JOURNAL OF PURE AND APPLIED MICROBIOLOGY, September 2013. Vol. 7(3), p. 2117-2122

Discrete Bacterial Foraging Optimization

Liwei Tian[1], Yichuan Shao[2]* and Hongwei Zhao[2]

[1]Personnel Division, Shenyang University, Shenyang - 110 014, China.
[2]College of Information Engineering, Shenyang University, Shenyang, 110 014, China.

(Received: 21 June 2013; accepted: 10 August 2013)

Bacterial Foraging Optimization (BFO) is a recently developed nature-inspired swarm intelligence algorithm, which is based on the foraging behavior of E. coli bacteria. In order to apply BFO in discrete landscape, a binary version of adaptive BFO (BABFO) algorithm is proposed in this manuscript. Unlike the original BFO algorithm, the proposed BABFO represents a food source as a discrete binary variable and applies adaptive operators to change the foraging trajectories of the individual bacterium. With four mathematical benchmark functions, BABFO is proved to have significantly better performance than the other two successful discrete optimizer, namely the genetic algorithm (GA) and particle swarm optimization (PSO).

Key words: Discrete optimization; Bacterial foraging; Swarm intelligence.

A novel evolutionary computation technique called bacterial foraging optimization (BFO) has been proposed recently[1-2]. In this scheme, the foraging (methods for locating, handling, and ingesting food) behavior of *E. coli* bacteria present in our intestines is mimicked. They undergo different stages such as chemotaxis, swarming, reproduction, and elimination and dispersal. In the chemotaxis stage, it can have tumble followed by a tumble or a tumble followed by a run. On the other hand, in swarming, each *E. coli* bacterium will signal other via attractants to swarm together. Furthermore, in reproduction the least healthy bacteria die and the other healthiest bacteria each split into two bacteria, which are placed in the same location. Besides, in elimination and dispersal, any one bacterium is eliminated from the total set just by dispersing it to a random location on the optimization domain.

Due to its simplicity and efficiency, the BFO algorithm has been applied to solve many practical optimization problems. Until now, BFO has been applied successfully to some engineering problems, such as constrained optimization problems, neural networks and clustering[3-6]. As mentioned above, the original version of BFO algorithm is only able to optimize continuous problems. However, many optimization problems are set in a space featuring discrete, qualitative distinctions between variables and between levels of variables. Typical examples include problems which require the ordering or arranging of discrete elements, as in scheduling and routing problems. Besides these pure combinatorial problems, researchers frequently cast floating-point problems in binary terms, and solve them in a discrete number space. As any problem, discrete or continuous, can be expressed in a binary notation, it is seen that an optimizer which operates on two-valued functions might be advantageous. Hence, this paper aims at developing a novel binary bacterial foraging optimization with adaptive strategy, which can solve general binary optimization benchmarks and more complex real-word discrete problems. In the proposed BABFO, we develop a differential expression, in which the relevant variables are interpreted in terms of changes of probabilities that changed each iteration. The main feature of this new operator is that it works in binary space, while still maintains the major characteristics of the original BFO's expression. In addition, in order to improve the

* To whom all correspondence should be addressed.
E-mail: yichuan_shao@sina.com

convergence speed and accuracy, the adaptive strategy is applied in the proposed model.

The rest of the paper is organized as follows. Section 2 gives a simple description of classical BFO. Section 3 motivates and describes the BABFO algorithm. Section 4 presents the experimental settings and results for each algorithm. Section 5 concludes the paper.

Bacterial Foraging Optimization

Natural selection tends to eliminate animals with poor foraging strategies and favor the propagation of genes of those animals that have successful foraging strategies, since they are more likely to enjoy reproductive success. After many generations, poor foraging strategies are either eliminated or shaped into good ones. Similar social foraging capabilities have also been a source of inspiration to some authors in the area of distributed Optimization and Control.

Based on the biology and physics underlying the foraging behavior of *E. coli* bacteria, Passino and Liu[1] exploit a variety of bacterial swarming and social foraging behaviors, discussing how the control system on the *E. coli* dictates how foraging should proceed. In the bacterial foraging process, four motile behaviors (chemotaxis, swarming, reproduction, elimination and dispersal) are mimicked.

