Hindawi Publishing Corporation
Abstract and Applied Analysis
Volume 2011, Article ID 108269, 27 pages
doi:10.1155/2011/108269

Research Article

Adaptive Bacterial Foraging Optimization

Hanning Chen, Yunlong Zhu, and Kunyuan Hu

Key Laboratory of Industrial Informatics, Shenyang Institute of Automation, Chinese Academy of Sciences, Faculty Office III, Nanta Street 114, Dongling District, Shenyang 110016, China

Correspondence should be addressed to Hanning Chen, perfect_chn@hotmail.com

Received 14 September 2010; Accepted 3 February 2011

Academic Editor: Yoshikazu Giga

Bacterial Foraging Optimization (BFO) is a recently developed nature-inspired optimization algorithm, which is based on the foraging behavior of *E. coli* bacteria. Up to now, BFO has been applied successfully to some engineering problems due to its simplicity and ease of implementation. However, BFO possesses a poor convergence behavior over complex optimization problems as compared to other nature-inspired optimization techniques. This paper first analyzes how the run-length unit parameter of BFO controls the exploration of the whole search space and the exploitation of the promising areas. Then it presents a variation on the original BFO, called the adaptive bacterial foraging optimization (ABFO), employing the adaptive foraging strategies to improve the performance of the original BFO. This improvement is achieved by enabling the bacterial foraging algorithm to adjust the run-length unit parameter dynamically during algorithm execution in order to balance the exploration/exploitation tradeoff. The experiments compare the performance of two versions of ABFO with the original BFO, the standard particle swarm optimization (PSO) and a real-coded genetic algorithm (GA) on four widely-used benchmark functions. The proposed ABFO shows a marked improvement in performance over the original BFO and appears to be comparable with the PSO and GA.

1. Introduction

In the past several decades, research on optimization has attracted more and more attention. The most general unconstrained optimization problem can be defined as

$$\text{Minimize } f(X), \quad X = [x_1, x_2, \ldots, x_D], \tag{1.1}$$

where D is the number of the parameters to be optimized.

There are different optimization methods and algorithms that can be grouped into deterministic and stochastic [1, 2]. Deterministic techniques depend on the mathematical

nature of the problem. Weaknesses of this technique are dependent on gradient, local optimums, and inefficiency in large-scale search space. Stochastic techniques are considered to be more user friendly because they do not depend on the mathematical properties of a given function and are hence more appropriate for finding the global optimal solutions for any type of objective function. As many real-world optimization problems become increasingly complex, using stochastic methods is inevitable.

Nature ecosystems have always been the rich source of mechanisms for designing artificial computational systems to solve difficult engineering and computer science problems. In the optimization domain, researchers have been inspired by biological processes to develop some effective stochastic techniques that mimic the specific structures or behaviors of certain creatures. For examples, genetic algorithms (GA), originally conceived by Holland [3], represent a fairly abstract model of Darwinian evolution and biological genetics; ant colony optimization (ACO), proposed by Dorigo et al. [4, 5], is developed based on the foraging behaviors of real ant colonies; particle swarm optimization (PSO), proposed by Eberchart and Kennedy [6] and Chen et al. [7], glean ideas from social behavior of bird flocking and fish schooling. These algorithms have been found to perform better than the classical heuristic or gradient-based methods, especially for optimizing the nondifferentiable, multimodal, and discrete complex functions. Currently, these nature-inspired paradigms have already come to be widely used in many areas.

In recent years, chemotaxis (i.e., the bacterial foraging behavior) as a rich source of potential engineering applications and computational model has attracted more and more attention. A few models have been developed to mimic bacterial foraging behavior and have been applied for solving some practical problems [8–10]. Among them, bacterial foraging optimization (BFO) is a population-based numerical optimization algorithm presented by Passino [10]. BFO is a simple but powerful optimization tool that mimics the foraging behavior of *E. coli* bacteria. Until now, BFO has been applied successfully to some engineering problems, such as optimal control [11], harmonic estimation [12], transmission loss reduction, [13] and machine learning [14]. However, experiments with complex and multimodal benchmark functions reveal that the original BFO algorithm possesses a poor convergence behavior, and its search performance heavily decreases with the growth of search space dimensionality and the problem complexity.

Several BFO variants have been developed to improve its optimization performance. In [13], Tripathy et al. proposed an improved BFO algorithm using two approaches: (1) in order to speed up the convergence, the average value is replaced by the minimum value of all the chemotactic cost functions for deciding the bacterium's health; (2) for swarming, the distances of all the bacteria in a new chemotactic step are evaluated from the globally optimal bacterium to these points and not the distances of each bacterium from the rest of the others. Mishra [12] proposed a fuzzy bacterial foraging (FBF) algorithm using Takagi-Sugeno-type fuzzy inference scheme to select the optimal chemotactic step size in BFO. Kim et al. proposed a hybrid approach involving GA and BFO for function optimization [15]. Biswas et al. proposed a synergism of BFOA with the particle swarm optimization [16]. The new algorithm, named by the authors as bacterial swarm optimization (BSO), was shown to perform in a statistically better way as compared to both of its classical counterparts over several numerical benchmarks. Two cooperative variants of BFO, namely, the cooperative bacterial foraging optimizer (CBFO) and multicolony bacterial foraging optimization (MCBFO), have been described in our previous works [17, 18]. In [17], instead of simply considering chemotaxis behavior of single bacterium, MCBFO also introduced the communication strategies employed by natural multicolony bacterial community in

order to coordinate pattern emerges. In [18], the proposed CBFO applied two cooperative approaches to the original BFO, namely, the serial heterogeneous cooperation on the implicit space decomposition level and the serial heterogeneous cooperation on the hybrid space decomposition level.

In order to improve the BFO's performance on complex optimization problems with high dimensionality, we apply two natural foraging strategies, namely, the producer-scrounger foraging (PSF) and the area concentrated search (ACS), to the original BFO, resulting in two new adaptive bacterial foraging optimization models (ABFOs), namely, $ABFO_1$ and $ABFO_2$. Instead of the simple description of chemotactic behavior in BFO, the proposed algorithms can also adaptively strike a balance between the exploration and the exploitation of the search space during the bacteria evolution, by which the significant improvement can be gained. In order to evaluate the performance of the proposed algorithms, extensive studies based on a set of 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. The simulation results are encouraging. The ABFO algorithms have markedly superior search performance when compared to the original BFO, while maintaining the similar or even superior performance compared to PSO and GA in terms of accuracy, robustness, and convergence speed on all benchmark functions. The proposed $ABFO_1$ and $ABFO_2$ described in this paper enhance previous BFO works in the following aspects:

(i) a new adaptive strategy, namely, the producer-scrounger foraging, to dynamically determine the chemotactic step sizes for the whole bacterial colony during a run, hence dividing the foraging procedure of artificial bacteria colony into multiple explore and exploit phases;

(ii) a new self-adaptive foraging strategy, namely, the area concentrate search, to respectively tune the chemotactic step size for each single bacterium during its run, hence casting the bacterial foraging process into heterogeneous fashion;

(iii) a comprehensive study comparing $ABFO_1$ and $ABFO_2$ with another two state-of-the-art global optimization algorithms, namely, GA and PSO, on high dimensional functions;

(iv) single and colonial bacterial behaviors in both $ABFO_1$ and $ABFO_2$ that were simulated respectively in order to analyze in depth the adaptive and self-adaptive foraging schemes in the proposed models;

(v) new results on benchmark functions up to 300 dimensions.

The rest of the paper is organized as follows. In Section 2, we will give brief reviews of the bacterial chemotaxis and the original BFO algorithm. The in-depth analysis of the influence of the run-length unit parameter on the bacterial behavior in BFO model is also presented here. In Section 3, two adaptive strategies employed by ABFOs are described and the state-of-the-art adaptation mechanism is summarized. Then, our adaptive bacterial optimization algorithms will be introduced, and its implementation details will be given in Section 4. In Section 5, the experiment studies of the proposed ABFO algorithms and other algorithms are presented with descriptions of the benchmark functions, experimental settings, and experimental results. Finally, Section 6 concludes the paper.

2. The Classical BFO Algorithm

Bacterial foraging algorithm is inspired by an activity called "chemotaxis" exhibited by bacterial foraging behaviors. Motile bacteria such as *E. coli* and *salmonella* propel themselves by rotation of the flagella. To move forward, the flagella rotates counterclockwise and the organism "swims" (or "runs") while a clockwise rotation of the flagellum causes the bacterium to randomly "tumble" itself in a new direction and swim again [19]. Alternation between "swim" and "tumble" enables the bacterium to search for nutrients in random directions. Swimming is more frequent as the bacterium approaches a nutrient gradient. Tumbling, hence direction changes, is more frequent as the bacterium moves away from some food to search for more. Basically, bacterial chemotaxis is a complex combination of swimming and tumbling that keeps bacteria in places of higher concentrations of nutrients.

2.1. Bacterial Foraging Optimization

The classical bacterial foraging optimization (BFO) system consists of three principal mechanisms, namely, chemotaxis, reproduction, and elimination-dispersal [1]. We briefly describe each of these processes as follows.

2.1.1. Chemotaxis

In the classical BFO, a unit walk with random direction represents a "tumble" and a unit walk with the same direction in the last step indicates a "run". Suppose $\theta^i(j,k,l)$ represents the bacterium at jth chemotactic, kth reproductive, and lth elimination-dispersal step. $C(i)$, namely, the run-length unit parameter, is the chemotactic step size during each run or tumble. Then, in each computational chemotactic step, the movement of the ith bacterium can be represented as

$$\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i)\frac{\Delta(i)}{\sqrt{\Delta^T(i)\Delta(i)}}, \tag{2.1}$$

where $\Delta(i)$ is the direction vector of the jth chemotactic step. When the bacterial movement is *run*, $\Delta(i)$ is the same with the last chemotactic step; otherwise, $\Delta(i)$ is a random vector whose elements lie in $[-1, 1]$.

With the activity of run or tumble taken at each step of the chemotaxis process, a step fitness, denoted as $J(i,j,k,l)$, will be evaluated.

