Self-Adaptation in Bacterial Foraging Optimization Algorithm*

Hanning Chen[1, 2], Yunlong Zhu[1], Kunyuan Hu[1]
[1]*Key Laboratory of Industrial Informatics, Shenyang Institute of Automation, Chinese Academy of Sciences, 110016, Shenyang, China,*
[2]*School of Graduate, Chinese Academy of Sciences, 100039, Beijing, China*
{chenhanning, ylzhu, hukunyuan}@sia.cn

Abstract

Bacterial Foraging Optimization (BFO) is a recently developed nature-inspired optimization algorithm, which is based on the foraging behavior of E. coli bacteria. However, BFO possesses a poor convergence behavior over complex optimization problems as compared to other nature-inspired optimization techniques like Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). This paper first analyzes how the run-length unit parameter controls the exploration and exploitation ability of BFO, and then presents a variation on the original BFO algorithm, called the Self-adaptive Bacterial Foraging Optimization (SA-BFO), employing the adaptive search strategy to significantly improve the performance of the original algorithm. This is achieved by enabling SA-BFO to adjust the run-length unit parameter dynamically during evolution to balance the exploration/exploitation tradeoff. Application of SA-BFO on several benchmark functions shows a marked improvement in performance over the original BFO.

1. Introduction

In 2001, Bacterial Foraging Optimization (BFO) algorithm has been developed to model the bacterial foraging behavior for solving optimization problems [1]. Recently, the BFO algorithm has been applied successfully to some engineering problems [2, 3, 4]. However, experimentation with complex problems reveals that the BFO algorithm possesses a poor convergence behavior and its performance heavily decreases with the growth of the search space dimensionality and the problem complexity.

In order to improve the BFO's searching performance, we propose the Self-adaptive Bacterial Foraging Optimization (SA-BFO) in the present paper. In stead of the simple description of chemotactic behavior in original BFO, SA-BFO also incorporates the adaptive search strategy, which allows each bacterium strikes a good balance between exploration and exploitation during algorithmic execution by tuning its run-length unit self-adaptively. In order to evaluate the performance of SA-BFO, extensive studies based on a set of 4 well-known benchmark functions have been carried out. For comparison purposes, the work also implemented a real-coded Genetic Algorithm (GA), the standard Particle Swarm Optimization (PSO) and the original BFO on these functions respectively. Simulation results are encouraging: the SA-BFO algorithm has markedly superior search performance when compared to the original BFO, whilst maintains the similar or even superior performance compared to PSO and GA.

The rest of the paper is organized as follows. In Section 2, we will give the briefly reviews of the original BFO algorithm. The in-depth analysis of the influence of the run-length unit parameter on the bacterial behavior is also presented here. Then our Self-adaptive Bacterial Optimization model will be introduced and its implementation details will be given in Section 3. In Section 4, the experiment studies of the proposed SA-BFO and other algorithms are presented with descriptions of the benchmark functions, experimental settings and the experimental results.

2. The BFO algorithm

2.1. Step-by-step algorithm

In what follows we briefly outline the original BFO algorithm step by step:

* This work was supported in part by the National 863 plans projects of China under Grant 2006AA04A124 and 2006AA04A117.

978-1-4244-2197-8/08/$25.00 ©2008 IEEE

[Step 1] Initialize parameters n, S, N_c, N_s, N_{re}, N_{ed}, P_{ed}, $C(i)$ (i=1,2,…,S), θ^i. Where,

n: Dimension of the search space,

S: The number of bacterium,

N_c: chemotactic steps,

N_s: swim steps,

N_{re}: reproductive steps,

N_{ed}: elimination and dispersal steps,

P_{ed}: probability of elimination,

$C(i)$: the run-length unit during each run or tumble.

[Step 2] Elimination-dispersal loop: $l = l$+1.

[Step 3] Reproduction loop: $k = k$+1.

[Step 4] Chemotaxis loop: $j = j$+1.

[substep **a]** For i = 1=1, 2,…, S, take a chemotactic step for bacteria i as follows.

[substep **b]** Compute fitness function, $J(i,j,k,l)$.

[substep **c]** Let $J_{\text{last}} = J(i,j,k,l)$ to save this value since we may find better value via a run.