Chemotaxis: a chemotactic step can be defined as a tumble followed by a tumble or a tumble followed by a run lifetime. To represent a tumble, a unit length random direction, say, $\phi(j)$ is generated; this will be used to define the direction of movement after a tumble. In particular

$$\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i)\phi(j) \quad ...(1)$$

where $\phi^i(j,k,l)$ represents the ith bacterium at jth chemotactic kth reproductive and lth elimination and dispersal step. $C(i)$ is the size of the step taken in the random direction specified by the tumble (run length unit).

Swarming

E.coli cells can cooperatively self-organize into highly structured colonies with elevated environmental adaptability using an intricate communication mechanism (e.g. quorum-sensing, chemotactic signaling and plasmid exchange). Roughly speaking, the cells provide an attraction signal to each other so they swarm together. The mathematical representation for swarming can be represented by

$$J_{cc}(\theta, P(j,k,l)) = \sum_{i=1}^{S} J_{cc}^i(\theta, \theta^i(j,k,l))$$

$$= \sum_{i=1}^{S}\left[-d_{attract}\exp\left(-w_{attract}\sum_{m=1}^{P}(\theta_m - \theta_m^i)^2\right)\right]$$

$$+\sum_{i=1}^{S}\left[h_{repelent}\exp\left(-w_{repelent}\sum_{m=1}^{P}(\theta_m - \theta_m^i)^2\right)\right] \quad ...(2)$$

Where $J_{cc}(\phi,P(j,k,l)$ is the cost function value to e added to be added to the actual cost function to be minimized to present a time varying cost function, S is the total number of bacteria, p is the number of parameters to be optimized which are present in each bacterium, and $d_{attract}, w_{attract}, h_{repelent}, w_{repelent}$ are different coefficients that are to be chosen properly.

Reproduction

The least healthy bacteria die and the other healthier bacteria each split into two bacteria, which are placed in the same location. This makes the population of bacteria constant.

Elimination and Dispersal

It is possible that in the local environment, the lives of a population of bacteria changes either gradually (e.g., via consumption of nutrients) or suddenly due to some other influence. Events can occur such that all the bacteria in a region are killed or a group is dispersed into a new part of the environment. They have the effect of possibly destroying the chemotactic progress, but they also have the effect of assisting in chemotaxis, since dispersal may place bacteria near good food sources. From a broad perspective, elimination and dispersal are parts of the population-level long-distance motile behavior.

Binary Adaptive Bacterial Foraging Optimization

The position clipping boundary condition

In order to restrict the bees' positions within the range [0,1], A normalization method and a threshold level has to be introduced to map all real valued numbers of $\phi(j,k,l)$ to the range [0,1]. The round function and the position clipping boundary condition (PCBC) can be used to accomplish this last modification. The resulting

change in position then is defined by the following rule:

$$\theta^i(j+1,k,l) = round(\theta^i(j+1,k,l))$$
$$\text{if} \quad \theta_d^i \geq UB, \text{ then } \theta_d^i = UB$$
$$\text{else if} \quad \theta_d^i \leq LB, \text{ then } \theta_d^i = LB \quad ...(5)$$

Here the $round(\bullet)$ function rounds the elements of each bacterium to the nearest integers, d is the dimension number of each bacterium, and the PCBC strategy handles the bounded search space. Once the new food source $\phi(j+1,k,l)$ is obtained, it will be evaluated and compared to $\phi(j,k,l)$. If the fitness of $\phi(j+1,k,l$ is equal to or better than that of $\phi(j,k,l)$, $\phi(j+1,k,l$ will replace and become a new member of the population; otherwise is retained. In other words, a greedy selection mechanism is employed as the selection operation between the old and the current food sources.

Self-adaptive strategy

In BABFO evolution process, each bacterium displays alternatively two distinct search states[8]:

(1) *Exploration* state, during which the bacterium employs a large run-length unit to explore the previously unscanned regions in the search space as fast as possible.

(2) *Exploitation* state, during which the bacterium uses a small run-length unit to exploit the promising regions slowly in its immediate vicinity.

Each bacterium in the colony permanently maintains an appropriate balance between *Exploration* and *Exploitation* states by varying its own run-length unit adaptively. This is achieved by taking into account two decision indicators: a fitness improvement and no improvement registered lately. The criteria that determine the adjustment of individual run-length unit and the entrance into one of the states are the following:

Criterion-1: if the bacterium discovers a new promising domain, the run-length unit of this bacterium is adapted to another smaller one. Here "discovers a new promising domain" means this bacterium register a fitness improvement beyond a certain precision from the last generation to the current. Following *Criterion-1*, the bacterium's behavior will self-adapt into Exploitation state.

