A SYNERGY OF DIFFERENTIAL EVOLUTION AND BACTERIAL FORAGING OPTIMIZATION FOR GLOBAL OPTIMIZATION

Arijit Biswas, Sambarta Dasgupta, Swagatam Das[*], *Ajith Abraham*[†]

Abstract: The social foraging behavior of *Escherichia coli* bacteria has recently been studied by several researchers to develop a new algorithm for distributed optimization control. The Bacterial Foraging Optimization Algorithm (BFOA), as it is called now, has many features analogous to classical Evolutionary Algorithms (EA). Passino [1] pointed out that the foraging algorithms can be integrated in the framework of evolutionary algorithms. In this way BFOA can be used to model some key survival activities of the population, which is evolving. This article proposes a hybridization of BFOA with another very popular optimization technique of current interest called Differential Evolution (DE). The computational chemotaxis of BFOA, which may also be viewed as a stochastic gradient search, has been coupled with DE type mutation and crossing over of the optimization agents. This leads to the new hybrid algorithm, which has been shown to overcome the problems of slow and premature convergence of both the classical DE and BFOA over several benchmark functions as well as real world optimization problems.

Key words: *Bacterial foraging, hybrid optimization, differential evolution, genetic algorithm, and radar poly-phase code design*

Received: November 26, 2007
Revised and accepted: December 15, 2007

1. Introduction

Foraging can be modeled as an optimization process where an animal seeks to maximize energy per unit time spent for foraging. This view led Passino *et al.* to formulate a new bio-inspired algorithm for distributed search and optimization, which they named Bacteria Foraging Optimization Algorithm (BFOA) [1,

[*]Arijit Biswas, Sambarta Dasgupta, Swagatam Das
Department of Electronics and Telecommunication Engineering, Jadavpur University, Kolkata, India, E-mail: arijitbiswas87@gmail.com, sambartadg@gmail.com, swagatamdas19@yahoo.co.in
[†]Ajith Abraham
Center of Excellence for Quantifiable Quality of Service, Norwegian University of Science and Technology, Trondheim, Norway, E-mail: ajith.abraham@ieee.org

2]. Although the algorithm is no way equivalent to a conventional Evolutionary Algorithm (EA) [3, 4] both the paradigms that mean BFOA and EA have many analogous features. The reader who is familiar with both this algorithms may easily perceive the algorithmic resemblance between the fitness function and nutrient concentration function, selection and bacterial reproduction, mutation and elimination-dispersal. Hence it is more expected that the EA and BFOA may reinforce each other if integrated as a single hybrid evolutionary algorithm. One of the main driving forces of the BFOA is the computational chemotaxis which actually entails a kind of stochastic gradient search (where only an approximation of gradient is used and not the analytical gradient information). In the present article this feature of BFOA has been integrated into the framework of the Differential Evolution (DE) [5, 6] algorithm. The DE is a simple Genetic Algorithm (GA) [7, 8], which applies a differential mutation operator that distinguishes it from the traditional GA. It has repeatedly outperformed many types of EA's and other meta-heuristics like Particle Swarm Optimization (PSO) when tested over both benchmark and real world problems. DE is, however, not free from problems of slow and premature convergence. Similarly experimentation with several benchmark functions reveals that the BFOA also possesses a poor convergence behavior over multimodal and rough fitness landscapes. Until date there have been a few successful applications of the said algorithm in optimal control engineering, harmonic estimation [9], transmission loss reduction [10], machine learning [11] and active power filter design [12] but hardly any research effort has been undertaken to improve the basic optimizing capabilities of the BFOA. Kim *et al.* proposed a hybrid approach involving the GA and BFOA for numerical optimization [13].

The main objective of this work is to illustrate that the integration of some features from both the DE and the BFOA can prove very effective in tackling many nearly intractable optimization problems on which both the classical algorithms perform poorly. The proposed algorithm referred to here as Chemotactic Differential Evolution (CDE) has been extensively compared with two state of the art variants of DE, the classical BFOA and BFOA–GA hybrid. Since the chemotaxis used in the CDE deviates from the classical form, we compare the CDE also with the BFOA equipped with adaptive chemotactic step. Such comparisons over test bed of six well-known numerical benchmark functions and a real-world problem of spread-spectrum radar poly-phase code design, illustrate the effectiveness of the proposed algorithm.