2.1.2. Reproduction

The health status of each bacterium is calculated as the sum of the step fitness during its life, namely, $\sum_{j=1}^{N_c} J(i,j,k,l)$, where N_c is the maximum step in a chemotaxis process. All bacteria are sorted in reverse order according to health status. In the reproduction step, only the first half of population survives, and a surviving bacterium splits into two identical ones, which are then placed in the same locations. Thus, the population of bacteria keeps constant.

2.1.3. Elimination and Dispersal

The chemotaxis provides a basis for local search, and the reproduction process speeds up the convergence which has been simulated by the classical BFO, while, to a large extent, only chemotaxis and reproduction are not enough for global optima searching. Since bacteria may get stuck around the initial positions or local optima, it is possible for the diversity of BFO to change either gradually or suddenly to eliminate the accidents of being trapped into the local optima. In BFO, the dispersion event happens after a certain number of reproduction processes. Then, some bacteria are chosen, according to a preset probability P_{ed}, to be killed and moved to another position within the environment.

2.2. Step-by-Step Algorithm

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

Step 1. Initialize parameters $n, S, N_c, N_s, N_{\mathrm{re}}, N_{\mathrm{ed}}, P_{\mathrm{ed}}, C(i)$ $(i = 1, 2, \ldots, S), \theta^i$, 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 (i.e., the chemotactic step size 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 4.1. For $i = 1 = 1, 2, \ldots, S$, take a chemotactic step for bacteria i as follows.

Substep 4.2. Compute fitness function, $J(i, j, k, l)$.

Substep 4.3. Let $J_{\mathrm{last}} = J(i, j, k, l)$ to save this value since we may find better value via a run.

Substep 4.4. Tumble: generate a random vector $\Delta(i) \in R^n$ with each element $\Delta_m(i)$, $m = 1, 2, \ldots, S$, a random number on $[-1, 1]$.

Substep 4.5. Move: let (2.1). This results in a step of size $C(i)$ in the direction of the tumble for bacteria i.

Substep 4.6. Compute $J(i, j + 1, k, l)$ with $\theta^i(j + 1, k, l)$.

Substep 4.7. Swim:

(i) let $m = 0$ (counter for swim length)

(ii) while $m < N_s$ (if not climbed down too long)

(a) let $m = m + 1$

(b) if $J(i, j+1, k, l) < J_{\text{last}}$, let $J_{\text{last}} = J(i, j+1, k, l)$, Then, another step of size $C(i)$ in this same direction will be taken as (2.1) and use the new generated $\theta^i(j+1, k, l)$ to compute the new $J(i, j+1, k, l)$

(c) else let $m = N_s$.

Substep 4.8. 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 6.1. For the given k and l, and for each $i = 1, 2, \ldots, S$, let

$$J^i_{\text{health}} = \sum_{j=1}^{N_c+1} J(i, j, k, l) \tag{2.2}$$

be the health of the bacteria. Sort bacterium in order of ascending values (J_{health}).

Substep 6.2. 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; start the next generation in the chemotactic loop.

Step 8. Elimination-dispersal: for $i = 1, 2, \ldots, S$, 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 $l < N_{\text{ed}}$, then go to Step 2, otherwise, end.

2.3. 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 two separate environments (a unimodal and a multimodal case) by tuning the run-length unit parameter $C(i)$, which can essentially influence the bacterial behaviors.

The first case is the minimization of the 2-d sphere function, which is a widely used unimodal benchmark with a single optimum $(0, 0)$ and the minimum is 0

$$f_a(x, y) = x^2 + y^2. \tag{2.3}$$

Figure 1(a) illustrates the trajectories of two bacteria foraging for the optimum in the search space defined by the sphere function, which is contour plotted. The two bacteria simultaneously start at (−4, −4) and (4, 4) with different run-length unit. The simulation takes 1000 chemotactic steps (note that the reproduction and elimination-dispersal events are not considered here) while only the steps before the bacteria find the function fitness of 0.1 are plotted. That is, the parameter setting for BFO is $S = 2$, $N_c = 1000$, $N_s = 4$, $N_{\text{re}} = 1$, $N_{\text{ed}} = 1$, $C(1) = 0.1$, and $C(2) = 0.01$. Figure 1(b) shows the converging process of the function values found by these two bacteria over 1000 steps.

From the chemotactic motions of two virtual bacteria (represented as the black and blue trajectories, respectively, in Figure 1), we can observe that both bacteria can travel up the gradient to pursue the optimum of sphere function. We can also observe from Figure 1(a) that the larger the parameter $C(i)$, the smaller the number of steps to reach the goal, although the path seems to be less direct sometimes. However, from Figure 1(b), we can observe that bacteria with the smaller $C(i)$ obtain the better performance than the other one with the larger $C(i)$. That is, if the $C(i)$ is larger than the precision goal, the bacterium is not able to stop at the minimum.

The other simulation case is on the 2-d Ackley function, 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,

$$f_b(x,y) = -20\exp\left(-0.2\sqrt{\frac{1}{2}\left(x^2+y^2\right)}\right) - \exp\left(\frac{1}{2}\left(\cos 2\pi x + \cos 2\pi y\right)\right) + 20 + e. \tag{2.4}$$

Figure 2(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 $C(i)$. The precision goal was set to 0.1. That is, the parameter setting for BFO is $S = 2$, $N_c = 1000$, $N_s = 4$, $N_{\text{re}} = 1$, $N_{\text{ed}} = 1$, $C(1) = 0.1$, and $C(2) = 0.01$. Figure 2(b) illustrates the foraging process of the function values found by these two bacteria over 1000 chemotactic steps.

From the chemotactic motions of the two virtual bacteria as in Figure 2(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 containing local optima. It can also escape from these local optima to enter the domain with the global optimum, but it 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 for its whole life cycle. That is, if the bacteria with small $C(i)$ is trapped 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 exploring ability (i.e., global investigation of the search place) while the bacteria with relatively small run-length unit have exploiting skill (i.e., 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 significant improvement in performance of the original algorithm.

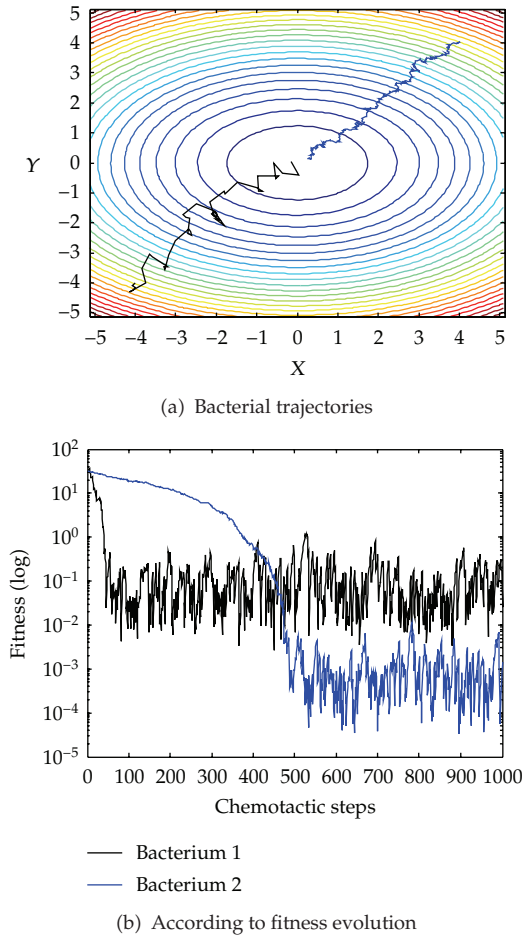
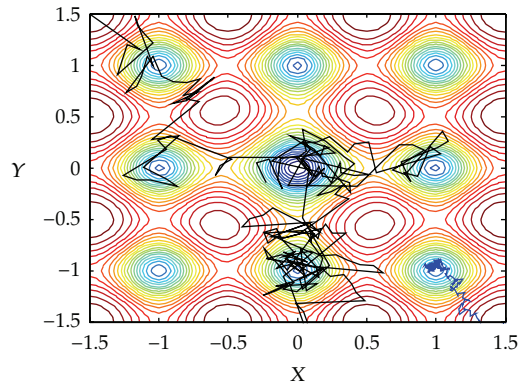


Figure 1: Bacteria evolution on f_a. Bacterium 1: start point (−4, −4), run-length unit parameter $C(1) = 0.1$, chemotactic steps to precise goal (0.1) are 46. Bacterium 2: start point (4, 4), run-length unit parameter $C(1) = 0.01$, chemotactic steps to precise goal (0.1) are 453.

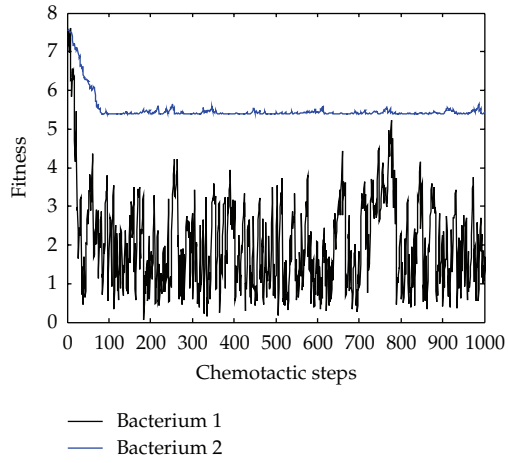
3. Adaptive Strategies and Algorithms

The balance between exploration of the search space and exploitation of potentially good solutions is considered as a fundamental problem in nature-inspired systems. Too much stress on exploration results in a pure random search whereas too much exploitation results in a pure local search. Clearly, intelligent search must self-adaptively combine exploration of the new regions of the space with evaluation of potential solutions already identified.

The most important contribution of the present paper is to define the artificial bacteria in the ABFO models that are capable of self-adapting their exploration and exploitation behaviors in the foraging process. In this section, we will first introduce two adaptive foraging strategies, namely, producer-scrounger foraging and area concentrated search, from which the proposed two ABFO algorithms glean ideas. Secondly, the taxonomy of adaptation methods in natural-inspired algorithms will be given to guide the readers in classifying the kind of algorithms we are dealing with in this work.



(a) Bacterial trajectories



(b) According to fitness evolution

Figure 2: Bacteria evolution on f_b. 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 $C(1) = 0.01$, cannot reach the precise goal.