Criterion-2: if the bacterium's current fitness is unchanged for a number K (user-defined) of consecutive generations, then augment this bacterium's run-length unit and this bacterium

Table 1. Results comparison of three algorithms on Goldberg for 30 runs

D	BABFO			BPSO			BGA		
	Best fitness	Mean fitness	Stardard deviation	Best fitness	Mean fitness	Stardard deviation	Best fitness	Mean fitness	Stardard deviation
30	0.0900	0.1840	0.0699	0.1000	0.3867	0.1432	0.2000	0.4700	0.1418
60	0.2250	0.5760	0.2752	0.6000	0.9900	0.2090	0.6000	1.1700	0.2842
90	0.2800	1.6173	0.6929	1.0000	1.5567	0.2402	1.2000	2.0567	0.4321
120	0.3075	2.6713	1.2235	1.6000	2.1867	0.3014	2.3000	3.3267	0.4510

Table 2. Results comparison of three algorithms on Bopilar for 30 runs

D	BABFO			BPSO			BGA		
	Best fitness	Mean fitness	Stardard deviation	Best fitness	Mean fitness	Stardard deviation	Best fitness	Mean fitness	Stardard deviation
30	0.1200	0.1750	0.0285	0	0.4067	0.1530	0.6000	0.7600	0.1429
60	0.2100	0.2600	0.0186	1	1.4600	0.2472	1.4000	1.8000	0.1576
90	0.2800	0.2990	0.0124	2.0000	2.5333	0.2537	2.2000	2.6667	0.2057
120	0.2625	0.3212	0.0199	3.0000	3.5333	0.2591	3.0000	3.6600	0.2581

enters Exploration state. This situation means that the bacterium searches on an un-promising domain or the domain where this bacterium focuses its search has nothing new to offer.

Numerical Examples And Results

In order to fully evaluate the performance of BABFO on discrete problems, we have employed a carefully chosen set of discrete benchmark functions[7] as follows:

Golderg's order-3

The fitness f of a bit-string is the sum of the result of separately applying the following function to consecutive groups of three components each:

$$f_1(x) = \begin{cases} 0.9 \; if \;\; |y| = 0 \\ 0.6 \; if \;\; |y| = 1 \\ 0.3 \; if \;\; |y| = 2 \\ 1.0 \; if \;\; |y| = 3 \end{cases} \quad ...(4)$$

If the string size (i.e. the dimension of the problem) is D, the maximum value is D/3 for the string 111…111. In practice, we will then use as fitness the value D/3-f so that the problem is now to find the minimum 0.

Bipolar order-6

The fitness f is the sum of the result of applying the following function to consecutive groups of six components each:

$$f_2(x) = \begin{cases} 1.0 \; if \;\; |y| = 0 \; or \; 6 \\ 0.0 \; if \;\; |y| = 1 \; or \; 5 \\ 0.4 \; if \;\; |y| = 2 \; or \; 4 \\ 0.8 \; if \;\; |y| = 3 \end{cases} \quad ...(5)$$

The maximum value is D/6. In practice, we will use as fitness the value D/6-f so that the problem is now to find the minimum 0.

Mulenbein's order-5

The fitness f is the sum of the result of applying the following function to consecutive groups of five components each:

$$f_3(x) = \begin{cases} 4.0 \; if \; y = 00000 \\ 3.0 \; if \; y = 00001 \\ 2.0 \; if \; y = 00011 \\ 1.0 \; if \; y = 00111 \\ 3.5 \; if \; y = 11111 \\ 0.0 \;\; ortherwise \end{cases} \quad ...(6)$$

Table 3. Results comparison of three algorithms on Muhlenbein for 30 runs

D	BABFO			BPSO			BGA		
	Best fitness	Mean fitness	Stardard deviation	Best fitness	Mean fitness	Stardard deviation	Best fitness	Mean fitness	Stardard deviation
30	0	0	0	0	0.8500	0.5594	0	1.3833	1.7747
60	0	0	0	0	1.5167	0.8457	0.5000	4.6833	2.4300
90	0	0	0	1	3.2000	1.6744	4.5000	11.916	4.6887
120	0	0	0	3	7.3000	2.8545	12	19.650	5.3902

