, contrary to the reproduction, this operator enhances the diversity of the algorithm.

In BFO algorithm, the eliminate-dispersal events happen for N_{ed} times. That is to say, there are three loops for the bacteria population in BFO algorithm. The outer loop is elimination-dispersal event, the middle loop is reproduction event and the inner loop is chemotactic event. The algorithm ends after all the three loops are finished. The pseudo code of original BFO algorithm is given in Figure 2.

```
Initialization
For $i$=1:$N_{ed}$
  For $j$=1: $N_{re}$
    For $k$=1:$N_c$
      For $n$=1:$S$
        $Jlast$=$J(n)$
        Generate a tumble angle for bacterium n;
        Update the position of bacterium $n$ by Eq. (1);
        Recalculate the $J(n)$
        $m$=0
        While ( $m$<$N_s$)
          If $J(n)$<$Jlast$
            $Jlast$=$J(n)$;
            Run one more step using Eq. (1);
            Recalculate the $J(n)$;
            $m$=$m$+1;
          Else
            $m$=$N_s$;
          End if
        End while
      End for
      Update the best value achieved so far;
    End for
    Sort the population according to $J$;
    For $m$=1:$S$/2
      Bacterium($k$+$S$/2)= Bacterium($k$);
    End For
  End for
  For $l$=1:$S$
    If (rand<$P_e$)
      Move Bacterium $l$ to a random position
    End if
  End for
End for
```

Figure 2: Pseudo code of original BFO algorithm

3. BACTERIAL FORAGING OPTIMIZATION WITH LIFECYCLE AND SOCIAL LEARNING

To improve the optimization ability of BFO algorithms, much virants are .pro. In the proposed BFOLS algorithm, three strategies are used to improve the original BFO:

3.1 Lifecycle model of bacterium

As mentioned above, in original BFO algorithm, there are three loops for the population. Bacteria will reproduce after N_c times of chemotactic steps and dispersal after N_{re} times of reproduction. As a result, the parameter setting of N_c and N_{re} are important to the performance of the algorithm. Unsuitable parameter values may destroy the chemotactic searching progress [17]. To avoid this, we remove the three loops In BFOLS algorithm. Instead, for each bacterium, we will decide it to reproduce, die or migrate by certain conditions in the bacteria's cycle.

The idea of lifecycle has been used in some swarm intelligence algorithms [18] [19]. Based on Niu's model in [19], a new lifecycle model of bacteria is founded in this paper. In the model, a bacterium could be represented by a six-tuple as followed:

$$B = \{P,\ F,\ N,\ T,\ D,\ C\} \tag{3}$$

P, *F*, *N*, *T*, *D*, *C* represent the position, fitness, nutrient, state, tumble direction, step length respectively. It should be noted that fitness is the evaluation to the current position of a bacterium, and nutrient is total nutrient gained by the bacterium in its whole searching process.

We define the nutrient updating formula as Eq. (4). $Flast$ represents the fitness of the bacterium's last position (For a minimum problem, fitness is larger when the function value is smaller). In initialization stage, nutrients of all bacteria are zero. In the bacterium's chemotactic processes, if the new position is better than the last one, it is regarded that the bacterium will gain nutrient from the environment and the nutrient is added by one. Otherwise, it loses nutrient in the searching process and its nutrient is minus by one.

$$N(i) = \begin{cases} N(i)+1 & if\,(F(i) > Flast) \\ N(i)-1 & if\,(F(i) < Flast) \end{cases} \tag{4}$$

There are five states defined in the lifecycle model: *born*, *forage*, *split*, *die* and *migrate*. That is, *T*= {*Born*| *Forage*| *Split* | *Die* | *Migrate*}. The bacteria are born when they are initialized. Then they will forage for nutrient. In the foraging process, if a bacterium obtains sufficient nutrient, it will split into two in the same position; if the bacterium enters bad area and loses nutrient to a certain threshold, it will die and be eliminated from the population; if the bacterium is with a bad nutrient value but hasn't died yet, it may migrate to a random position according to certain probability. After split or migrate, the bacterium is regarded as new born and its nutrient will be reset to 0. The state transition diagram is shown in Figure 3.

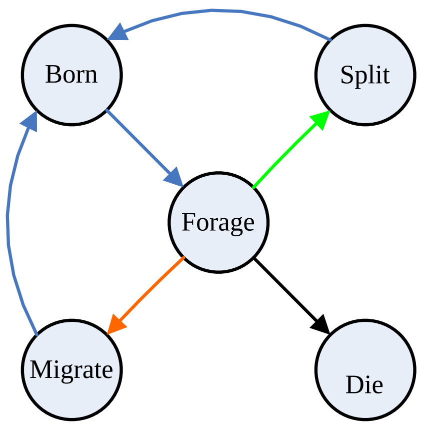


Figure 3: State transition in lifecycle model of bacteria In BFOLS

The split criterion and dead criterion are listed in Formula (5) and (6). N_{split} and N_{adapt} are two new parameters used to control and adjust the split criterion and dead criterion. *S* is the initial population size and S^i is the current population size. It should be noticed that the population size will increase by one if a bacterium splits and reduce by one if a bacterium dies. As a result, the population size may vary in the searching process. At the beginning of the algorithm, as S equals to S^i, the bacterium will split when its nutrient is larger than N_{split}, and die when its nutrient is smaller than 0. We don't need to worry that the population will decrease suddenly at the first beginning because when it is first time for the bacteria to decide whether to die or not, they have passed through the chemotactic process so most of the bacteria's nutrient values are larger than zero. With the algorithm runs, the population size may change and differ from the initial population size. On certain conditions, the population size will reduce to zero, which makes the algorithm unable to continue. Oppositely, if the population size becomes too large, it will cost too much computation and hard to evolve. To avoid the population size becoming too large or too small, a self-adaptive strategy is introduced: If S^i is larger than S, for each N_{adapt} of their differences, the split threshold value will increase by one. And if S is larger than S^i, for each N_{adapt} of their differences, the death threshold value will decrease by one. The strategy is also in accord with nature. Behaviors of organisms will be affected by the environment their lived. If the population is too crowded, the competition between the individuals will increase and death becomes common. If the population is small, the individuals are easier to survival and reproduce. By this strategy, the split threshold is enhanced when population size is large and the dead condition is stricter when population is small, which controls the population size in a relatively stable range.

$$Nutrient(i) > \max(N_{split}, N_{split} + (S^i - S) / N_{adapt}) \tag{5}$$

$$Nutrient(i) < \min(0, \quad 0 + (S^i - S) / N_{adapt}) \tag{6}$$

When the nutrient of a bacterium is less than zero, but it hasn't died yet, it could migrate with a probability. A random number is generated and if the number is less than migration probability P_e, it will migrate and move to a randomly produced position. Nutrient of the bacterium will be reset to zero.