The rest of the paper is organized in the following way. In Section 2 we briefly review both the BFOA and the DE and point out the motivation for the incorporation of computational chemotaxis in the DE. Section 3 describes the proposed hybrid optimization algorithm in sufficient details. Section 4 describes the experimental set-up and simulation strategies. Numerical results have been presented and discussed in Section 5. Finally, future research directions are provided and conclusions are drawn in Section 6.

2. Brief Overview of BFOA and DE

In this section we briefly outline both the BFOA and the DE algorithms.

2.1 The differential evolution algorithm

Like any other evolutionary algorithm, the DE also starts with a population of *NP* D-dimensional parameter vectors. We will represent subsequent generations in the DE by discrete time steps like $t = 0, 1, 2 \ldots t, t + 1$ etc. Since the vectors are likely to be changed over different generations we may adopt the following notation for representing the i-th vector of the population at the current generation (i.e., at time $t = t$) as:

$$\vec{X}_i(t) = [x_{i,1}(t), x_{i,2}(t), x_{i,3}(t) \ldots\ldots x_{i,D}(t)]. \tag{1}$$

For each parameter of the problem, there may be a certain range within which value of the parameter should lie for better search results. At the very beginning of a DE run or at $t = 0$, problem parameters or independent variables are initialized somewhere in their feasible numerical range. So, if the j-th parameter of the given problem has its lower and upper bound as $x_{\min,j}$ and $x_{\max,j}$ respectively, then we may initialize the j-th component of the i-th population members as

$$x_{i,j}(0) = x_{\min,j} + rand_j(0, 1).(x_{\max,j} - x_{\min,j}), \tag{2}$$

where $rand_j(0, 1)$ is the j-th instantiation of a uniformly distributed random number lying between 0 and 1. The following steps are taken next.

2.1.1 Mutation

After initialization, the DE creates a *donor* vector $\vec{V}_i(t)$ corresponding to each population member or *target* vector $\vec{X}_i(t)$ in the current generation through mutation. It is the method of creating this donor vector, which demarcates between the various DE schemes. For example, five most frequently referred mutation strategies implemented in the public-domain DE codes available online at http://www.icsi.berkeley.edu/∼storn/code.html are listed as follows:

$$\text{"DE/rand/1":} \vec{V}_i(t) = \vec{X}_{r_1^i}(t) + F.(\vec{X}_{r_2^i}(t) - \vec{X}_{r_3^i}(t)) \tag{3a}$$

$$\text{"DE/best/1":} \vec{V}_i(t) = \vec{X}_{best}(t) + F.(\vec{X}_{r_1^i}(t) - \vec{X}_{r_2^i}(t)) \tag{3b}$$

"DE/target-to-best/1":

$$\vec{V}_i(t) = \vec{X}_i(t) + F.(\vec{X}_{best}(t) - \vec{X}_i(t)) + F.(\vec{X}_{r_1^i}(t) - \vec{X}_{r_2^i}(t)) \tag{3c}$$

$$\text{"DE/best/2":} \vec{V}_i(t) = \vec{X}_{best}(t) + F.(\vec{X}_{r_1^i}(t) - \vec{X}_{r_2^i}(t)) + F.(\vec{X}_{r_3^i}(t) - \vec{X}_{r_4^i}(t)) \tag{3d}$$

$$\text{"DE/rand/2":} \vec{V}_i(t) = \vec{X}_{r_1^i}(t) + F.(\vec{X}_{r_2^i}(t) - \vec{X}_{r_3^i}(t)) + F.(\vec{X}_{r_4^i}(t) - \vec{X}_{r_5^i}(t)) \tag{3e}$$

The indices $r_1^i, r_2^i, r_3^i, r_4^i$ and r_5^i are mutually exclusive integers randomly chosen from range [1, *NP*], which are also different from index i. These indices are randomly generated once for each mutant vector. The scaling factor F is a positive control parameter for scaling the difference vectors. $\vec{X}_{best,G}$ is the best individual vector with the best fitness function value in the population at generation G. The general convention used for naming the various mutation strategies is DE/x/y/z, where DE stands for the Differential Evolution, x represents a string denoting the vector to be perturbed and y is the number of difference vectors considered for perturbation of $x.z$ stands for the type of crossover being used (exp: exponential; bin: binomial). The following section discusses the crossover step in the DE.