Table 4. Results comparison of three algorithms on Clerc's Zebra-3 for 30 runs

D	BABFO			BPSO			BGA		
	Best fitness	Mean fitness	Stardard deviation	Best fitness	Mean fitness	Stardard deviation	Best fitness	Mean fitness	Stardard deviation
30	0	0	0	0.1000	0.3567	0.1194	0.2000	0.4833	0.1392
60	0	0	0	0.6000	0.9967	0.2109	0.7000	1.2067	0.2363
90	0	0	0	0.9000	1.6267	0.2935	1.3000	1.9067	0.3433
120	0	0	0	1.5000	2.1300	0.2984	1.8000	3.1433	0.6191

The maximum value is 3.5D/5. In practice, the value 3.5D/5-f is use as the fitness so that the problem is now to find the minimum 0.

Clerc's Zebra-3

The fitness f is the sum of the result of applying the following function to consecutive groups of three components each, if the rank of the group is even (first rank = 0):

$$f_{4a}(x) = \begin{cases} 0.9 \; if \;\; |y| = 0 \\ 0.6 \; if \;\; |y| = 1 \\ 0.3 \; if \;\; |y| = 2 \\ 1.0 \; if \;\; |y| = 3 \end{cases} \quad ...(7)$$

If the rank of the group is odd:

$$f_{4b}(x) = \begin{cases} 0.9 \; if \;\; |y| = 3 \\ 0.6 \; if \;\; |y| = 2 \\ 0.3 \; if \;\; |y| = 1 \\ 1.0 \; if \;\; |y| = 0 \end{cases} \quad ...(8)$$

In practice, we will then use as fitness the value D/3-f so that the problem is now to find the minimum 0.

For comparison, the proposed BABFO, the binary version of GA[8] and PSO[9] were tested on these benchmark functions. The population size for all algorithms was set at 120. The max generation