3.1. The Producer-Scrounger Foraging

In nature, group foraging is a widespread phenomenon since individuals can join groups to improve their foraging success (i.e., the average feeding rate can be increased and its variance can be reduced [20]) by sharing information with each other. Living in groups allows individuals to allocate foraging effort between two different roles, namely, the producer and the scrounger. The "producer" can be used to locate food patches independently while the "scrounger" can be used to exploit the food discovered by other group members [21]. Producer-scrounger models suggest equilibrium and flexibility in use between the two strategies in response to changes in environment, which alters the costs and benefits of producing and scrounging.

3.2. Area Concentrated Search

A central problem for the natural predators in the foraging process is how to balance two conflicting alternatives: the exploitation (to search thoroughly in promising areas)

and the exploration (to move to distant areas potentially better than the actual one). Through natural selection, some predatory animals have developed the area concentrated search (ACS, also called area-restricted search) strategy, by which a predator is able to respond to variations in prey distributions by varying its searching efforts: following an encounter with food resource, a forager searches intensively in a more circumscribed region while a failure to encounter a resource leads to a more extensive, less circumspect mode of search. That is, ACS assumes that regions dense with preys should be exploited slowly, to maximize the chances of encounter, and less dense regions explored rapidly, to minimize the time spent searching in unprofitable areas [22]. ACS in a continuous patchy environment thereby ensures that foraging behaviors will, to some extent, match the distribution of resource, and may be viewed as a simple form of habitat selection [23]. The ACS behavior is regarded as an efficiently adaptive search strategy [24], which is employed by many search-intensive predators, such as birds, lizards, insects, and other higher organisms.

3.3. Taxonomy of Adaptation

In this section, we review the well-known classification of adaptive evolutionary algorithms [25, 26], which can also be used to classify other adaptive nature-inspired algorithms [27].

First, the classification of the type of adaptation can be made on the adaptive mechanism used in the algorithmic process. In particular, attention is paid to the issue of whether the feedback is used or not while the algorithm is searching for the solution to the problem.

In [21], the algorithms are first divided into static and dynamic algorithms. The term "static" means no changes of the parameters occur throughout the run of the algorithm. The term "dynamic adaptation" is used to classify any algorithm where the strategy parameters are modified according to some rules without external control. Based on the mechanism of adaptation, three subclasses of dynamic adaptation are distinguished: deterministic, adaptive, and self-adaptive algorithms.

(i) A deterministic adaptation takes place if the strategy parameter is changed according to some deterministic rules, which modify the value of the parameter without taking into account any feedback from the algorithm itself.

(ii) An adaptive dynamic adaptation takes feedback from the algorithm itself into account and changes the strategies parameters accordingly.

(iii) A self-adaptive dynamic adaptation can be used to implement the self-adaptation of parameters. In this case, the parameters to be adapted are encoded into the representation of the individual.

Furthermore, these dynamic algorithms can be distinguished among different families attending to the level at which the adaptive parameters operate: environment (adaptation of the individual as a response to changes in the environment, e.g., penalty terms in the fitness function), population (parameters global to the entire population), individual (parameters held within an individual), and component (strategy parameters local to some component or gene of the individual). These levels of adaptation can be used with each of the types of dynamic adaptation.

According to this taxonomy on adaptation, this work proposes two variants of adaptive bacterial foraging optimization algorithms, namely, $ABFO_0$ and $ABFO_1$, which can

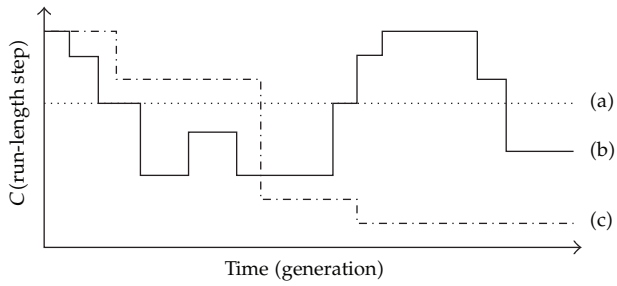


Figure 3: Idealized evolution of the parameter $C(i)$ for (a) nonadaptation, (b) dynamic self-adaptation, and (c) dynamic adaptation.

be classified into the adaptive dynamic adaptation on the population level and the self-adaptive dynamic adaptation on the individual level, respectively.

4. Description of ABFO Algorithms

In this section, we consider two variants of ABFO aimed at different types of adaptation, namely, dynamic adaptation and self-adaptation. We should note that the basic philosophy behind these two ABFO variants is the producer-scrounger foraging and the area concentrated Search theory, respectively. The bacteria should always pursue the appropriate balance between exploration and exploitation of the foraging process in the search space, and this should be done by dynamically controlling parameters (i.e., the bacterial run-length unit—$C(i)$) that takes into account the current status of search (i.e., the quality of the solutions). Such adaptation mechanism enables the ABFO algorithms to cope with complex, highly multimodal search landscapes efficiently. For comparison purposes, in Figure 3, we illustrate a theoretical idealization of the evolution of the parameter $C(i)$ in three different classes of algorithms, namely, the original BFO (nonadaptive), ABFO_0 (adaptive) and ABFO_1 (self-adaptive).

4.1. The ABFO_0 Algorithm (Adaptive Version)

We now present the dynamic adaptive version of ABFO—the ABFO_0 algorithm. As indicated in Section 2, the bacterium with a small run-length unit has the exploring ability while the bacterium with a relatively large run-length unit has the exploiting skill. This inspired us to divide the foraging procedure of artificial bacteria colony into multiple explore/exploit phases and each phase is characterized by the different value of run-length unit and occupies a portion of generations. This approach produces two classes of bacterial individuals, namely, the producers and scroungers, depending on the particular run-length unit that they used. The bacterial producers explore the search space and have the responsibility to find the promising domains and to leave the local optima that have been visited while the bacterial scroungers focus on the precision of the found solutions. That is, the bacteria perform exploitation of the neighborhood of the best-so-far solutions found by the producers. This strategy encompasses the following features.

```
(1) IF (t mod n = 0) then
(2)    IF (f_best < ε(t)) then
(3)       C(t + 1) = C(t − n)/α;
(4)       ε(t + 1) = ε(t)/β;
(5)    ELSE
(6)       C(t + 1) = C(t − n);
(7)       ε(t + 1) = ε(t − n);
(8)    END IF
(9) ELSE
(10)   C(t + 1) = C(t);
(11)   ε(t + 1) = ε(t);
(12) END IF
```

Pseudocode 1: Pseudocode of the dynamic adaptive strategy.

In the initial phase, the bacteria colony searches the whole space of the problem with a large run-length unit C_{initial} (here, the same run-length unit is used for all bacteria in the colony), which permits all the bacterial producers to explore the whole space efficiently and avoid being trapped in local optima. Each bacterial producer records all its visited points, and the point with the highest fitness value is considered as potential solution candidate, which are supplied as an input to the next phase. When entering into the next phase, the bacteria colony is reinitialized with a relatively smaller run-length unit from the potential candidates found in the previous phase. That is, the bacterial scroungers join resources uncovered by the producers. Then, they start exploiting the neighborhood of these current best positions until the needed criteria (i.e., the feedback from the search process) for switching to the next phase is reached. That is, in each phase (except the initial phase), the newly initialized bacterial scroungers will join the resources uncovered by others in the last forgoing phase and then exploit them for more precise solutions.

This dynamic adaptive strategy is given in Pseudocode 1. Where t is the current generation number, f_{best} is the best fitness value among all the bacteria in the colony, $\varepsilon(t)$ is the required precision in the current generation, and n, α, and β are user-defined constants. Using this strategy, changes in the parameter values of C are now based on feedback from the search, and the adaptation happens every n generations. Here, in order to perform fine-tuning exploitation of the global optimum, the run-length unit $C(t)$ adaptively decreases form phase to phase, as shown in Figure 3(c).

The flowchart of the ABFO$_0$ algorithm can be illustrated by Figure 4(a), 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 ith bacterium's position of the bacteria colony, and N_s is the maximum number of steps for a single activity of *Swim*. It should be noted that we embed the reproduction, elimination, and dispersal events in each chemotactic generation. This can improve the algorithm convergence rate significantly. We also simplify the "bacterial health" by the bacterium's current fitness.

4.2. The ABFO$_1$ Algorithm (Self-Adaptive Version)

In the ABFO$_1$ algorithm, we introduce an "individual run-length unit" to the ith bacterium of the colony and each bacterium can only modify the search behaviour of itself by using the current status of its own. In this way, not only the position (solution vector), but also the run-length unit of each bacterium undergoes evolution, respectively. This individual-level self-adaptation may provide us with more accurate information about the search because it is the separate individuals that engaged in searching the solution space not just the whole colony.

The proposed ABFO$_1$ is inspired by ACS strategy described in Section 3.2. In the foraging process of ABFO$_1$ model, 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 has to permanently maintain an appropriate balance between *exploration* and *exploitation* states by varying its own run-length unit adaptively. In ABFO$_1$, the adaptation of the individual run-length unit is done by taking into account two decision indicators: a fitness improvement (finding a promising domain) and no improvement registered lately (current domain is food exhausted). The criteria that determine the adjustment of individual run-length unit and the entrance into one of the states (i.e., exploitation and exploration) are the following.

(i) *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 registers a fitness improvement beyond a certain precision from the last generation to the current. Following *Criterion-1*, the bacterium's behaviour will self-adapt into exploitation state.

(ii) *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 an unpromising domain or the domain where this bacterium focuses its search has nothing new to offer.

This self-adaptive strategy is given in Pseudocode 2, where t is the current generation number, $C^i(t)$ 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, and C_{initial} and $\varepsilon_{\text{initial}}$ are the initialized run-length unit and precision goal, respectively.

The flowchart of the ABFO$_1$ algorithm can be illustrated by Figure 4(b), 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 ith bacterium's position of the bacteria colony, N_s is the maximum number of steps for a single activity of *Swim*, and $flag^i$ is the number of generations the ith bacterium has not improved its own fitness.

```
(1) FOR (each bacterium i) IN PARALLEL
(2)   IF (Criterion-1) then
(3)     C^i(t + 1) = C^i(t)/α;    //exploit
(4)     ε^i(t + 1) = ε^i/β;
(5)   ELSE IF (Criterion-2) then
(6)     C^i(t + 1) = C_initial;    //explore
(7)     ε^i(t + 1) = ε_initial;
(8)   ELSE
(9)     C^i(t + 1) = C^i(t);
(10)    ε^i(t + 1) = ε^i(t);
(11)  END IF
(12) END FOR IN PARALLEL
```

Pseudocode 2: Pseudocode of the dynamic self-adaptive strategy.