2.1.2 Crossover

Next, to increase the potential diversity of the population a crossover scheme is undertaken. The DE family of algorithms can use two kinds of cross over schemes, namely *exponential* and *binomial.* The donor vector exchanges its "body parts", i.e., components with the target vector $\vec{X}_i(t)$ under this scheme to form the *trial* vector $\vec{U}_i(t)$. We here outline the binomial crossover scheme, which comes into play in our present analysis. In this case the crossover is performed on each of the D variables whenever a randomly picked number between 0 and 1 is within the CR value. In this case the number of parameters inherited from the mutant has a (nearly) binomial distribution. The scheme may be outlined as

$$\begin{aligned} u_{i,j}(t) \quad &= \quad v_{i,j}(t) \text{ If } (rand_i(0,1) \leq CR) \text{ or } (j = rn(i)) \\ & \quad\quad x_{i,j}(t) \text{ If } (rand_i(0,1) > CR) \text{ or } (j \neq rn(i)) \end{aligned} \tag{4}$$

where $rand_j(0,1) \in [0,1]$ is the j-th evaluation of a uniform random number generator. $rn(i) \in [1, 2, \ldots, D]$ is a randomly chosen index which ensures that $\vec{U}_i(t)$ gets at least one component from $\vec{V}_i(t)$. It is instantiated once for each vector. In this article we have not taken into account the term $rn(i)$ so that CR may be exactly equal to the cross-over probability p_{Cr}.

2.1.3 Selection

In this way for each target vector $\vec{X}_i(t)$ a trial vector $\vec{U}_i(t)$ is created. To keep the population size constant over subsequent generations, the next step of the algorithm calls for 'selection' to determine which one of the target and the trial vector will survive in the next generation, i.e., at time $t = t + 1$. The DE actually involves the Darwinian principle of "Survival of the fittest" in its selection process, which may be outlined as,

$$\vec{X}_i(t+1) = \begin{cases} \vec{U}_i(t) & \text{if } f(\vec{U}_i(t)) \leq f(\vec{X}_i(t)) \\ \vec{X}_i(t) & \text{if } f(\vec{U}_i(t)) > f(\vec{X}_i(t)) \end{cases} \tag{5}$$

where f is the function to be minimized. So if the new trial vector yields a better value of the fitness function, it replaces its parent in the next generation; otherwise the parent is retained in the population. Hence the population either gets better (w.r.t the fitness function) or remains constant but never deteriorates.

2.2 The bacterial foraging optimization algorithm (BFOA)

The bacterial swarm proceeds through four principal mechanisms namely chemotaxis, swarming, reproduction and elimination-dispersal. Below we briefly describe each of these processes and finally provide a pseudo-code of the entire algorithm.

i) **Chemotaxis**: This process simulates the movement of an *E.coli* cell through swimming and tumbling via flagella. Biologically an *E.coli* bacterium can move in two different ways. It can swim for a period of time in the same direction or it may tumble, and alternate between these two modes of operation for the entire lifetime. Suppose $\theta^i(j,k,l)$ represents i-th bacterium at j-th chemotactic, k-th reproductive and l-th elimination dispersal step. $C(i)$ is the size of the step taken in the random direction specified by the tumble (run length unit). Then in computational chemotaxis the movement of the bacterium may be represented by

$$\theta^i(j+1,k,l)=\theta^i(j,k,l)+C(i)\frac{\Delta(i)}{\sqrt{\Delta^T(i)\Delta(i)}}, \tag{6}$$

where Δ indicates a unit length vector in the random direction.

ii) **Swarming**: An interesting group behavior has been observed for several motile species of bacteria including *E.coli* and *S. typhimurium*, where stable spatio-temporal patterns (swarms) are formed in semisolid nutrient medium. A group of *E.coli* cells arrange themselve in a traveling ring by moving up the nutrient gradient when placed amidst a semisolid matrix with a single nutrient chemo-effecter. The cells when stimulated by high level of succinate release an attractant aspertate, which helps them to aggregate into groups and thus move as concentric patterns of swarms of high bacterial density. The cell to cell, signaling in *E.coli* swarm may be represented with the following function.