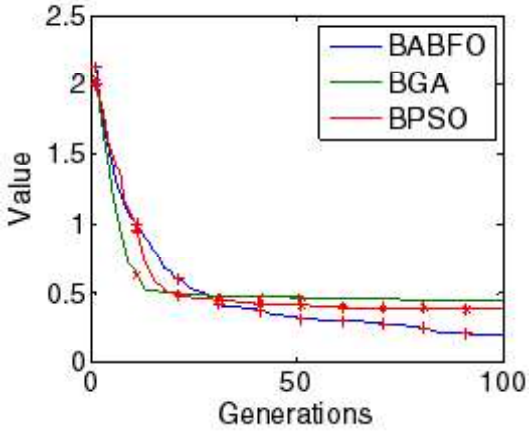

Fig. 1. Goldberg order-3 function

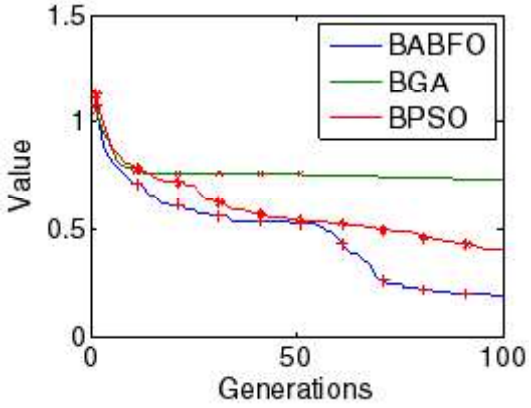

Fig. 2. Bipolar order-6 function

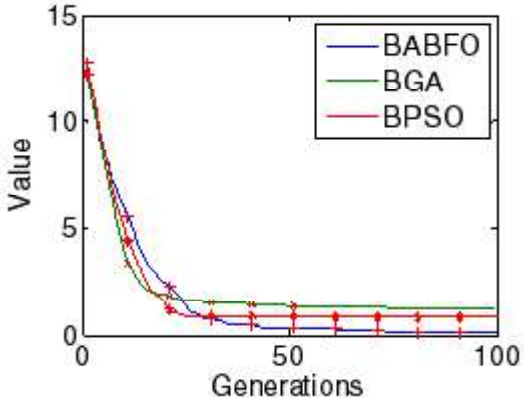

Fig. 3. Multimodal problem

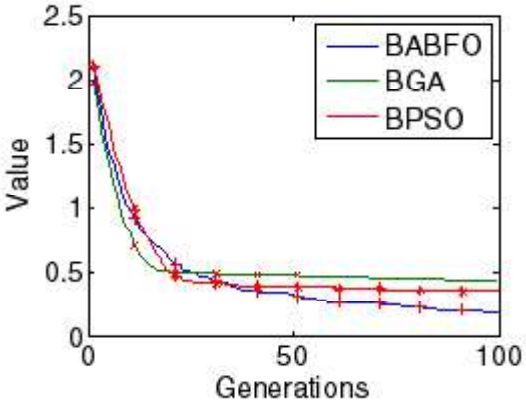

Fig. 4. Clerc Zebra-3 function

of each run is 1000. For BABFO, we take P_{ed}=0.25, N_c=100, N_s=4, N_{re}=5 and N_{ed}=2, then BABFO performs totally 1000 chemotactic steps in each run, which make a fair comparison in regard of the parameter values. For BGA, single point crossover operation with the rate of 0.8 was employed and mutation rate was set to be 0.01. For BPSO, the learning rate parameters were set to the values c_1= c_2=2 and the inertia weight w=1.

In this experiment, all algorithms are tested on 30, 60, 90 and 120 dimensions of each benchmark function, respectively. The experimental results, including the best, mean and standard deviation of the function values found in 30 runs are proposed in Table 1-4. The mean convergence results of 4 functions with 120 dimensions are showed in Fig.1-4.

From the results, we can observe that BABFO obtain an obviously remarkable performance. It can be seen from Fig. 1-4 that BABFO converged greatly faster and to significantly better results - the minimum of functions $f_1 \sim f_4$ - than BGA and BPSO for all discrete cases.

CONCLUSIONS

Since the original BFO algorithm cannot be directly applied to solve discrete problems, this paper proposed a novel discrete bacterial foraging algorithm - BABFO, which works in binary space, while still maintains the major characteristics of the original BFO's expression. From the simulation results, it is concluded that the performance of the proposed algorithm is better than BGA and BPSO on four benchmark functions.

The Future work should focus on how practically useful the BABFO algorithm are for engineering optimization problems. These depend on extensive evaluation on many benchmark functions and real-world problems.

ACKNOWLEDGMENTS

This work was supported in part by the International S&T Cooperation Program of China (ISTCP) under Grant 2011DFA91810-5 and Program for New Century Excellent Talents in University of Ministry of Education of China under Grant NCET-12-1012.

REFERENCES

1. Müeller, S., Marchetto, J., Airaghi, S., Koumoutsakos, P., Optimization based on bacterial chemotaxis. *IEEE Trans. on Evolutionary Computation,* 2002; **6**(1): 16–29.
2. Passino, K. M., Biomimicry of Bacterial Foraging for Distributed Optimization and Control. IEEE Control System Magazine, 2002; 52-67.
3. Kim, D. H., Cho, J. H., 2005. Adaptive Tuning of PID Controller for Multivariable System Using Bacterial Foraging Based Optimization. AWIC 2005; LNAE 3528, pp. 231-235.
4. Mishra, S., A hybrid least square-fuzzy bacterial foraging strategy for harmonic estimation. IEEE Trans. *Evolutionary Computation.* 2005; **9**(1): 61-73.
5. Tripathy, M., Mishra, S., Lai, L.L. and Zhang, Q.P., Transmission Loss Reduction Based on FACTS and Bacteria Foraging Algorithm. PPSN 2006; 222-231.
6. Kim, D.H., Cho, C. H., Bacterial Foraging Based Neural Network Fuzzy Learning. IICAI 2005, pp. 2030-2036.
7. Clerc, M. Discrete Particle Swarm Optimization, New Optimization Techniques in Engineering, Springer-Verlag, 2004.
8. Chen, H.N., Zhu, Y.L., Hu, K.Y., Adaptive Bacterial Foraging Optimization, Abstract and Applied Analysis, 2011.
9. Sumathi, S., Hamsapriya, T., Surekha, P., 2008. Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab. Springer-verlag, Berlin.
10. Shi, Y., Eberhart, R., Empirical study of particle swarm optimization. In: Proceedings of Congress on Evolutionary Computation, 1999; 1945-1950.