5. Simulation Results

5.1. Benchmark Functions

The set of benchmark functions contains four functions that are commonly used in evolutionary computation literature [28, 29] to show solution quality and convergence rate. The formulas and the properties of these functions are listed below.

(1) Sphere function

$$f_1(x) = \sum_{i=1}^{n} x_i^2. \tag{5.1}$$

(2) Rosenbrock function

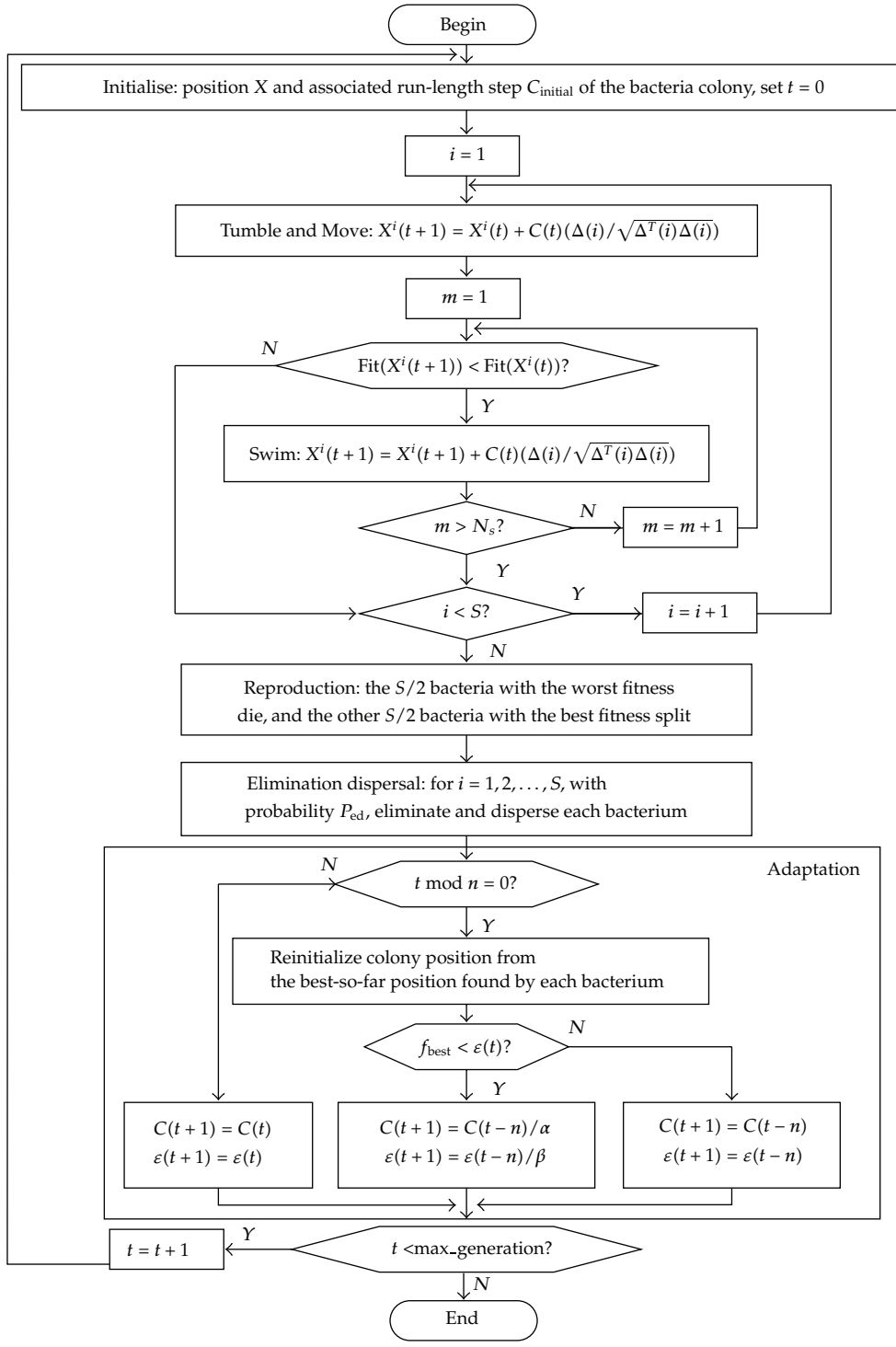
$$f_2(x) = \sum_{i=1}^{n} 100 \times \left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2. \tag{5.2}$$

(3) Rastrigin function

$$f_3(x) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right). \tag{5.3}$$

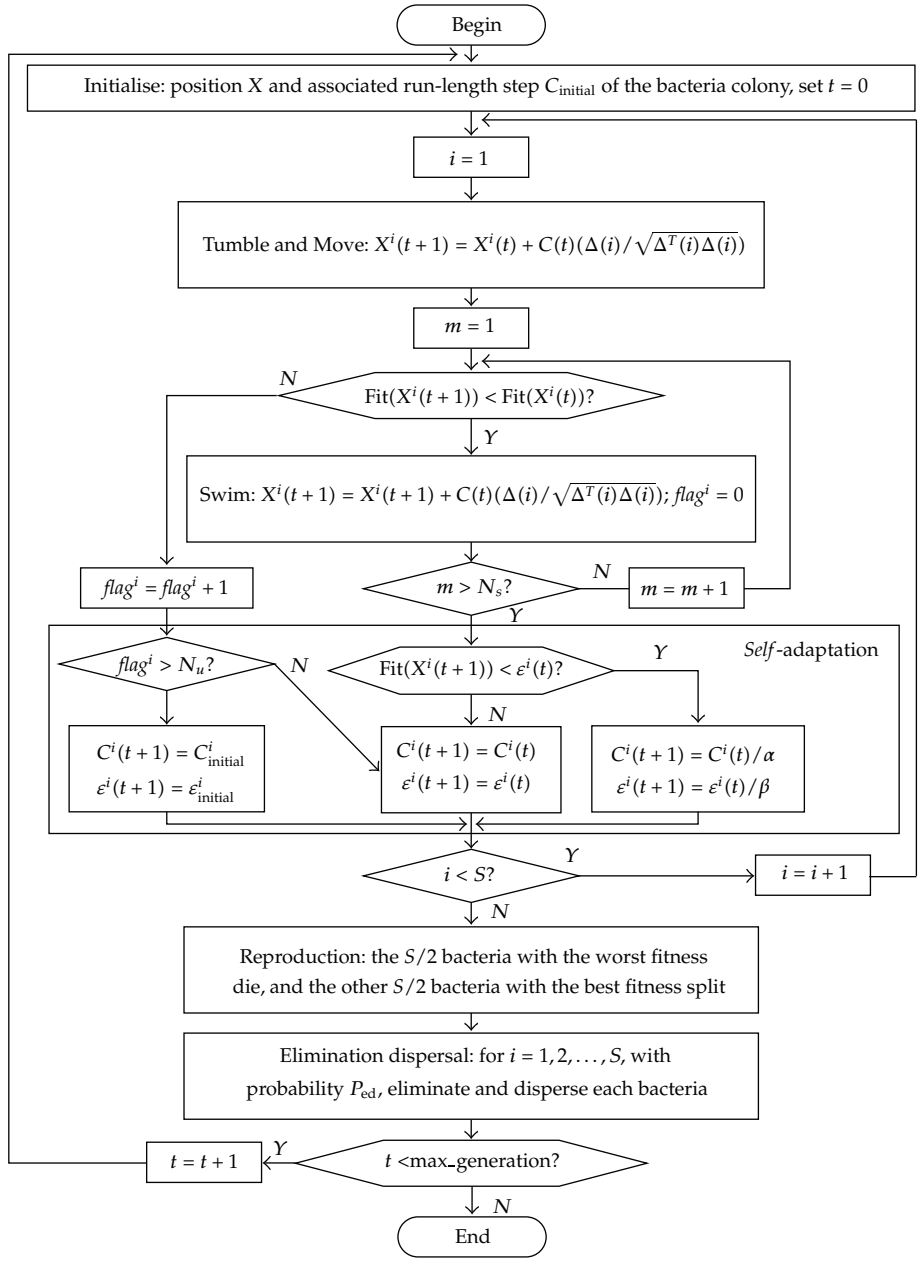
(4) Griewank function

$$f_4(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1. \tag{5.4}$$



(a) ABFO_0

Figure 4: Continued.



(b) ABFO_1

Figure 4: Flowcharts of the proposed algorithms.

The first problem is the sphere function, which is a widely used unimodal benchmark and is easy to solve. The second problem is the Rosenbrock function. It can be treated as a multimodal problem. It has a narrow valley from the perceived local optima to the global optimum. The third problem, namely, the Rastrigin function, is a complex multimodal problem with a large number of local optima. Algorithms may easily fall into a local

Table 1: Parameters of the test functions.

Function	R	X^*	$f(X^*)$
Sphere	$[-5.12, 5.12]^D$	$[0, 0 \cdots 0]$	0
Rosenbrock	$[-2.048, 2.048]^D$	$[1, 1 \cdots 1]$	0
Rastrigin	$[-5.12, 5.12]^D$	$[0, 0 \cdots 0]$	0
Griewank	$[-600, 600]^D$	$[0, 0 \cdots 0]$	0

Table 2: Parameters of the BFO algorithms.

	$ABFO_0$						$ABFO_1$					
Function	S	C_{initial}	$\varepsilon_{\text{initial}}$	n	α	β	S	C_{initial}	$\varepsilon_{\text{initial}}$	K_u	α	β
Sphere	100	0.1	100	10	10	10	100	0.1	100	20	10	10
Rosenbrock	100	0.1	100	20	10	10	100	0.1	100	20	10	10
Rastrigin	100	0.1	100	10	10	10	100	0.1	100	20	10	10
Griewank	100	10	100	200	10	10	100	10	100	20	10	10

optimum when attempting to solve it. The last problem is the multimodal Griewank function. Griewank has linkage among variables that makes it difficult to reach the global optimum. The interesting phenomenon of the Griewank is that it is more difficult for lower dimensions than higher dimensions. All functions are tested on 2 and 10 dimensions. The search ranges R, the global optimum X^*, and the corresponding fitness $f(X^*)$ value of each function are listed in Table 1.