$$J_{cc}(\theta^i(j,k,l))=\sum_{i=1}^{s}[-d_{attrac\tan t}\exp(-w_{attrac\tan t}\sum_{m=1}^{p}(\theta_m-\theta_m^i)^2)]+$$
$$+\sum_{i=1}^{s}[h_{repellant}\exp(-w_{repellant}\sum_{m=1}^{p}(\theta_m-\theta_m^i)^2)] \tag{7}$$

where $\theta=[\theta_1,\theta_{2,\dots\dots\dots,}\theta_D]^T$ is a point in the D-dimensional search domain.

iii) **Reproduction:** The least healthy bacteria eventually die while each of the healthier bacteria (those yielding higher value of fitness function) asexually split into two bacteria which are placed in the same location. This keeps the swarm size constant.

iv) **Elimination and dispersal**: Gradual or sudden changes in the local environment where a bacterium population lives may occur due to various reasons, e.g. a significant local rise of temperature may kill a group of bacteria that are currently in a region with a high concentration of nutrient gradients. Events can take place in such a fashion that all the bacteria in a region are killed or a group is dispersed into a new location. To simulate this phenomenon in the BFOA some bacteria are liquidated at random with a very small probability while the new replacements are randomly initialized over the search space.

The pseudo-code of the complete algorithm has been provided below:

The BFOA Algorithm
Parameters:

[Step 1] Initialize parameters $n, N, N_C, N_S, N_{re}, N_{ed}, P_{ed}, C(i)(i = 1, 2 \ldots N), \theta^i$.
Where,

n:	Dimension of the search space,
N:	the number of bacteria in the population,
N_C:	chemotactic steps,
N_{re}:	the number of reproduction steps,
N_{ed}:	the number of elimination-dispersal events,
P_{ed}:	elimination-dispersal with probability,
$C(i)$:	the size of the step taken in the random direction specified by the tumble.

Algorithm:

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

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

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

[a] For $i = 1, 2 \ldots N$, take a chemotactic step for bacterium i as follows.

[b] Compute fitness function, J $(i,\ j,\ k,\ l)$.
Let, $J(i, j, k, l) = J(i, j, k, l) + J_{cc}(\theta^i(j, k, l), P(j, k, l))$ (i.e., add on the cell-to cell attractant–repellant profile to simulate the swarming behavior) where J_{cc} is defined in (2).

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

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

[e] Move: Let

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

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

[f] Compute $J(i, j{+}1, k, l)$ and let $J(i, j, k, l) = J(i, j, k, l){+}J_{cc}(\theta^i(j, k, l), P(j, k, l))$.

[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 $J(i, j + 1, k, l) < J_{last}$ (if doing better), let $J_{last} = J(i, j + 1, k, l)$ and let

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

  and use this $\theta(i + 1, j, k)$ to compute the new J$(i, j + 1, k, l)$as we did in [f]
- ∗ Else, let m=N_s. This is the end of the while statement.

[h] Go to next bacterium (i+1) if $i \neq N$ (i.e., go to [b] to process the next bacterium).

[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:

[a] For the given k and l, and for each $i = 1, 2, \ldots, N$, let

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

be the health of bacterium i (a measure of how many nutrients it got over its lifetime and how successful it was at avoiding noxious substances). Sort bacteria and chemotactic parameters $C(i)$ in order of ascending cost J_{health} (higher cost means lower health).

[b] The S_r bacteria with the highest J_{health} values die and the remaining S_r bacteria with the best values split (this process is performed by the copies that are made and placed at the same location as their parent).

[**Step 7**] If $k < N_{re}$, go to [step 3]. In this case, we have not reached the number of specified reproduction steps, so we start the next generation of the chemotactic loop.

[**Step 8**] Elimination-dispersal: For $i = 1, 2, \ldots, N$, with probability P_{ed}, eliminate and disperse each bacterium, and this result 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.