It should be mentioned that the splitting, death and migration operators are judged in sequence, if one of them is done, the algorithm will breakout from the current cycle and won't execute the rest of judgments.

3.2 Social Learning

Social learning is the core motivation in the formation of the collective knowledge of swarm intelligence [20]. For example, in PSO algorithm, particles learn from the best particles and themselves [21]; In ABC algorithm, bees learn from their neighbors [22]. However, the social learning is seldom used in original BFO algorithm. In chemotactic steps of original BFO, the tumble directions are generated randomly. Information carried by the bacteria in rich-nutrient positions is not utilized. In our BFOLS, we assume that all bacteria can memory the best position they have reached and share the information to other bacteria. And in chemotactic steps, a bacterium will decide which direction to tumble using the information of its personal best position and the population's global best position. Based on the assumption, the tumble directions in our BFOLS are generated using Eq. (7). Where θ_{gbest} is the global best of the population found so far and $\theta_{i,pbest}$ is the ith bacterium's personal historical best. The tumble direction is then normalized as unit vector by Eq. (2) and the position updating is still using the Eq. (1).

$$\Delta_i = (\theta_{gbest} - \theta_i) + (\theta_{i,pbest} - \theta_i) \tag{7}$$

The direction generating equation is similar with the velocity updating equation of PSO algorithm [23]. They all used the global best and personal best. However, they are not the same. First, there is no inertia term in Eq. (7). Usually, the bacteria will run more than one times in chemotactic steps. An inertia term will enlarge the difference of between θ_{gbest}, $\theta_{i,pbest}$ and the current position tremendously. Second, there are no learning factors in Eq. (7). Because the direction will be normalized to unit vector and the learning factors is meaningless.

By social learning, the bacteria will move to better areas with higher probability as good information is fully utilized.

3.3 Adaptive search strategy

As it mentioned above, the constant step length will make the population hard to converge to the optimal point. In an intelligence optimization algorithm, it is important to balance its exploration ability and exploitation ability. The exploration ability ensures that the algorithm can search the whole space and escape from local optima. The exploitation ability guarantees that the algorithm can search local areas carefully and converge to the optimal point. Generally, in the early stage of an algorithm, we should enhance the exploration ability to search all the areas. In the later stage of the algorithm, we should enhance the exploitation ability to search the good areas intensively.

There are various step length varying strategies [24] [25]. In BFOLS, we use the decreasing step length. The step length will decrease with the fitness evaluations, as shown in Eq. (8). C_s is the step length at the beginning. C_e is the step length at the end. $nowEva$ is the current fitness evaluations count. $TotEva$ is the total fitness evaluations. In the early stage of BFOLS algorithm, larger step length provides the exploration ability. And at the later stage, small step length is used to make the algorithm turn to exploitation.

$$C = C_s - (C_s - C_e) \times nowEva / TotEva \tag{8}$$

To strengthen the idea further, an elaborate adaptive search strategy is introduced based on the decreasing step length mentioned above. In the new strategy, the bacteria's step lengths may vary from each other. And their values are related with their nutrient, which are calculated using Eq. (9).

$$C(i) = \begin{cases} C / Nutrient(i) & if\,(Nurtrient(i) > 0) \\ C & if\,(Nurtrient(i) \le 0) \end{cases} \tag{9}$$

With a higher nutrient value, the bacterium's step length is shortened further. This is also in accordance with the food searching behaviors in natural. The higher nutrient value indicates that the bacterium is located in potential rich-nutrient area with a larger probability. As a result, it is necessary to exploit the area carefully with smaller step length.

The pseudo code of BFOLS algorithm is listed in Figure 4.

```
Initialization
While (termination conditions are not met)
    $S$=size of the last population; $i$=0;
    while $i<S$
        $i=i+1$;
        $Jlast=J(i)$
        Generate a tumble angle for bacterium n;
        Update the position of bacterium n by Eq. (1);
        Recalculate the $J(i)$
        Update personal best and global best;
        $m$=0
        While ( $m<N_s$)
            If $J(i)<Jlast$
                $Jlast=J(i)$
                Run one more step using Eq. (1);
                Recalculate the $J(i)$;
                Update personal best and global best;
                $m=m+1$;
            Else
                $m=N_s$;
            End if
        End while
        If (Nutrition ($i$) is larger than split threshold value)
            Split bacterium $i$ into two bacteria; Break;
        End if
        If (Nutrition ($i$) is less than dead threshold value)
            Remove it from the population;
            $i=i-1$; $S=S-1$; Break;
        End if
        If (Nutrition ($i$) is less than 0 and rand<$P_e$ )
            Move bacterium $i$ to a random position;
        End if
    End while
End while
```

Figure 4: Pseudo code of BFOLS algorithm

4. EXPERIMENTS

In this section, first we will test the optimization ability of BFOLS algorithm on a set of benchmark functions. Several other intelligent algorithms will be employed for comparison, including original BFO, PSO and Genetic Algorithm (GA) [26]. Statistical techniques are also used [27] [28]. Iman-Davenport test and Holm method are employed to analyze the differences among these algorithms. As two extra control parameters N_{split} and N_{adapt} are introduced, the settings of the two parameters are then tested to determine their best values. At last, the varying tendency of population size in BFOLS is tested and analyzed.

4.1. Performance test of BFOLS

A. Benchmark Functions

The BFOLS algorithm was tested on a set of benchmark functions with dimensions of 2 and 20 respectively. The functions are listed in Table 1. Among them, f_1-f_6 are basic functions widely adopted by other researchers [29] [30], their global minima are all zero. f_7-f_{14} are shifted and rotated functions selected from CEC2005 test-bed, global minima of these functions are different to each other. For each function, its standard variable range is used. Function f_{10} is a special case. It has no bounds. The initialization range of this function is [0, 600], and the global optima is outside of its initialization range.

It should be mentioned that the bacteria may run different times in a chemotactic step. As a result, different computational complexity may be taken in each iteration for different algorithms and iterations count is no longer a reasonable measure. In order to compare the different algorithms, a fair measure method must be selected. In this paper, we use number of function evaluations (FEs) as a measure criterion, which is also used in many other works [31] [32] [33]. All algorithms were terminated after 20,000 function evaluations on dimension of 2 and 60,000 function evaluations on dimension of 20.