5.2. Parameter Settings for the Involved Algorithms

An experiment was conducted to compare the five algorithms, including the original BFO, the real-coded genetic algorithm (GA), the standard particle swarm optimization (PSO) and the proposed $ABFO_0$ and $ABFO_1$, on the four benchmark functions with two dimensions and ten dimensions. The experiments were run 25 times respectively for each algorithm on each benchmark function and the maximum generation was set at 1000. The mean values and standard deviation of the results are presented.

The parameters settings for $ABFO_0$ and $ABFO_1$ are summarized in Table 2. It should be noted that both the parameters α and β are the run-length unit decreasing parameters. In this work, after performing a series of hand-tuning experiments, we found that keeping the run-length unit decreasing parameter $\alpha = \beta = 10$ provide considerably good performance of convergence and accuracy over all benchmark functions considered here. For the original BFO algorithm, we take $C = 0.1$ and $P_{\text{ed}} = 0.25$, which is the standard set of these two parameter values as recommended in [10]. We set $S = 100$, $N_c = 100$, $N_s = 4$, $N_{\text{re}} = 5$, and $N_{\text{ed}} = 2$, 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 (i.e., the global version with inertia weight). The parameters were given by the default setting of [28]: the acceleration factors c_1 and c_2 were both 2.0; a decaying inertia weight ω starting at 0.9 and ending at 0.4 was used. The population size was set at 100 for the PSO algorithm. The GA algorithm we executed is a real-coded genetic algorithm with intermediate crossover and Gaussian mutation. The population of the GA is 100 and all

Table 3: Comparison among $ABFO_0$, $ABFO_1$, BFO, PSO, and GA on 2-D problems.

	2-D	BFO	$ABFO_0$	$ABFO_1$	PSO	GA
	Best	5.6423*e*–007	4.0368*e*–133	**0**	5.0576*e*–124	2.8638*e*–048
	Worst	3.3404*e*–005	3.8546*e*–111	**0**	2.4892*e*–114	1.0587*e*–045
f_1	Mean	1.4291*e*–005	1.3068*e*–112	**0**	8.5226*e*–116	2.3105*e*–046
	Std	9.6035*e*–006	7.0343*e*–112	**0**	4.5408*e*–115	2.7237*e*–046
	Rank	5	3	**1**	2	4
	Best	1.5294*e*–006	**0**	7.8886*e*–031	**0**	6.8564*e*–010
	Worst	9.2935*e*–004	**1.3078*e*–029**	2.1905*e*–014	5.3075*e*–025	3.8578*e*–005
f_2	Mean	1.7353*e*–004	**8.1803*e*–031**	7.3528*e*–016	3.9893*e*–026	1.9021*e*–006
	Std	2.1344*e*–004	**2.6648*e*–030**	3.9983*e*–015	1.1947*e*–025	7.0743*e*–006
	Rank	5	**1**	3	2	4
	Best	9.0825*e*–004	**0**	4.6509*e*–011	**0**	**0**
	Worst	0.1256	**0**	2.7349*e*–009	**0**	**0**
f_3	Mean	0.0259	**0**	1.2655*e*–009	**0**	**0**
	Std	0.0306	**0**	7.4950*e*–010	**0**	**0**
	Rank	5	**1**	4	1	1
	Best	0.0296	1.5306*e*–012	**0**	0.0311	**0**
	Worst	2.9974	**1.4551*e*–009**	7.7411*e*–005	3.8791	0.0271
f_4	Mean	0.9975	**5.2282*e*–010**	5.1542*e*–006	1.1178	0.0063
	Std	0.8142	**4.4791*e*–010**	1.5892*e*–005	1.0521	0.0061
	Rank	4	**1**	2	5	3
	Average rank	4.75	**1.5**	2.5	2.5	3
	Final rank	5	**1**	2	2	3

the control parameters, for example, mutation rate and crossover rate, and so forth, were set to be the same as [30].

5.3. Results for the 2-D Problems

Table 3 presents the means and variances of the 30 runs of the five algorithms on the four test functions with $D = 2$. The best results among the five algorithms are shown in bold. Figure 5 presents the convergence characteristics in terms of the best fitness value of the median run of each algorithm for each test function.

From the results, we observe that our two adaptive BFO algorithms achieved significantly better performance on all benchmark functions than the original BFO algorithm. $ABFO_1$ surpasses all other algorithms on function 1, which is the unimodal function that adopted to assess the convergence rates of optimization algorithms. The $ABFO_0$ performs better on all multimodal functions 2, 3, and 4 when the other algorithms miss the global optimum basin. The Griewank function is a good example. $ABFO_0$ successfully avoids falling into local optima and continues to find better results even after the PSO, GA, and BFO seem to have stagnated. The $ABFO_1$ achieved almost the same performance as the $ABFO_0$ on functions 2, 3, and 4.

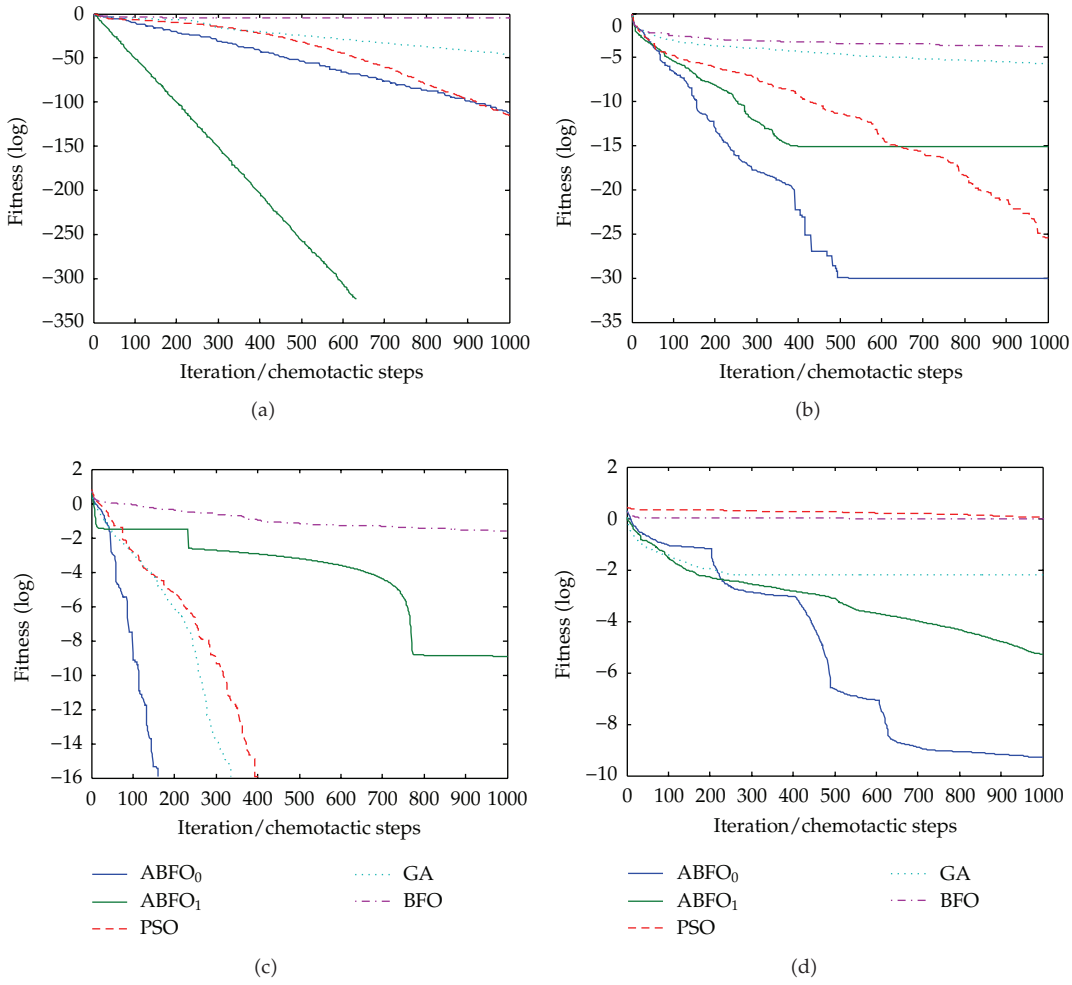


Figure 5: Convergence results of BFO, $ABFO_0$, $ABFO_1$, PSO, and GA on 2-D benchmark functions. (a) Sphere function; (b) Rosenbrock function; (c) Rastrigin function; (d) Griewank function.

From the rank values presented in Table 3, the search performance of the algorithms tested here is ordered as $ABFO_0 > ABFO_1 = PSO > GA > BFO$.

5.4. Results for the 10-D Problems

The experiments conducted on 2-D problems are repeated on the 10-D problems. The results and convergence processes are presented in Table 4 and Figure 6, respectively. Since all 10-D functions become more difficult than their 2-D counterparts, the results obtained by all tested algorithms are not as good as in 2-D cases. Comparing the results in Table 4 and convergence processes in Figure 6, we can observe that the algorithms achieved similar ranking as in the 2-D problems. That is, the search performance of the algorithms tested here is ordered as $ABFO_0 > ABFO_1 > PSO = GA > BFO$.