[substep **d]** Tumble: Generate a random vector $\Delta(i) \in R^n$ with each element $\Delta_m(i)$, m = 1, 2, …, S, a random number on [-1, 1].

[substep **e]** Move: Let

$$\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i)\frac{\Delta(i)}{\sqrt{\Delta^T(i)\Delta(i)}} \qquad (1)$$

This results in a step of size $C(i)$ in the direction of the tumble for bacteria i.

[substep **f]** Compute $J(i,j+1,k,l)$ with $\theta^i(j+1,k,l)$.

[substep **g]** Swim:

(**i**) Let $m = 0$ (counter for swim length).

(**ii**) While $m < N_s$ (if have not climbed down too long)

• Let $m = m$+1.

• If $J(i,j+1,k,l) < J_{\text{last}}$, let $J_{\text{last}} = J(i,j+1,k,l)$. then anther step of size $C(i)$ in this same direction will be taken as equation (1) and use the new generated $\theta^i(j+1,k,l)$ to compute the new $J(i,j+1,k,l)$.

• Else let m = N_s.

[substep **h]** Go to next bacterium (i+1): if $i \neq S$ go to (b) to process the next bacteria.

[Step 5] If $j < N_c$, go to step 3. In this case, continue chemotaxis since the life of the bacteria is not over.

[Step 6] Reproduction:

[substep **a]** For the given k and l, and for each i = 1, 2, …, S, let J_{health} be the health of the bacteria. Sort bacterium in order of ascending values.

$$J^i_{health} = \sum_{j=1}^{N_c+1} J(i,j,k,l) \qquad (2)$$

[substep **b]** The S_r bacteria with the highest J_{health} values die and the other S_r bacteria with the best values split and the copies that are made are placed at the same location as their parent.

[Step 7] If $k < N_{\text{re}}$ go to step 2. In this case the number of specified reproduction steps is not reached and start the next generation in the chemotactic loop.

[Step 8] Elimination–dispersal: For i = 1, 2, …, S, with probability p_{ed}, eliminate and disperse each bacteria, which results in keeping the number of bacteria in the population constant. To do this, if a bacterium is eliminated, simply disperse one to a random location on the optimization domain. If $l < N_{ed}$, then go to step 2; otherwise end.

2.1. Single bacterial behavior in BFO

In order to get an insight into the behavior of the virtual bacteria in BFO model, we illustrate the bacterial trajectories in a fitness landscape by tuning the run-length unit parameter $C(i)$, which can essentially influence the bacterial behaviors.

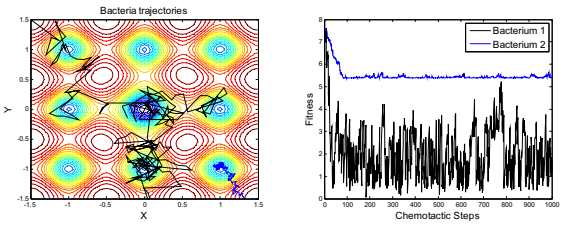
The fitness landscape is defined by the 2-D Ackley function (Equation 3), which has one narrow global optimum basin and many minor local optima. It is a widely used multimodal benchmark with the global optimum (0, 0) and the minimum is 0.

$$f(x,y) = -20\exp\left(-0.2\sqrt{\tfrac{1}{2}(x^2+y^2)}\right) - \exp\left(\tfrac{1}{2}(\cos 2\pi x + \cos 2\pi y)\right) + 20 + e \qquad (3)$$

Fig. 1(a) shows the contour lines of the Ackley function together with the foraging path of two bacteria that simultaneously start at (-1.5, 1.5) and (1.5, -1.5) with different $C(i)$ respectively. The precision goal was set to 0.1. I.e., the parameter setting for BFO is S=2, N_c=1000, N_s=4, N_{re}=1, N_{ed}=1, $C(1)$=0.1 and $C(2)$=0.01. Fig. 1(b) illustrates the fitness evolution process of these two bacteria over 1000 chemotactic steps.