Table 1: Benchmark functions used in the experiment

	Function	Formulation	Variable ranges	f(x*)
f_1	Sphere	$f(x)=\sum_{i=1}^{D} x_i^2$	[-5.12, 5.12]	0
f_2	Rosenbrock	$f(x)=\sum_{i=1}^{D-1}\left(100(x_i^2-x_{i+1})^2+(1-x_i)^2\right)$	[−15, 15]	0
f_3	Rastrigin	$f(x)=\sum_{i=1}^{D}\left(x_i^2-10\cos(2\pi x_i)+10\right)$	[-10, 10]	0
f_4	Ackley	$f(x)=20+e-20e^{\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}x_i^2}\right)}-e^{\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right)}$	[−32.768, 32.768]	0
f_5	Griewank	$f(x)=\frac{1}{4000}\left(\sum_{i=1}^{D}x_i^2\right)-\left(\prod_{i=1}^{D}\cos(\frac{x_i}{\sqrt{i}})\right)+1$	[−600, 600]	0
f_6	Schwefel2.22	$f(x)=\sum_{i=1}^{D}\lvert x_i\rvert+\prod_{i=1}^{D}\lvert x_i\rvert$	[-10, 10]	0
f_7	Shifted Sphere	$f(x)=\sum_{i=1}^{D} z_i^2+f\_bias_1, z=x-o$	[-100, 100]	-450
f_8	Shifted Schwefel1.2	$f(x)=\sum_{i=1}^{D}(\sum_{j=1}^{i} z_j)^2+f\_bias_2, z=x-o$	[-100, 100]	-450
f_9	Shifted Rosenbrock	$f(x)=\sum_{i=1}^{D-1}(100(z_i^2-z_{i+1})^2+(z_i-1)^2)+f\_bias_6,$ $z=x-o$	[-100, 100]	390
f_{10}	Shifted Rotated Griewank	$f(x)=\sum_{i=1}^{D}\frac{z_i^2}{4000}-\prod_{i=1}^{D}\cos(\frac{z_i}{\sqrt{i}})+1+f\_bias_7,$ $z=(x-o)*M$	No bounds	-180

f_{11}	Shifted Rotated Ackley	$f(x)=-20\exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}z_i^2})-\exp(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi z_i))$ $+20+e+f\_bias_8, z=(x-o)*M$	[-32, 32]	-140
f_{12}	Shifted Rastrigin	$f(x)=\sum_{i=1}^{D}(z_i^2-10\cos(2\pi z_i)+10)+f\_bias_9,$ $z=(x-o)$	[-5, 5]	-330
f_{13}	Shifted Rotated Rastrigin	$f(x)=\sum_{i=1}^{D}(z_i^2-10\cos(2\pi z_i)+10)+f\_bias_{10},$ $z=(x-o)*M$	[-5, 5]	-330
f_{14}	Shifted Rotated Weierstrass	$f(x)=\sum_{i=1}^{D}(\sum_{k=0}^{k\max}[a^k\cos(2\pi b^k(z_i+0.5))])-D\sum_{k=0}^{k\max}[a^k\cos(2\pi b^k\cdot 0.5)]$ $+f\_bias_{11},\ z=(x-o)*M$	[-0.5, 0.5]	90

B. Parameter Settings for the Involved Algorithms

The population sizes *S* of all algorithms were 50. In original BFO algorithm, the parameters are set as followed: N_c=50, N_s=4, N_{re}=4, N_{ed}=10, P_e=0.25, *C*=0.1, S_r=*S*/2=25. The parameter settings are similar with that in Passino's work except N_c is smaller and N_{ed} is larger [4]. This is because the termination criterion has changed to be the function evaluations. Smaller N_c is selected to guarantee that the algorithm can run through the chemotactic, reproduction, eliminate and dispersal processes. Larger N_{ed} is selected to guarantee that the BFO algorithm will not terminate before the maximum function evaluations. In our BFOLS algorithm, as it has mentioned previously, N_c, N_{re}, N_{ed}, and S_r are no longer needed. N_s=4, P_e=0.25, which are the same with they are in BFO. The started step C_s=0.1(*Ub*-*Lb*), ended step C_e=0.00001(*Ub*-*Lb*) where *Lb* and *Ub* refer the lower bound and upper bound of the variables of the problems. This will make the algorithm suitable for problems of different scales. The step of the whole population deceases from C_s to C_e linearly, and step of each bacterium is calculated by the Eq. (9) mentioned above. The values of the two control parameters N_{split} and N_{adapt} are set to be 30 and 5. Standard PSO and GA algorithm was used in this experiment. In PSO algorithm, inertia weight ω decreased from 0.9 to 0.7. The learning factors *C1*=*C2*=2.0 [34]. V_{min}=0.1×*Lb*, V_{max}=0.1×*Ub*. In GA algorithm, single-point crossover is used, crossover probability is 0.95 and mutation probability is 0.1 [26].

C. Experiment Results and statistical analysis

The error values ($f(x)$-$f(x^*)$) of BFOLS, BFO PSO and GA algorithms on the benchmark functions with dimension of 2 and 20 are listed in Table 2 and Table 3 respectively. Mean and standard deviation values obtained by these algorithms for 30 times of independently runs are given. Best values of them on each function are marked as bold. As there are 14 functions on each dimension, it will take too much space to give all the convergence plots. Here only f_1-f_6's convergence plots are given, seen in Figure 5.

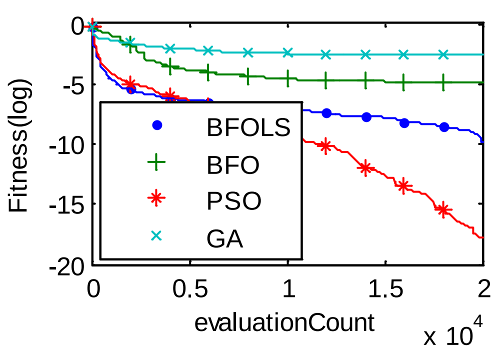
Table 2: Error values obtained by BFOLS, BFO, PSO and GA algorithms with dimension of 2

Function		BFOLS	BFO	PSO	GA
f_1	Mean	1.66266e-010	1.14309e-005	**1.48427e-018**	2.46555e-003
	Std	2.35439e-010	1.44400e-005	**3.38437e-018**	4.05650e-003