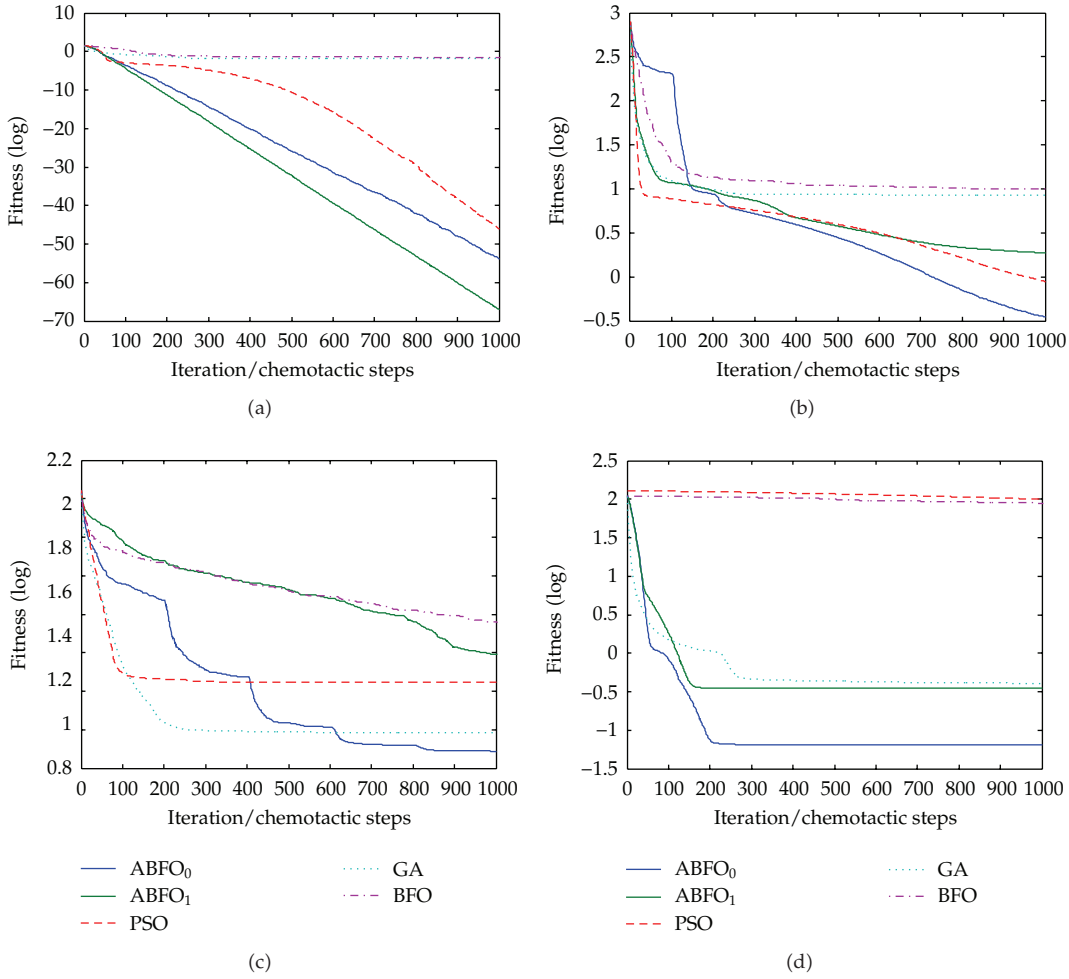


Figure 6: Convergence results of BFO, $ABFO_0$, $ABFO_1$, PSO, and GA on 10-D benchmark functions. (a) Sphere function; (b) Rosenbrock function; (c) Rastrigin function; (d) Griewank function.

5.5. Results for the High-Dimensional Problems

Many real-world optimization problems usually involve hundreds or even thousands of variables. However, previous studies showed that although some algorithms generated good results on relatively low-dimensional benchmark functions, they do not perform satisfactorily for some large-scale cases [31]. Therefore, in order to assess the scalability of our ABFO algorithms, which is crucial for its applicability to real-world problems, the benchmark functions (f_1 to f_4) were extended to 300 dimensions and used in our experimental studies as high-dimensional benchmark functions. The results are presented in Table 5. As the convergence graphs are similar to the 10-D problems, they are not presented.

From Table 5, it can be seen that in terms of final average results, both ABFO algorithms markedly outperformed the other algorithms and converged to good near-optimal solutions. It can also be seen that although PSO and GA achieved satisfactory results

Table 4: Comparison among $ABFO_0$, $ABFO_1$, BFO, PSO, and GA on 10-D problems.

	10-D	BFO	$ABFO_0$	$ABFO_1$	PSO	GA
	Best	0.0131	4.6730*e*–059	5.0821*e*–071	9.7085*e*–052	9.2242*e*–004
	Worst	0.0456	2.3559*e*–053	1.3038*e*–066	1.1714*e*–045	0.0654
f_1	Mean	0.0325	1.2647*e*–054	9.3583*e*–068	7.6021*e*–047	0.0133
	Std	0.0086	4.3424*e*–054	2.5360*e*–067	2.2319*e*–046	0.0149
	Rank	5	2	**1**	3	4
	Best	7.8436	0.1645	0.3620	0.4578	6.6031
	Worst	11.3192	0.5903	4.3232	5.2075	10.1496
f_2	Mean	9.8819	0.3492	1.8660	0.8852	8.5506
	Std	1.0079	0.0966	1.8064	1.0667	0.7340
	Rank	5	**1**	3	2	4
	Best	17.2125	1.1011	5.9785	2.9849	1.0413
	Worst	29.0473	9.0694	32.0221	17.9092	12.3464
f_3	Mean	22.6397	4.8844	15.5429	10.8119	6.0909
	Std	3.1330	1.6419	8.4543	3.8555	2.9628
	Rank	5	**1**	4	3	2
	Best	39.0662	0	0.0910	52.3209	0.1405
	Worst	133.2687	0.1328	0.7516	133.6037	0.9791
f_4	Mean	88.7932	0.0647	0.3551	100.0376	0.4061
	Std	25.0455	0.0308	0.1605	17.2171	0.1875
	Rank	4	**1**	2	5	3
	Average rank	4.75	**1.25**	2.5	3.25	3.25
	Final rank	5	**1**	2	3	3

Table 5: Comparison among $ABFO_0$, $ABFO_1$, BFO, PSO, and GA on 300-d problems.

	Mean function value				
Function	BFO	$ABFO_0$	$ABFO_1$	PSO	GA
$f_1(x)^{300D}$	1.4854e+003	7.4192*e*–003	1.0888*e*–001	1.0031*e*–001	9.0737e+001
$f_2(x)^{300D}$	2.5891e+004	4.6924*e*–002	4.1758*e*–002	3.0752e+002	5.7737e+003
$f_3(x)^{300D}$	3.8524e+003	1.5210*e*–002	9.0428*e*–001	2.0771e+002	2.8621e+003
$f_4(x)^{300D}$	8.2126e+003	8.3185*e*–003	9.2521*e*–001	5.9560e+003	1.0287e+003

in 2- and 10-dimensional benchmark problems, they cannot be scaled up to handle most of the 300-dimensional cases. For the four problems we tested, the original BFO algorithm always failed to converge to good near-optimal solutions.

5.6. Adaptive Bacterial Behaviors in ABFO Model

In order to further analyze the adaptive foraging behaviours of the two proposed ABFO models, we run two simulations. In both simulations, we excluded the reproduction, elimination, and dispersion events in order to illustrate the bacterial behaviours clearly.

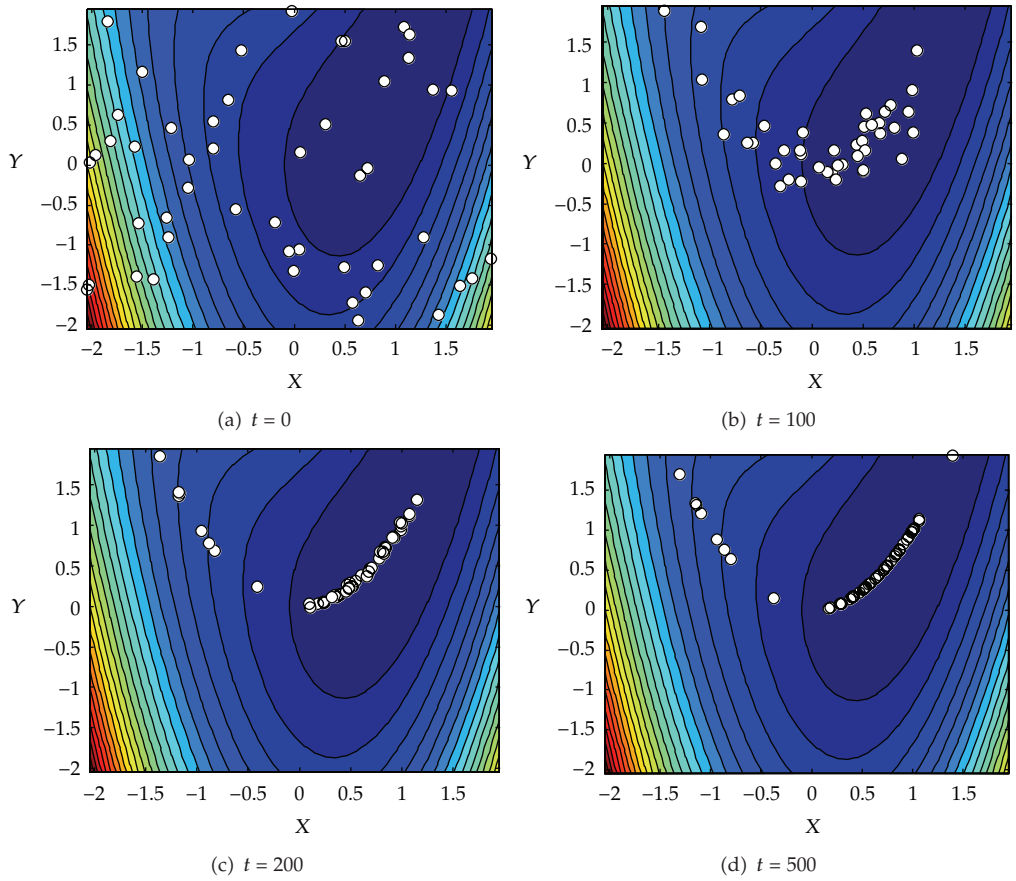


Figure 7: Population evolution of $ABFO_0$ on Rosenbrock function in 500 chemotactic steps.

In the first simulation, the population evolution of the $ABFO_0$ was simulated on 2-D Rosenbrock and Rastrigin functions, which are illustrated in Figures 7 and 8, respectively. Specially, it shows the positions of the bacteria on certain chemotactic steps, where each white circle represents a bacterium. In both cases, the evolution process proceeds 500 chemotactic steps, and we choose $S = 50$, $n = 100$, $C_{\text{initial}} = 0.1$, $\varepsilon_{\text{initial}} = 100$, and $\alpha = \beta = 10$.