From the chemotactic motions of the two virtual bacteria as in Fig. 1(a), we can observe that the bacterium with larger $C(i)$=0.1 can explore the whole search space and stay for a while in several domains with local optima. It can also escape from these local optima to enter the domain with the global optimum, but was not able to stop there. On the other hand, the bacteria with the smaller $C(i)$=0.01 was attracted into the domain with a local optima, which closed to its start point, and exploited this local minimum with high-precision for its whole life cycle. That is, if the bacteria with small $C(i)$ traps in a local minima, it is not able to escape from it.

Obviously, in the context of original BFO model, the bacteria with large run-length unit have the exploring ability (global investigation of the search place) while



(a) (b)

Fig. 1. Bacteria evolution on Arckley. Bacterium 1: start point (-1.5, 1.5), run-length unit parameter $C(1)$=0.1, chemotactic steps to precise goal (0.1) are 184; Bacterium 2: start point (1.5, -1.5), run-length unit parameter $C(1)$=0.01, can not reach the precise goal.

the bacteria with relatively small run-length unit have the exploiting skill (the fine search around a local optimum). However, it is difficult to decide which values of static run-length unit is best suited for a given problem. Hence, we introduce the preprogrammed change of this parameter during the evolution to balance exploration and exploitation, which results in the significant improvement in performance of the original algorithm.

3. The SA-BFO algorithm

In the SA-BFO algorithm, we introduce an "individual run-length unit" to the i^{th} bacterium of the colony and each bacterium can only modify the search behavior of itself by using the current status of its own. In this way, not only the position but also the run-length unit of each bacterium undergoes evolution. In SA-BFO evolution process, each bacterium displays alternatively two distinct search states:

(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

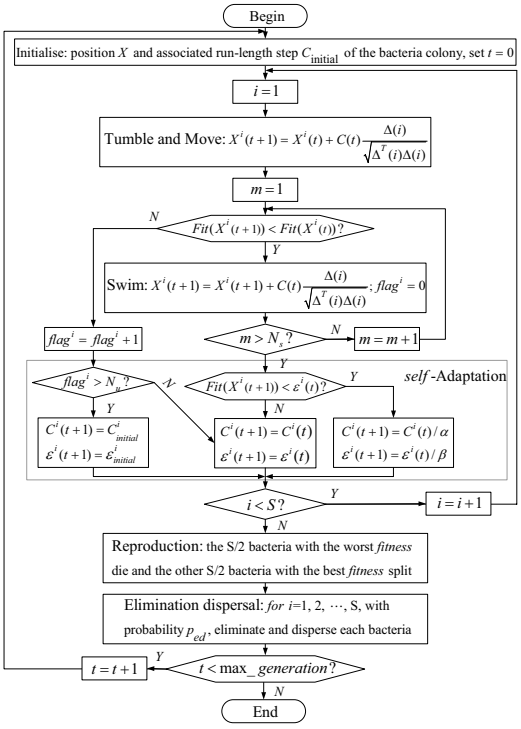


Fig. 2. Flowchart of the SA-BFO algorithm.

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_u (user-defined) of consecutive generations, then augment this bacterium's run-length unit and this bacterium 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.

The flowchart of the SA-BFO algorithm can be illustrated by Fig. 2, where S is the colony size, t is the chemotactic generation counter from 1 to max-generation, i is the bacterium's ID counter from 1 to S, X^i is the i^{th} bacterium's position of the bacteria colony, N_s is the maximum number of steps for a single activity of *Swim*, $flag^i$ is the number of generations the *ith* bacterium has not improved its own fitness, $C^i(t)$ is the current run-length unit of the *ith* bacterium, $\varepsilon^i(t)$ is the required precision in the current generation of the *ith* bacterium, α and β are user-defined constants, $C_{initial}$ and $\varepsilon_{initial}$ are the initialized run-length unit and precision goal respectively. As we can see, we embed the reproduction, elimination and dispersal events in each chemotactic generation. This can improve the