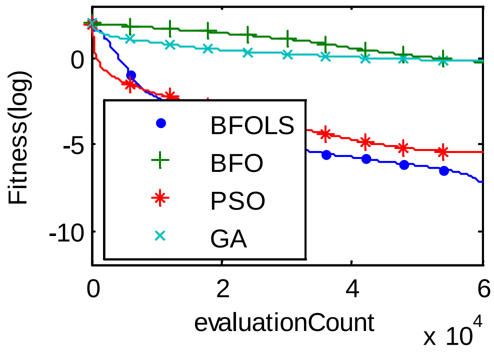
f_2	Mean	4.7678e-008	4.94397e-003	**4.26062e-015**	1.76305e+000
	Std	5.69309e-008	5.08761e-003	**9.56508e-015**	2.11319e+000
f_3	Mean	1.40217e-007	6.55588e-001	**3.07902e-015**	1.59279e-001
	Std	1.46112e-007	3.62200e-001	**8.83798e-015**	1.88468e-001
f_4	Mean	1.90708e-004	1.18787e-001	**1.06646e-008**	3.62955e-001
	Std	1.11928e-004	6.54950e-002	**1.36111e-008**	2.58463e-001
f_5	Mean	**1.17638e-006**	2.67927e-002	2.92970e-004	9.38310e-002
	Std	**2.53546e-006**	1.28042e-002	1.36535e-003	6.92769e-002
f_6	Mean	1.60310e-005	8.0626e-003	**2.47171e-009**	3.37681e-002
	Std	1.36623e-005	4.42736e-003	**4.17088e-009**	2.83197e-002
f_7	Mean	4.52226e-008	3.81605e-003	**0**	1.06661e+001
	Std	3.93677e-008	3.62032e-003	**0**	1.02199e+001
f_8	Mean	5.57271e-008	4.75829e-003	**5.68434e-015**	1.91645e+000
	Std	8.51665e-008	4.00577e-003	**1.73446e-014**	4.00548e+000
f_9	Mean	**2.15994e-006**	2.00971e-001	6.66463e-006	4.24819e+001
	Std	**2.21157e-006**	1.58643e-001	2.94369e-005	4.77406e+001
f_{10}	Mean	**1.64445e-004**	4.59019e-002	1.66967e-003	2.87956e-001
	Std	**8.78281e-004**	2.35742e-002	3.37535e-003	1.40164e-001
f_{11}	Mean	1.19137e+001	**9.69262e+000**	1.66669e+001	1.21447e+001
	Std	9.89835e+000	**5.40612e+000**	7.5811e+000	4.10647e+000
f_{12}	Mean	**4.63100e-008**	2.50975e-001	1.98992e-001	7.41804e-001
	Std	**5.64908e-008**	3.35611e-001	4.04787e-001	6.75828e-001
f_{13}	Mean	**3.49518e-008**	1.85255e-001	3.31653e-002	9.61984e-001
	Std	**2.31262e-008**	1.65632e-001	1.81654e-001	6.48141e-001
f_{14}	Mean	1.81406e-003	1.01915e-001	**9.25237e-006**	3.39486e-001
	Std	6.56816e-004	3.84900e-002	**6.56108e-006**	1.11498e-001

Table 3: Error values obtained by the BFOLS, BFO, PSO and GA algorithms with dimension of 20

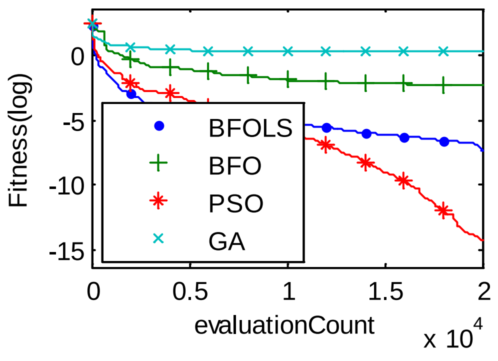
Function		BFOLS	BFO	PSO	GA
f_1	Mean	**6.89944e-008**	6.17635e-001	3.76236e-006	6.61625e-001
	Std	**4.37473e-008**	1.94300e-001	4.85485e-006	2.34433e-001
f_2	Mean	**2.36439e+001**	2.02447e+003	2.49117e+001	6.56345e+002
	Std	2.39321e+001	8.62087e+002	**2.12081e+001**	4.55687e+002
f_3	Mean	**1.09317e+001**	4.94922e+002	3.35838e+001	6.80152e+001
	Std	**5.90291e+000**	6.74151e+001	1.07949e+001	1.77908e+001
f_4	Mean	**1.29613e-003**	1.95483e+001	1.10071e+000	1.86919e+001
	Std	**2.60321e-003**	3.71818e-001	9.14650e-001	1.30767e-000
f_5	Mean	**2.92754e-002**	3.00705e+000	3.89587e-002	3.11937e+000
	Std	**2.12876e-002**	4.5062e-001	3.13277e-002	8.55760e-001
f_6	Mean	**8.20052e-004**	5.43634e+001	1.39941e-001	4.92696e+000
	Std	**2.08416e-004**	1.20626e+001	1.87068e-001	1.09457e+000
f_7	Mean	**2.00442e-005**	7.63447e+002	6.53663e+001	9.33638e+001
	Std	**1.01518e-005**	2.85716e+002	1.47148e+002	2.21348e+001
f_8	Mean	**1.96773e+001**	1.00992e+004	2.58670e+002	1.61144e+004
	Std	**3.32945e+001**	2.54700e+003	5.37769e+002	6.68451e+003
f_9	Mean	**3.20719e+001**	1.80082e+007	2.23033e+005	1.41913e+005
	Std	**2.84519e+001**	2.24514e+007	7.69626e+005	1.29572e+005
f_{10}	Mean	**1.74564e-001**	6.12735e+001	8.44887e+000	1.94291e+001
	Std	**2.39218e-001**	3.07002e+001	6.86895e+000	6.31473e+000
f_{11}	Mean	**2.07203e+001**	2.08745e+001	2.07481e+001	2.08665e+001
	Std	8.27531e-002	**5.76786e-002**	6.78494e-002	5.81818e-002
f_{12}	Mean	7.99944e+001	1.86911e+004	9.86338e+001	**5.00084e+001**
	Std	2.08287e+001	2.49192e+001	1.75115e+001	**1.01565e+001**
f_{13}	Mean	**1.45491e+002**	3.06702e+002	1.74580e+002	1.77517e+002
	Std	2.95067e+001	3.87859e+001	3.93332e+001	**2.79148e+001**
f_{14}	Mean	1.90236e+001	2.04888e+001	**1.44403e+001**	2.51640e+001
	Std	2.13827e+000	1.14255e+000	2.83690e+000	**7.57584e-001**