Initially, in Figure 7(a), we see that the bacteria colony is distributed randomly over the nutrient map defined by the 2-D Rosenbrock function. In Figure 7(b), we can observe that, in the end of the first phase, all the bacterial producers have found the long and narrow valley of the Rosenbrock function (which contains the global optimum) and move around it. In the second and third phases (Figures 7(c) and 7(d)), the bacterial scroungers initialized in this valley found by the producers and exploit the global optimum along it. Qualitatively, we found a similar pattern in Figure 8, where the bacteria colony pursue the valleys and avoid the peaks of the multimodal Rastrigin function. In the first phase (Figures 8(a) and 8(b)), starting from their random initial positions, the bacterial producers explore many regions of the nutrient map. In the second and third phases (Figures 8(c) and 8(d)), the bacterial scroungers join the sources found by the producers and find a good deal of local optima of Rastrigin function, including the global optimum.

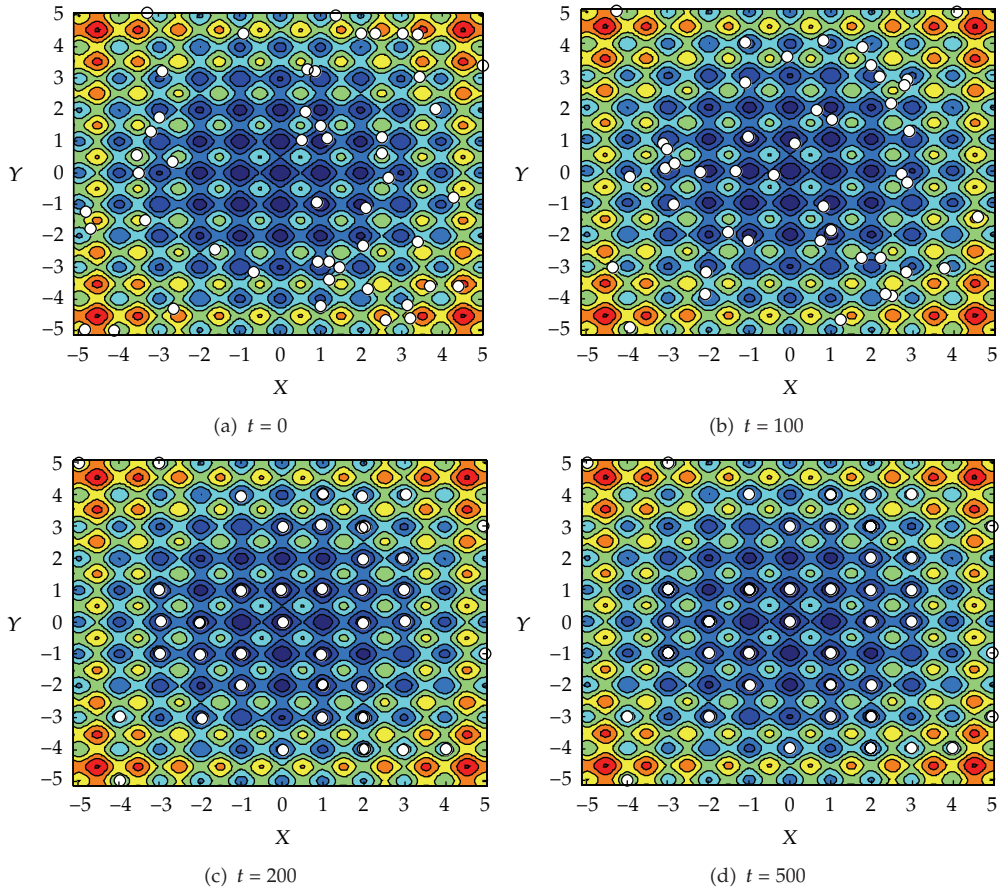


Figure 8: Population evolution of $ABFO_0$ on Rastrigin function in 500 chemotactic steps.

In the second simulation, we demonstrate the self-adaptive foraging behaviour of a single bacterium in the $ABFO_1$ model. Figures 9(a)–9(d) illustrate the trajectories of a single bacterium on the four benchmark functions, respectively, namely, the 2-D Dejong, Rosenbrock, Rastrigin, and Griewank function, which are contour plotted. In all the cases, the evolution process proceeds 1000 chemotactic steps, and we choose the same parameters setting as in Table 2, except $S = 1$.

Figure 9(a) illustrates the bacterial trajectories in the 2-D unimodal Sphere function, which start at point (−5, −5). As we can see, the proposed self-adaptive strategy is important because it permits the bacteria to refine its foraging behaviour adaptively. At the beginning of the simulation, the bacterium starts exploring the search space. In that manner, the bacterium does not waste much time before finding the promising region that contains the global optimum, because the large run-length unit encourages long-range search. On the other hand, by self-adapting the parameters, the bacterium slows down (i.e., the bacterium enters the exploitation state) near the optimum in order to pursue the more and more precise solutions. Figure 9(c) illustrates the bacterial trajectories (start at point (−4, −4)) in the contour-plotted 2-D multimodal Rastrigin function. We can observe that the bacterium was switching between exploitation and exploration states by self-adapting its run-length

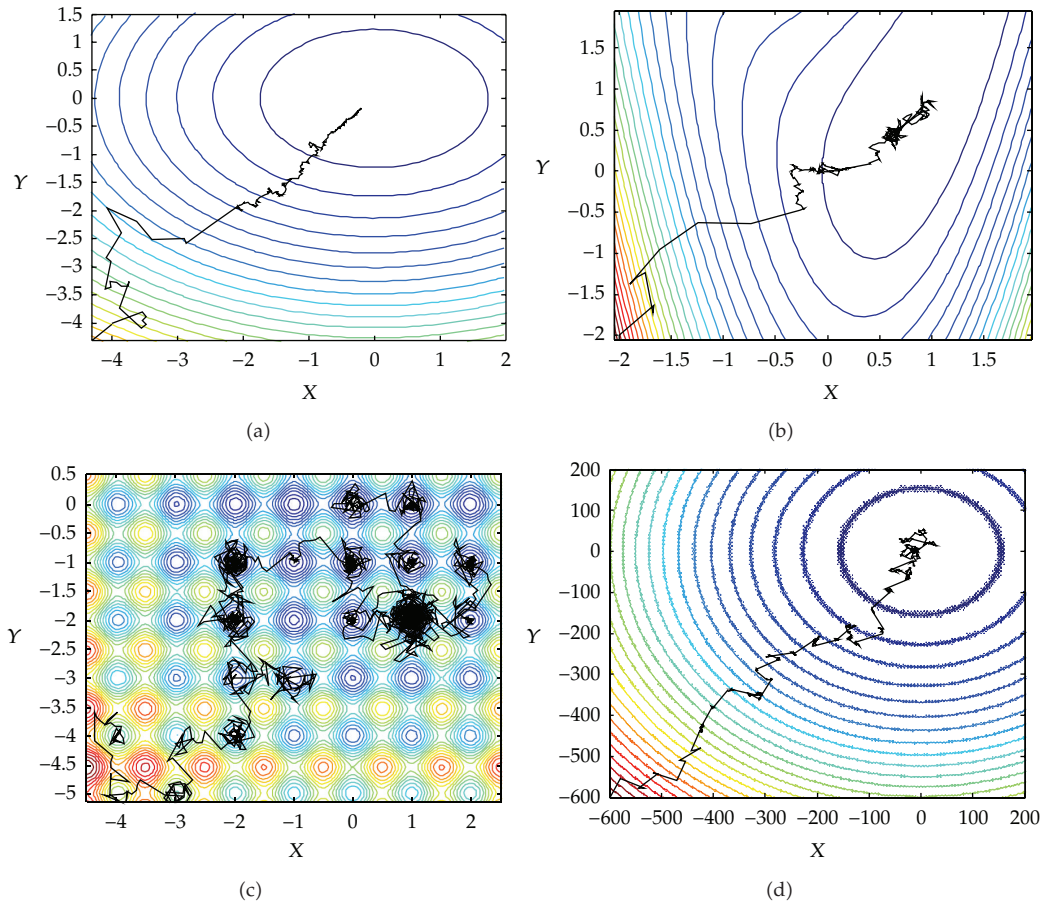


Figure 9: Single bacterium self-adaptive foraging trajectories of $ABFO_1$ model on 2-D benchmark functions. (a) Sphere function; (b) Rosenbrock function; (c) Rastrigin function; (d) Griewank function.

unit. 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 by this bacterium, it moves away from this region and starts exploring the other regions of the search space until another better region is discovered. Finally, we can observe that the bacterium finds the domain that contains the global optimum of Rastrigin function. The similar self-adaptive pattern can be noticed in Figures 9(b) and 9(d), which plotted the bacterial foraging trajectories on Rosenbrock and Griewank functions, respectively. Clearly, this result captures the important aspects of the ACS behaviour that takes place in nature.

In Figure 10, we have also drawn the associated evolution of the run-length unit of this bacterium along its search on all the functions. This provides an intuitive explanation of what is happening during the search of the self-adaptive bacterial foraging algorithm. From Figure 10, 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.