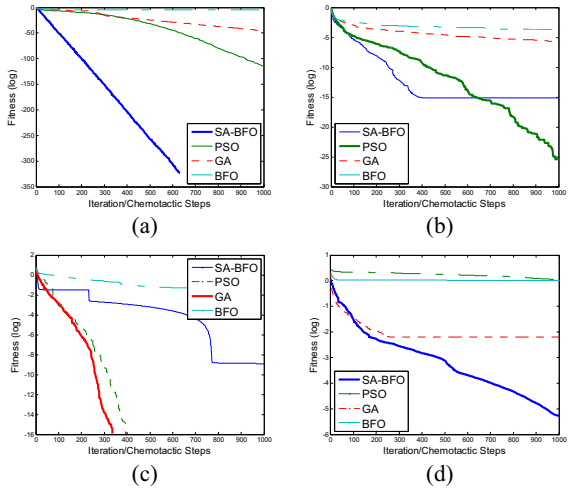


Fig. 3. Convergence results of SA-BFO, BFO, PSO, and GA on 2-D benchmark functions. (a) Sphere function; (b) Rosenbrock function; (c) Rastrigrin function; (d) Griewank function.

algorithm convergence rate significantly. We also simplify the "bacterial health" by the bacterium's current fitness.

4. Experimental study

4.1. Benchmark functions

The set of benchmark functions contains four functions that are commonly used in evolutionary computation literature to show solution quality and convergence rate [7, 8]. The formulas of these functions are listed below.

1. Sphere function

$$f_1(x)=\sum_{i=1}^{n}x_i^2 \qquad x\in[-5.12,5.12]^D \tag{5}$$

2. Rosenbrock function

$$f_2(x)=\sum_{i=1}^{n}100\times(x_{i+1}-x_i^2)^2+(1-x_i)^2 \quad x\in[-3,3]^D \tag{6}$$

3. Rastrigrin function

$$f_3(x)=\sum_{i=1}^{n}(x_i^2-10\cos(2\pi x_i)+10 \quad x\in[-5.12,5.12]^D \tag{7}$$

4. Griewank function

$$f_4(x)=\frac{1}{4000}\sum_{i=1}^{n}x_i^2-\prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}})+1 \quad x\in[-600,600]^D \tag{8}$$

Table 1. Parameters of SA-BFO

Function	S	$C_{initial}$	$\varepsilon_{initial}$	K_u	α	β
Sphere	100	0.1	100	20	10	10
Rosenbrock	100	0.1	100	20	10	10
Rastrigrin	100	0.1	100	20	10	10
Griewank	100	10	100	20	10	10

4.2. Parameter settings for algorithms

Experiment was conducted to compare four algorithms (including the original BFO, the real-coded Genetic Algorithm (GA), the standard Particle Swarm Optimization (PSO) and the proposed SA-BFO on four benchmark functions with two dimensions. The experiments run 30 times respectively for each algorithm on each benchmark function and the maximum generation is set at 1000. The mean values and standard deviation of the results are presented.

The parameters settings for SA-BFO are summarized in Table 1. For the original BFO algorithm, we take C=0.1 and P_{ed}=0.25, which is the standard set of these two parameter values as recommended in [1]. We set S=100, N_c=100, N_s=4, N_{re}=5 and N_{ed}=2, then BFO performs totally 1000 chemotactic steps in each run, which make a fair comparison in regard of the parameter values. The PSO algorithm we used is the standard one and the parameters were given by the default setting of [5]. The GA algorithm we executed is a real-coded Genetic Algorithm with intermediate crossover and Gaussian mutation (the parameters were to be the same of [6]). The population size of all the algorithms was set at 100.

4.3. Simulation results and discussion

The representative results obtained are presented in Table 2, including the best, worst, mean and standarddeviation of the function values found in 30 runs. Fig. 3 present the evolution process for all algorithms according to the reported results in Table 2.

From the results, we observe that SA-BFO achieved significantly better performance on all benchmark functions than the original BFO algorithm. SA-BFO surpasses all other algorithms on function 1, which is the unimodal function that adopted to assess the convergence rates of optimization algorithm. SA-BFO

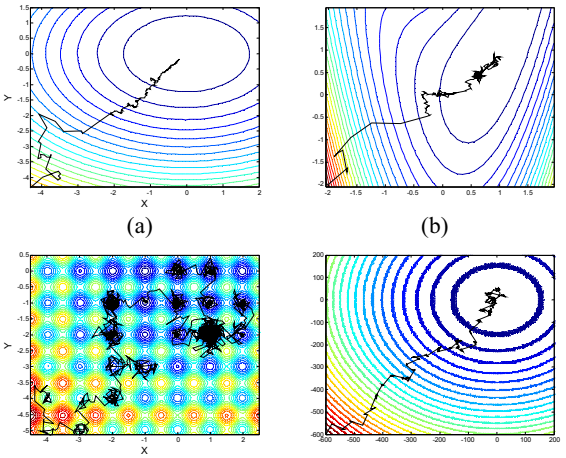


Fig. 4. Single bacterium self-adaptive foraging trajectories of SA-BFO model on 2-D benchmark functions. (a) Sphere function; (b) Rosenbrock function; (c) Rastrigrin function; (d) Griewank function.