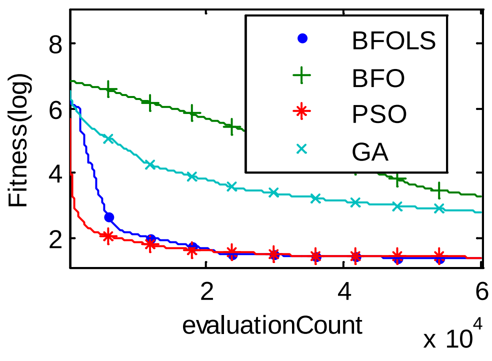
(a) Sphere-2D



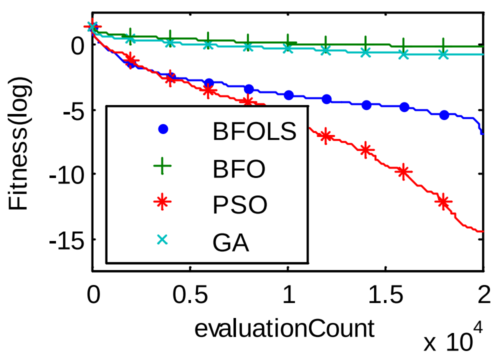
(b) Sphere-20D



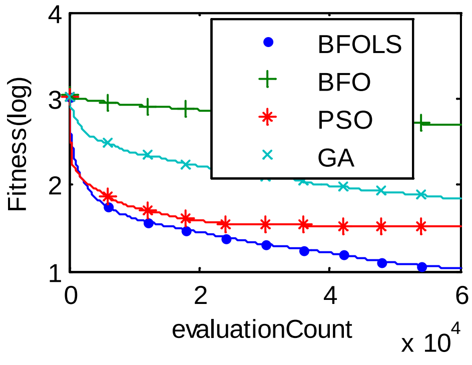
(c) Rosenbrock-2D



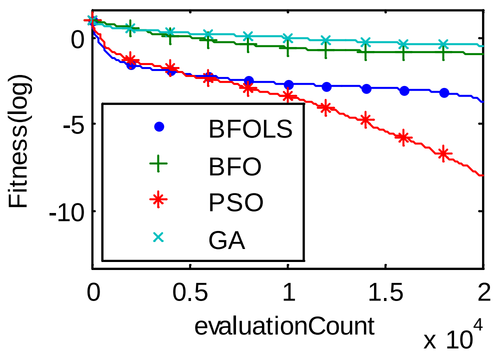
(d) Rosenbrock-20D



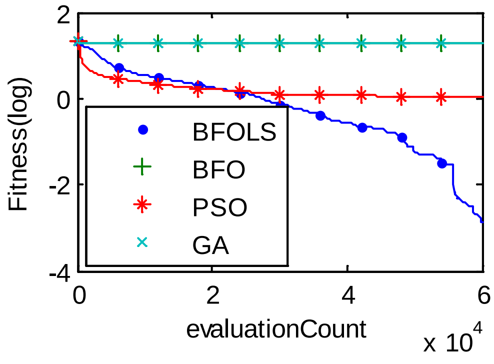
(e) Rastrigin-2D



(f) Rastrigin-20D

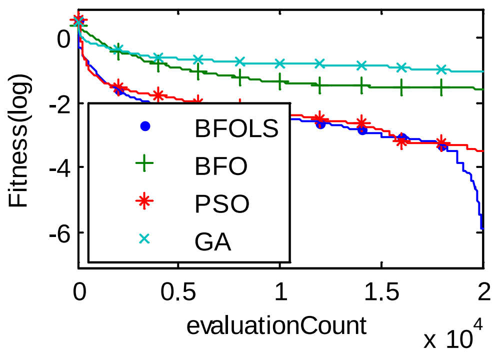


(g) Ackley-2D

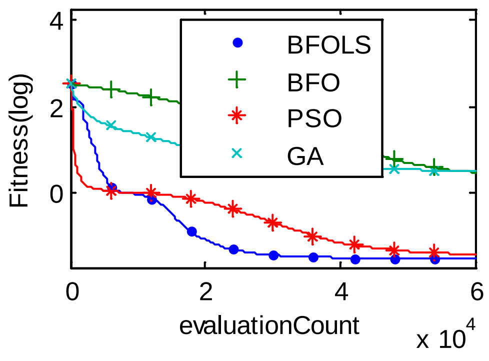


(h) Ackley-20D

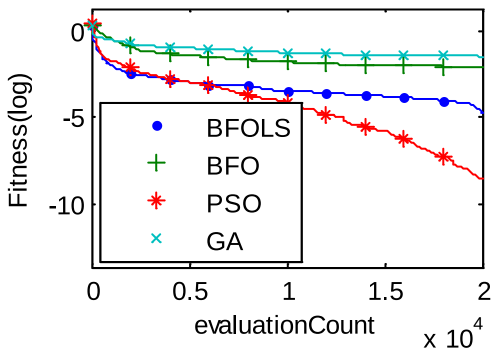
Figure 5: Convergence plots of BFOLS, BFO, PSO and GA on functions f_1-f_6 with dimension of 2 and 20



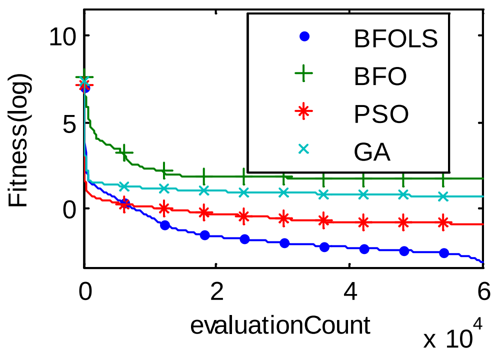
(i) Griewank-2D



(j) Griewank-20D



(k) Schwefel2.22-2D



(l) Schwefel2.22-20D

Figure 5: Continued

With dimension of 2, PSO obtained best mean error values on 8 functions. BFOLS performed best on 5 and BFO performed best on the rest one. With dimension of 20, BFOLS algorithm obtained best results on 12 functions of all 14. PSO and GA performed best on the rest two respectively. The average rankings of the four algorithms are listed in Table 4. The smaller the value is, the better the algorithm performs. On dimension of 2, the performance order is PSO> BFOLS> BFO> GA. And on dimension of 20, the performance order is BFOLS> PSO> GA> BFO.