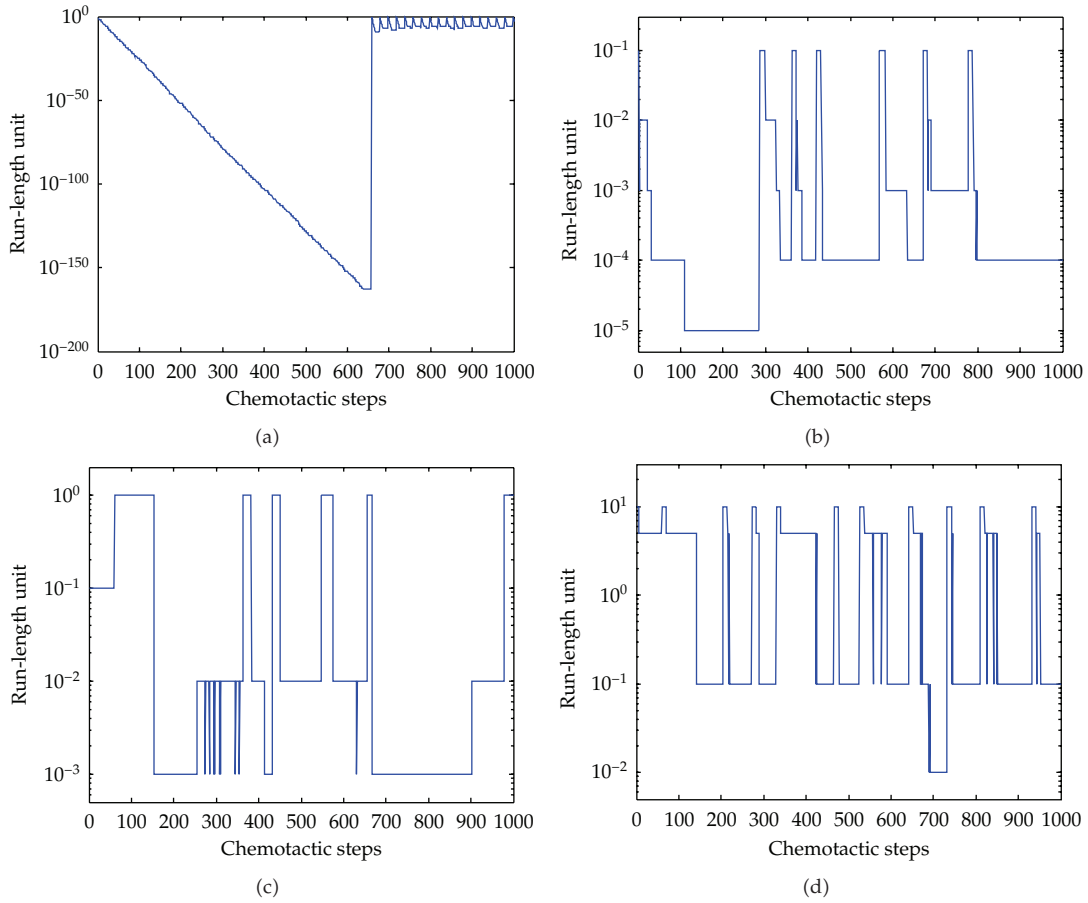


Figure 10: Dynamics of the run-length unit automatically produced by the self-adaptive criterion during the single bacterium foraging on 2-D benchmark functions. (a) Sphere function; (b) Rosenbrock function; (c) Rastrigin function; (d) Griewank function.

6. Conclusions

In this paper, we first 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 behaviours. That is, the bacteria with the large run-length unit have the exploring ability while the bacteria with the relatively small run-length unit have the exploiting skill. This feature is, to our knowledge, used in this paper for the first time. Then, we introduced two methods of casting bacterial foraging optimization into the adaptive fashion by changing the value of the run-length unit during the algorithm execution. Based on the two adaptive methods and the original BFO, we present two adaptive bacterial foraging optimizers, namely $ABFO_0$ and $ABFO_1$, both of which significantly improve the performance of the original algorithm.

The $ABFO_0$ algorithm is based on the producer-scrounger foraging model. In $ABFO_0$, the algorithm evolution process is divided into multiple explore/exploit phases, each characterized by the different classes of bacterial individuals—producers and scroungers—depending on the particular run-length unit that they used. In explore phases, the bacterial

producer explores the search space using the large run-length unit, which permits them to explore the whole space to locate global optimum and avoid becoming trapped in local optima. While in exploit phases, the bacterial scroungers join the resource (i.e., the best-so-far solutions) found by the producers and perform exploitation of their neighborhood. The $ABFO_1$ algorithm is based on the animal searching strategy, ACS. Each bacterium in $ABFO_1$ can be characterized by focused and deeper exploitation of the promising regions and wider exploration of other regions of the search space. The 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.

This automatic alternate shifting between exploitation and exploration during execution is a noticeable feature of the adaptive mechanisms we propose, because it shows how the algorithm decides by itself when to explore and when to exploit the searching space.

Four widely used benchmark functions have been used to test the ABFO algorithms in comparison with the original BFO, the standard PSO and the real-coded GA. The simulation results are encouraging: the $ABFO_0$ and $ABFO_1$ are definitely better than the original BFO for all the test functions and appear to be comparable with the standard PSO and GA.

There are ways to improve our proposed algorithms. 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.

Acknowledgment

This paper is supported by the National 863 Plans Projects of China under Grants 2008AA04A105 and 2006AA04119-5, and the Natural Science Foundation of Liaoning Province under Grant 20091094.

References

[1] C. A. Floudas, *Deterministic Global Optimization: Theory, Methods and Applications*, vol. 37 of *Nonconvex Optimization and Its Applications*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

[2] J. C. Spall, *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*, Wiley-Interscience Series in Discrete Mathematics and Optimization, Wiley-Interscience, Hoboken, NJ, USA, 2003.

[3] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to B, control, and Artificial Intelligence*, University of Michigan Press, Ann Arbor, Mich, USA, 1975.

[4] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics. Part B*, vol. 26, no. 1, pp. 29–41, 1996.

[5] M. Dorigo, G. Di Caro, and L. M. Gambardella, "Ant algorithms for discrete optimization," *Artificial Life*, vol. 5, no. 2, pp. 137–172, 1999.

[6] R. C. Eberchart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceeding of the 6th International Symposium on Micromachine and Human Science*, pp. 39–43, Nagoya, Japan, 1995.

[7] H. Chen, Y. Zhu, K. Hu, and X. He, "Hierarchical swarm model: a new approach to optimization," *Discrete Dynamics in Nature and Society*, vol. 2010, Article ID 379649, 30 pages, 2010.

[8] H. J. Bremermann and R. W. Anderson, "An alternative to back-propagation: a simple rule of synaptic modification for neural net training and memory," Tech. Rep. PAM-483, Center for Pure and Applied Mathematics, University of California, 1990.

[9] S. D. Müller, J. Marchetto, S. Airaghi, and P. Koumoutsakos, "Optimization based on bacterial chemotaxis," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 16–29, 2002.

[10] K. M. Passino, "Biomimicry of bacterial foraging for distributed optimization and control," *IEEE Control Systems Magazine*, vol. 22, no. 3, pp. 52–67, 2002.

[11] D. H. Kim and J. H. Cho, "Adaptive tuning of PID controller for multivariable system using bacterial foraging based optimization," in *Proceedings of the 3rd International Atlantic Web Intelligence Conference on Advances in Web Intelligence (AWIC '05)*, vol. 3528 of *Lecture Notes in Computer Science*, pp. 231–235, June 2005.
[12] S. Mishra, "A hybrid least square-fuzzy bacterial foraging strategy for harmonic estimation," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 1, pp. 61–73, 2005.
[13] M. Tripathy, S. Mishra, L. L. Lai, and Q. P. Zhang, "Transmission loss reduction based on FACTS and bacteria foraging algorithm," in *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN '06)*, vol. 4193 of *Lecture Notes in Computer Science*, pp. 222–231, September 2006.
[14] D. H. Kim and C. H. Cho, "Bacterial foraging based neural network fuzzy learning," in *Proceedings of the Indian International Conference on Artificial Intelligence (IICAI '05)*, pp. 2030–2036, 2005.
[15] D. H. Kim, A. Abraham, and J. H. Cho, "A hybrid genetic algorithm and bacterial foraging approach for global optimization," *Information Sciences*, vol. 177, no. 18, pp. 3918–3937, 2007.
[16] A. Biswas, S. Dasgupta, S. Das, and A. Abraham, "Synergy of PSO and bacterial foraging optimization—a comparative study on numerical benchmarks," in *Proceedings of the 2nd International Sympium on Hybrid Artificial Intelligence Systems (HAIS): Advances Soft Computing*, vol. 44, pp. 255–263, 2007.
[17] H. Chen, Y. Zhu, and K. Hu, "Multi-colony bacteria foraging optimization with cell-to-cell communication for RFID network planning," *Applied Soft Computing Journal*, vol. 10, no. 2, pp. 539–547, 2010.
[18] 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.
[19] J. Adler, "Chemotaxis in bacteria," *Science*, vol. 153, no. 3737, pp. 708–716, 1966.
[20] J. Krause and G. D. Ruxton, *Living in Groups*, Oxford Series in Ecology and Evolution, Oxford University Press, Oxford, UK, 2002.
[21] D. C. Krakauer and M. A. Rodríguez-Gironés, "Searching and learning in a random environment," *Journal of Theoretical Biology*, vol. 177, no. 4, pp. 417–429, 1995.
[22] J. N. M. Smith, "The food searching behaviour of two European thrushes. II. The adaptiveness of the search patterns," *Behavior*, vol. 49, no. 1-2, pp. 1–61, 1974.
[23] L. A. Giraldeau and G. Beauchamp, "Food exploitation: searching for the optimal joining policy," *Trends in Ecology and Evolution*, vol. 14, no. 3, pp. 102–106, 1999.
[24] R. P. Gendron and J. E. R. Staddon, "Searching for cryptic prey: the effect of search rate," *American Naturalist*, vol. 121, no. 2, pp. 172–186, 1983.
[25] R. Hinterding, Z. Michalewicz, and A. E. Eiben, "Adaptation in evolutionary computation: A survey," in *Proceedings of the IEEE Conference on Evolutionary Computation (ICEC,97)*, pp. 65–69, Indianapolis, IN, USA, April 1997.
[26] T. Bäck, "Introduction to the Special Issue: self-adaptation," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 2, pp. 3–4, 2001.
[27] Y. Shi and R. C. Eberhart, "Fuzzy adaptive particle swarm optimization," in *Proceedings of the IEEE Conference on Evolutionary Computation (ICEC '01)*, vol. 1, pp. 101–106, May 2001.
[28] H. Chen and Y. Zhu, "Optimization based on symbiotic multi-species coevolution," *Applied Mathematics and Computation*, vol. 205, no. 1, pp. 47–60, 2008.
[29] H. Chen, Y. Zhu, and K. Hu, "Discrete and continuous optimization based on multi-swarm coevolution," *Natural Computing*, vol. 9, no. 3, pp. 659–682, 2010.
[30] S. Sumathi, T. Hamsapriya, and P. Surekha, *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*, Springer, Berlin, Germany, 2008.
[31] X. Yao and Y. Liu, "Scaling up evolutionary programming algorithms," in *Proceedings of the 7th Annual Conference on Evolutionary Programming (EP '98)*, pp. 103–112, 1998.