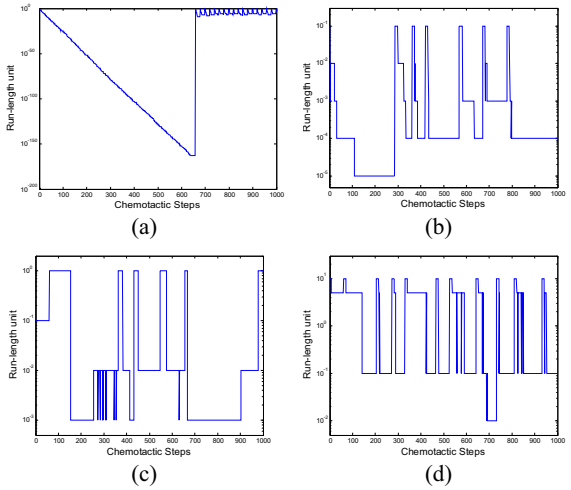


Fig. 5. Dynamics of the run-length unit automatically produced by the self-adaptive criterion during the single bacterium foraging on 2-D benchmark functions. (a) Sphere function; (b) Rosenbrock function; (c) Rastrigrin function; (d) Griewank function.

performs better on multimodal functions 4 when the other algorithms miss the global optimum basin.

In order to further analyze the adaptive foraging behaviors of the proposed SA-BFO model, we simulate the self-adaptive behaviors in SA-BFO to show how the algorithm decides by itself when to explore and when to exploit the searching space. In this simulation, we exclude the reproduction, elimination and dispersion events to illustrate the bacterial behaviors clearly. Fig. 4(a)~(d) illustrate the trajectories of a single bacterium on the four benchmark functions respectively, namely the 2D Dejong, Rosenbrock, Rastrigrin and Griewank function, which are contour plotted. In all the cases, the evolution process proceeds 1000 chemotatic steps, and we choose the same parameters setting as in Table 4, except $S = 1$.

As we can see, our self-adaptive strategy is important because it permits the bacteria refining its foraging behavior adaptively. We can observe that the bacterium switches between exploitation and exploration states by self-adapting its run-length unit. Fig. 9 (a) and (b) illustrate bacterial trajectories in the 2-D unimodal function (Sphere and Rosenbrock). At the beginning of the simulation, the bacterium starts exploring the search space using a large run-length unit. In this way, the bacterium does not waste much time before finding the promising region that contains the global optimum because of the large run-length unit encourages long-range search. Then the bacterium slows down (i.e. the bacterium gradually decreases its run-length unit) near the optimum to pursue the more and more precise solutions. Fig. 9(c) and (d) present the bacterial trajectories in the contour plotted 2-D multimodal function. We can observe that whenever the bacterium encounters a fitness improvement, this forager starts searching intensively in this promising region. Whereas, whenever it is highly probable that the good solutions lying in this region have been found, it move away from this region and start exploring the other regions of the search space until another better region is discovered. Finally, the bacterium finds the domain that contains the global optimum. Clearly, this result captures the important aspects of the self-adaptive behaviors that take place in nature.

In Fig. 5, we also illustrated the associated evolution processes of the run-length unit of this bacterium during foraging. This provides an intuitive explanation of what is happening during the execution of SA-BFO. From Fig. 5, we can notice that when the bacterium finds difficulties during an exploit state (with low run-length unit), then it increases the run-length unit to improve the exploration, and vice versa.

5. Conclusion

In this paper, we firstly analyzed the foraging behavior of the bacteria in the original BFO model. Specifically, we have studied the influence of the run-length unit parameter on the exploration/exploitation tradeoff for the bacterial foraging behaviors. That is, the bacteria with large run-length unit have the exploring ability while the bacteria with relatively

Table 2. Comparison among SA-BFO, BFO, PSO and GA. In bold are the best results.