Table 4: Average rankings of the four algorithms on dimension of 2 and 20

Dimension	BFOLS	BFO	PSO	GA
2	1.6429	2.9286	1.5714	3.8571
20	1.1429	3.7143	2.0714	3.0714

Table 5 shows the results of Iman-Davenport statistical test. The critical values are at the level of 0.05, which can be looked up in the *F*-distribution table with 3 and 39 degrees of freedom. On dimensions of 2 and 20, the Iman-Davenport values are all larger than their critical values, which mean that significant differences exist among the rankings of the algorithms under the two conditions.

Table 5: Results of the Iman-Davenport test

Dimension	Iman-Davenport Value	Critical Value α=0.05	Significant differences?
2	34.1852	2.85	Yes
20	42.3913	2.85	Yes

Holm tests were done as a post hoc procedure. With dimension of 2, PSO performed best. It is chosen as the control algorithm and the other algorithms will be compared with it. With dimension of 20, BFOLS algorithm is the control algorithm. The results of Holm test with dimensions of 2 and 20 are given in Table 6 and Table 7 respectively. The α/i values are with α=0.05. If α=0.10, the values are twice of the values listed.

As shown in Table 6, the *p* values of GA and BFO are smaller than their *α/i* values, which means that equality hypotheses are rejected and significant differences exist between these two algorithms and the control algorithm-PSO. The *p* value of BFOLS is larger than its *α/i* values, so the equality hypothesis can't be rejected. It denotes that no significant differences exist and it can be regard as equivalent to PSO. The situation is the same when α=0.10.

Table6: Results of Holm's test with dimension of 2

algorithm	*z*	*p* value	α/i	Significant differences?
GA	4.6843	2.8089E-6	0.0167	Yes
BFO	2.7813	0.0054	0.025	Yes
BFOLS	0.1464	0.8836	0.05	No

With dimension of 20, BFOLS algorithm is the control algorithm. The *p* values of BFO and GA are smaller than their *α/i* values, So BFOLS is significant better than the two algorithms. The equality hypothesis between BFOLS and PSO can't be rejected when α=0.05. However, when α=0.10, α/i value is 0.1 and the *p* value is smaller than it. So BFOLS is also significant better than PSO under the level of 0.1.

Table 7: Results of Holm's test with dimension of 20

algorithm	*z*	*p* value	α/i	Significant differences?
BFO	5.2699	1.3653E-7	0.0167	Yes
GA	3.9524	7.7373E-5	0.025	Yes
PSO	1.9030	0.0570	0.05	No

Overall, BFOLS shows significant improvement over the original BFO algorithm. And its optimization ability is better than the classic PSO and GA algorithms on higher dimensional problems, too.

4.2. Parameters setting test of BFOLS

In BFOLS, two extra parameters N_{split} and N_{adapt} are introduced. N_{split} is the initial split threshold. N_{adapt} makes the split threshold and the death threshold adjusted with the environment adaptively. To determine the best settings of these two parameters, we tested the algorithm on four benchmarks with different N_{split} and N_{adapt} values.

The four benchmark functions are Rosenbrock, Griewank, Ackley and Schwefel2.22. Each function was tested with dimensions of 2 and 20. N_{split} should a little larger so the bacteria will not reproduce sharply at first. N_{adapt} should larger than zero to let the split and death threshold vary adaptively. Test have been done that population size will reduce to zero and error occurs on some functions when N_{adapt} is zero. In the first group of tests, N_{adapt} were fixed to 5 and N_{split} were set to be 10, 20, 30, 50 and 100 separately. In the second group of tests, N_{split} were fixed to 30 and N_{adapt} were set to be 1, 2, 5, 20 and 60 separately. Results of BFOLS obtained with different parameter values are listed in Table 8 and Table 9.

It is clear from the table, on most benchmark functions, results obtained with different parameter values are almost at the same order of magnitude. That is to say, the performance of BFOLS is not that sensitive to the parameter values on most functions. However, on Ackley function with dimension of 20, the situations seem changed. While N_{adapt} was fixed, Results of BFOLS with N_{split} values of 10, 20, and 30 are much better than 50 and 100. While N_{split} was fixed, Results of BFO with N_{adapt} values of 5 and 20 are better than 1, 2 and 60 clearly. At the meanwhile, the average ranks show that it got the best rank with N_{split} were 30 and 100 in the first group of tests and with N_{adapt} was 5 in the second group of tests. Based on above facts, we recommend that N_{split} and N_{adapt} are set to be 30 and 5 or in the nearby range.

Table 8: Results of BFOLS with different N_{split} values under N_{adapt} =5

Function		N_{split}				
		10	20	30	50	100
Rosenbrock	Mean	5.03273e-007	9.32218e-008	4.76780e-008	1.7662e-008	1.48532e-008
2D	Rank	4	5	3	2	1
Rosenbrock	Mean	1.49457e+001	1.80154e+001	2.53310e+001	2.68780e+001	3.68623e+001
20D	Rank	1	2	3	4	5
Griewank	Mean	3.17617e-006	1.20719e-005	1.17638e-006	1.53849e-006	7.28572e-006
2D	Rank	3	5	1	2	4
Griewank	Mean	2.39571e-002	3.56497e-002	2.92754e-002	2.77610e-002	3.13200e-002
20D	Rank	1	5	3	2	4
Ackley	Mean	2.25585e-004	1.58819e-004	1.90708e-004	1.61910e-004	1.24685e-004
2D	Rank	5	2	4	3	1
Ackley	Mean	1.78481e-003	1.13355e-003	1.29613e-003	1.64810e+000	1.52429e+000
20D	Rank	3	1	2	5	4
Schwefel2.22	Mean	2.41253e-005	2.28243e-005	1.60310e-005	1.84355e-005	1.49146e-005
2D	Rank	5	4	2	3	1
Schwefel2.22	Mean	1.61622e-003	9.90601e-004	8.20052e-004	5.89764e-004	4.38522e-004
20D	Rank	5	4	3	2	1
Average Rank		3.375	3.5	2.625	2.875	2.625

Table 9: Results of BFOLS with different N_{adapt} values under N_{split}=30

Function		N_{adapt}				
		1	2	5	20	60
Rosenbrock	Mean	2.03042e-008	3.25644e-008	4.76780e-008	3.83752e-008	6.27226e-008
2D	Rank	1	2	4	3	5
Rosenbrock	Mean	2.07862e+001	1.85445e+001	2.53310e+001	1.21741e+001	2.00873e+001
20D	Rank	4	2	5	1	3
Griewank	Mean	1.21182e-006	7.3607e-006	1.17638e-006	8.76095e-006	4.21623e-006
2D	Rank	2	4	1	5	3
Griewank	Mean	4.68112e-002	4.45132e-002	2.92754e-002	3.10057e-002	4.52086e-002
20D	Rank	5	3	1	2	4
Ackley	Mean	1.49850e-004	1.34254e-004	1.90708e-004	1.63858e-004	1.56880e-004
2D	Rank	2	1	5	4	3
Ackley	Mean	7.99836e-001	6.77576e-001	1.29613e-003	2.79342e-003	1.55109e-001
20D	Rank	5	4	1	2	3
Schwefel2.22	Mean	2.61265e-005	2.08609e-005	1.60310e-005	1.20443e-005	1.82075e-005
2D	Rank	5	4	2	1	3
Schwefel2.22	Mean	6.70517e-004	8.60185e-004	8.20052e-004	9.97355e-004	8.10414e-004
20D	Rank	1	4	3	5	2
Average Rank		3.125	3	2.75	2.875	3.25

4.3. Population size varying simulation in BFOLS

As it has mentioned above, the population size of BFOLS may vary because the bacteria will split and die. As a result, the dynamic varying on function f_1-f_6 is tracked and recorded while the algorithm runs. The mean population size varying plots on the six functions with dimension of 2 and 20 are listed in Figure 6. The left subplot is with dimension of 2 and the right one is with dimension of 20. It can be seen that obvious regularities exist among both the two plots. With dimension of 2, the population size of BFOLS decreased firstly and increased at the end on all functions. With dimension of 20, population size increased fast at the beginning and then reached the peaks. After that, population size began to decrease. The varying plots probably correspond with two adaptive behaviors in the nature environment. As mentioned in Eq. (4), the nutrient updating is related with the improvement or deterioration of the position of the bacterium. On functions with dimension of two, the room for improvement is limited, which is similar with a saturated environment. The nutrient is limited and competitions between bacteria are fierce. As a result, bacteria die more than split and the population size decreases. At the end of the algorithm, the competition reduced as the number of bacteria decreased, so the bacteria reproduced more often adaptively. On functions with dimension of 20, there is much room for improvement and it could be regards as a eutrophic environment. Bacteria split easily and the population size increased sharply at first. Then it reached saturation. Competition increased and the population size began to decrease.

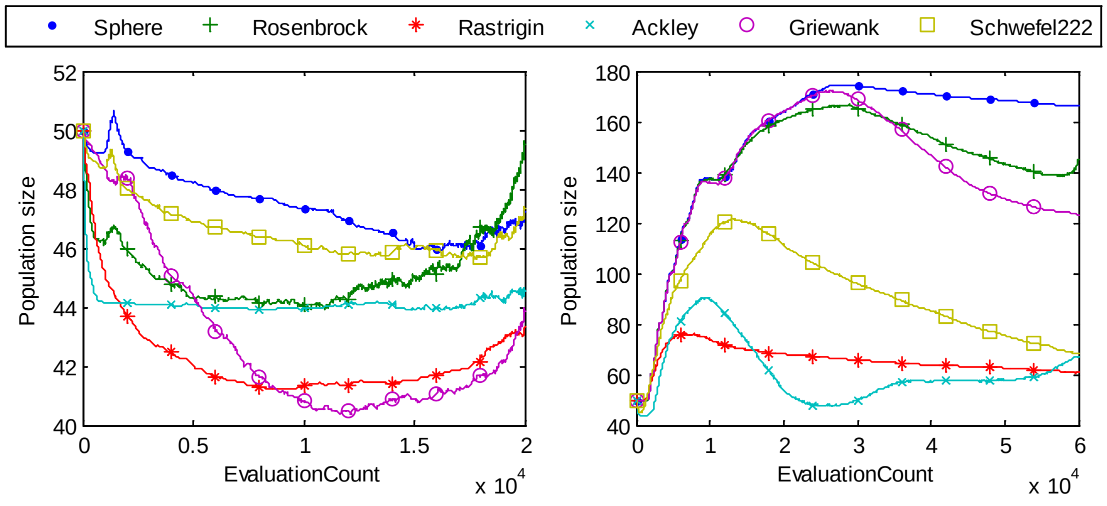


Figure 6: Mean population size varying plots of BFOLS on benchmark functions with dimensions of 2 (left) and 20 (right).

5. CONCLUSIONS

This paper analyzes the shortness of original Bacterial Foraging Optimization algorithm. To overcome its shortness, an adaptive BFO algorithm with Lifecycle and Social learning is proposed, which is named BFOLS. In the new algorithm, a lifecycle model of bacteria is founded. The bacteria will split, die or migrate dynamically in the foraging processes according to their nutrient. Social learning is also introduced in the chemotactic steps to guide the tumble direction. In BFOLS algorithm, the tumble angles are no longer generated randomly. Instead, they are produced using the information of the bacteria’s global best, the bacterium’ personal best and its current position. At last, an adaptive search step length strategy is employed. The step length of the bacteria population decreases linearly with iterations and the individuals adjusts their step lengths further according to their nutrient value.

To verify the optimization ability of BFOLS algorithm, it is tested on a set of benchmark functions with dimensions of 2 and 20. Original BFO, PSO and GA algorithms are used for comparison. With dimension of 2, it outperforms BFO and GA but a little worse than PSO. With dimension of 20, it shows significant better performance than GA and BFO. At the level of α=0.10, it is significant better than PSO, too. The settings of the new parameters are tested and discussed. N_{split} and N_{adapt} are recommend to be 30 and 5 or in the nearby range. The varying situations of population size are also tracked. With dimension of 2 and 20, their varying plots show obvious regularities and correspond with the population survival phenomenon in natural. Overall, the proposed BFOLS algorithm is a powerful algorithm for optimization. It offers significant improvements over original BFO, and show competitive performances compared other algorithms on higher dimensional problems.

Further research efforts could focus on the following aspects: First, other step length strategies can be used. Second, more benchmark functions with different dimensions could be tested.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant No. 61174164, 61003208, 61105067, 71001072, 71271140). And the authors are very grateful to the anonymous reviewers for their valuable suggestions and comments to improve the quality of this paper.

References

[1] M. Dorigo, L. M. Gambardella, “Ant Colony System: A cooperating learning approach to the travelling salesman problem,” *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53−66, 1997.

[2] J. Kennedy, R. C. Eberhart, “Particle swarm optimization,” *In: Proceedings of the 1995 IEEE International Conference on Neural Networks*, vol. 4, pp. 1942-1948, 1995.

[3] D. Karaboga, “An idea based on honey bee swarm for numerical optimization,” *Technical Report-TR06*, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.