	2-D	BFO	$ABFO_1$	PSO	GA
f_1	Best	5.6423e-007	**0**	5.0576e-124	2.8638e-048
	Worst	3.3404e-005	**0**	2.4892e-114	1.0587e-045
	Mean	1.4291e-005	**0**	8.5226e-116	2.3105e-046
	Std	9.6035e-006	**0**	4.5408e-115	2.7237e-046
f_2	Best	1.5294e-006	7.8886e-031	**0**	6.8564e-010
	Worst	9.2935e-004	2.1905e-014	5.3075e-025	3.8578e-005
	Mean	1.7353e-004	7.3528e-016	3.9893e-026	1.9021e-006
	Std	2.1344e-004	3.9983e-015	1.1947e-025	7.0743e-006
f_3	Best	9.0825e-004	4.6509e-011	**0**	**0**
	Worst	0.1256	2.7349e-009	**0**	**0**
	Mean	0.0259	1.2655e-009	**0**	**0**
	Std	0.0306	7.4950e-010	**0**	**0**
f_4	Best	0.0296	**0**	0.0311	**0**
	Worst	2.9974	7.7411e-005	3.8791	0.0271
	Mean	0.9975	5.1542e-006	1.1178	0.0063
	Std	0.8142	1.5892e-005	1.0521	0.0061

small run length unit have the exploiting skill. Then, we present the Self-adaptive Bacterial Foraging Optimization algorithm, which cast Bacterial Foraging Optimization into the adaptive fashion by changing the value of the run-length unit during the algorithm execution.

Four widely used benchmark functions have been used to test the SA-BFO algorithm in comparison with the original BFO, the standard PSO and the real-coded GA. The simulation results are encouraging: the SA-BFO is definitely better than the original BFO for all the test functions and appear to be comparable with the standard PSO and GA. The proposed algorithm achieves this by decreasing the run-length unit of each bacterium to encourage exploitation when it enters the promising region with high fitness, while increasing it to improve the exploration when the bacterium finds difficulties during exploitation. These automatic alternate shifting between exploitation and exploration during execution is a noticeable feature of the adaptive mechanisms we proposed, because it shows how the algorithm decides by itself when to explore and when to exploit the searching space.

We also simulated the self-adaptive foraging process of single bacterium based on the proposed model. The results evidently illustrate the dynamics of bacteria foraging that takes place in nature.

There are ways to improve our proposed algorithms. The further research efforts should focus on the tuning of the user-defined parameters for ABFO algorithms based on extensive evaluation on many benchmark functions and real-world problems.

6. References

[1] K. M. Passino, "Biomimicry of bacterial foraging", IEEE Control Systems Magazine, pp. 52–67, June, 2002.

[2] S. Mishra, "A hybrid least square-fuzzy bacterial foraging strategy for harmonic estimation", *IEEE Trans. Evolutionary Computation*, vol. 9, pp. 61-73. 2005.

[3] M. Tripathy, S. Mishra, L. L. Lai, and Q .P. Zhang, "Transmission Loss Reduction Based on FACTS and Bacteria Foraging Algorithm", *in Proc. 9th Int. Conf. on Parallel Problem Solving from Nature*, pp. 222-231, 2006.

[4] D.H. Kim, J.H. Cho, "Adaptive tuning of PID controller for multivariable system using bacterial foraging based optimization", *in Proc. 3rd Int. Atlantic Web Intelligence Conf.*, Lodz, Poland, 2005, pp. 231-238.

[5] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *IEEE Trans. Evol. Comput.*, vol. 6, pp. 58–73, Feb. 2002.

[6] S. Sumathi, T. Hamsapriya, P. Surekha, *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*, Springer, 2008.

[7] Y. Shi, R. C. Eberhart, "Empirical study of particle swarm optimization." *in Proc. of the IEEE Congress on Evolutionary Computation*, Piscataway NJ, pp. 1945-1950, 1999.

[8] H. N. Chen, Y. L. Zhu, "Optimization Based on Symbiotic Multi-Species Coevolution." *Applied Mathematics and Computation*, unpublished.