[4] K. M. Passino,”Biomimicry of bacterial foraging for distributed optimization and control,” *IEEE Control Systems Magazine*, vol. 22, pp. 52-67, 2002

[5] S. Dasgupta, S. Das, A. Abraham, A. Biswas, “Adaptive computational chemotaxis in bacterial foraging optimization: an analysis,” *IEEE Trans. Evolutionary Computation*, vol. 13, no. 4, pp. 919-941, 2009.

[6] S. Das, S. Dasgupta, A. Biswas, A. Abraham, A. Konar, “On stability of the chemotactic dynamics in bacterial-foraging optimization algorithm,” *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 39 no. 3, pp. 670-679, 2009.

[7] H. Chen, Y. Zhu, and K. Hu, “Cooperative bacterial foraging optimization,” *Discrete Dynamics in Nature and Society*, vol. 2009, Article ID 815247, 17 pages, 2009.

[8] D. H. Kima, A. Abrahamb and J. H. Cho, “A hybrid genetic algorithm and bacterialforaging approach for global optimization”, *Information Sciences*, vol. 177, no. 18, pp. 3918-3937, 2007.

[9] S. Das, A. Biswas, S. Dasgupta, A. Abraham, “Bacterial Foraging Optimization Algorithm, Theoretical Foundations, Analysis, and Applications,” *Foundations of Computational Intelligence* , vol. 203, pp. 23-55,2009.

[10] Y. Liu and K. M. Passino, “Biomimicry of social foraging bacteria for distributed optimization: Models, Principles, and Emergent Behaviors,” *Journal of Optimization Theory and Application*, vol. 115, no. 3, pp. 603--628, 2002.

[11] C. Wu, N. Zhang, J. Jiang and J. Yang, “Improved bacterial foraging algorithms and their applications to job shop scheduling problems,” *Lecture Notes in Computer Science*, vol. 4431, pp. 562 - 569, 2007.

[12] S. Dasgupta, S. Das, A. Biswas, A. Abraham, “Automatic circle detection on digital images with an adaptive bacterial foraging algorithm,” *Soft Comput.* vol. 14, no. 11, pp. 1151-1164, 2010.

[13] R. Majhi, G. Panda, B..Majhi and G. Sahoo, “Efficient prediction of stock market indices using adaptive bacterial foraging optimization (ABFO) and BFO based techniques,” *Expert Systems with Applications*, vol. 36, no. 6, pp. 10097-10104 , 2009.

[14] D. DeRosier, “The turn of the screw: The bacterial flagellar motor,” *Cell*, vol. 93, pp. 17-20, 1998.

[15] H. Berg and D. Brown, “Chemotaxis in escherichia coli analysed by three-dimensional tracking,” *Nature*, vol. 239, pp. 500-504, 1972.

[16] W. M. Korani, H. T. Dorrah and H. M. Emara, “Bacterial foraging oriented by Particle Swarm Optimization strategy for PID tuning,” *2009 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA),* Daejeon, pp. 445-450, 2009.

[17] A. Biswas, S. Das, A. Abraham, S. Dasgupta, “Stability analysis of the reproduction operator in bacterial foraging optimization,” *Theoretical Computer Science*, vol. 411, no. 21, pp. 2127-2139, 2010.

[18] H. Shen, Y. Zhu, L. Jin and H. Guo, “Lifecycle-based swarm optimization method for constrained optimization,” *Journal Of Computers*, vol. 6, no. 5, pp. 913-922, 2011.

[19] B. Niu, Y. Zhu, X. He and H. Shen, “A lifecycle model for simulating bacterial evolution,” *Neurocomputing*, vol. 72, pp. 142-148, 2008.

[20] R. C. Eberhart , Y. Shi and J. Kennedy, Swarm Intelligence, Morgan Kaufmann, 2001.

[21] R. C. Eberhart and J. Kennedy, “A new optimizer using particle swarm theory,” *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, Nagoya, Japan, pp. 39- 43, 1995.

[22] D. Karaboga, B. Akay, “A comparative study of Artificial Bee Colony algorithm,” *Applied Mathematics and Computation,* vol. 214, pp. 108-132, 2009.

[23] Y. Shi, and R. C. Eberhart, “A modified particle swarm optimizer,” *Proceedings of the IEEE Congress on Evolutionary Computation, Piscataway*, pp. 303-308, 1998.

[24] H. Chen, Y. Zhu, K. Hu, “Self-Adaptation in Bacterial Foraging Optimization algorithm,” *Proceedings of the 3th International Conference on Intelligent System & Knowledge Engineering*, Xiamen, China, pp. 1026-1031, 2008.

[25] B. Zhou, L. Gao and Y. Dai, “Gradient methods with adaptive step-sizes,” *Computational Optimization And Applications* vol. 35, no. 1, pp. 69-86, 2006.

[26] J. J. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, I975.

[27] J. Derrac, S. García, D. Molina, F. Herrera, “A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms,” *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3-18, 2011.

[28] S. García, A. Fernández, J. Luengo,F. Herrera, “A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability,” *Soft Comput* , vol. 13, no. 10, pp. 959-977, 2009.

[29] D. Karaboga and B. Basturk, “A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (abc) algorithm,” *Journal of Global Optimization*, vol. 39, no. 3, pp. 459-471, 2007.

[30] W. Zou, Y. Zhu, H. Chen, and X. Sui, “A clustering approach using cooperative artificial bee colony algorithm,” *Discrete Dynamics in Nature and Society*, vol. 2010, Article ID 459796, 16 pages, 2010.

[31] X. Yan, Y. Zhu, W. Zou, “A hybrid artificial bee colony algorithm for numerical function optimization,” *2011 11th International Conference on Hybrid Intelligent Systems (HIS)*, pp. 127- 132, 2011

[32] J. J. Liang, A. K. Qin, P. N. Suganthan and S. Baskar, “Comprehensive learning particle swarm optimizer for global optimization of multimodal functions,” *IEEE Transcations on Evolutionary Computing*, vol. 10, pp. 281-295, 2006.

[33] X. Yan, Y. Zhu, W. Zou, L. Wang, “A new approach for data clustering using hybrid artificial bee colony algorithm,” *Neurocomputing,* vol. 97, pp. 241-250, 2012.

[34] Y. Shi and R. C. Eberhart, “Empirical study of particle swarm optimization,” *in Proceedings of the IEEE Congress on Evolutionary Computation(CEC’99)*, Piscataway, NJ, USA, 3 (1999), pp.1945-1950.