3. The Hybrid Algorithm

The DE has reportedly outperformed powerful meta-heuristics like genetic algorithm (GA) and particle swarm optimization (PSO) [14]. Practical experiences suggest that the DE may occasionally stop proceeding towards the global optima, while the population has not converged to a local optima or any other point. Occasionally even new individuals may enter the population but the algorithm does not progress by finding any better solutions. This situation is usually referred to as *stagnation.* The DE also suffers from the problem of premature convergence [15] where the population converges to some local optima of a multimodal objective function loosing its diversity. On the other hand, experiments with several benchmark functions reveal that the BFOA possesses a poor convergence behavior over multimodal and rough fitness landscapes as compared to other bio-inspired optimization techniques like GA, PSO etc. [16]. Its performance is heavily affected with the growth of search space dimensionality. Previously to improve the performance of the DE, some attempts have been made to hybridize it with a few local search techniques, e.g. [17] and meta-heuristics like PSO [18]. Recently in 2007 Kim *et al.* developed a hybrid approach involving the GA and the BFOA for function optimization [13]. Their algorithm outperformed both the GA and the BFOA over several numerical benchmarks and a practical PID tuner design problem. In the present work following the same train of thought, we have incorporated an adaptive chemotactic step borrowed from the realm of the BFOA into the DE. The computational chemotaxis in the BFOA serves as a stochastic gradient descent based local search. It was seen to greatly improvise the convergence characteristics of the classical DE. The resulting hybrid algorithm is referred here as the CDE (Chemotactic Differential Evolution).

In the CDE, each trial solution vector first undergoes an adaptive computational chemotaxis. The trial solution is visualized as an *E.coli* bacterium. During the process of chemotaxis, bacterium in proximity of venomous substance takes larger chemotactic step to move towards the nutrient substances. Before each movement, it is ensured that bacterium moves in the direction of increasing nutrient substance concentration, i.e., region with smaller objective function value. After this, it is subjected to the DE mutation. For the trial solution vector in population three vectors, other than the previous one, are selected. One of the three vectors is added with scaled difference of the remaining two. The vector thus produced probabilistically interchanges its components with the original vector (just like genes of two chromosomes). Offspring vector replaces the original one if the objective function value is smaller for it. The process is repeated several times over the entire population in order to obtain the optimal solution. The brief pseudo-code of the algorithm has been provided below:

The CDE (Chemotactic DE) Algorithm

Initialize parameters S, N_C, , N_S,, *C(i)(i=1,2…N), F, CR.*

Where,

S: The number of bacteria in the population,

D: dimension,

N_C:	no. of chemotactic steps,
$C(i)$:	the size of the step taken in the random direction specified by the tumble.
F:	scale factor for DE type mutation
CR:	crossover Rate.

Set $j = 0$, $t = 0$;
Chemotaxis loop: $j = j + 1$;
Differential evolution mutation loop: $t = t + 1$;
$\theta(i, j, t)$ denotes the position of the i-th bacterium in the j-th chemotactic and t-th differential evolution loop.
for $i = 1, 2, \ldots, S$, a chemotactic step is taken for i-th bacterium.

(a) **Chemotaxis loop:**

(i) Value of the objective function $J(i, j, t)$ is computed where $J(i, j, t)$ symbolizes value of objective function at j-th chemotaxis cycle for i-th bacterium at t-th DE mutation step.

(ii) $J_{last} = J(i, j, t)$ we store this value of objective function for comparison with values of an objective function yet to be obtained in future.

(iii) **Tumble:** generate a random vector $\Delta(i) \in \Re^D$ with each element $\Delta_m(i), m = 1, 2, \ldots, D$ is a random number on $[-1, 1]$.

(iv) **Move:** $\theta(i, j+1, t) = \omega.\theta(i, j, t) + C(i).(\Delta(i)/\sqrt{\Delta(i).\Delta^T(i)})$.
Where $\omega =$ inertia factor which is generally equals to 1 but becomes 0.8 if the function has an optimal value close to 0.
$C(i) =$ step size for k-th bacterium $= ((J(i, j, t))^{1/3} - 20)/((J(i, j, t))^{1/3} + 300)$
Step size is made an increasing function of objective function value to have a feedback arrangement.

(v) $J(i, j, t)$ is computed.

(vi) **Swim:** We consider here only i-th bacterium is moving and others are not moving.

Now let $m = 0$;
while $m < N_s$ (no of steps less than max limit).
Let $m = m + 1$;
If $J(i, j, t) < J_{last}$ (if going better)

$$J_{last} = J(i, j, t).$$

And let, $\theta(i, j+1, t) = \omega.\theta(i, j, t) + C(i).(\Delta(i)/\sqrt{\Delta(i).\Delta^T(i)})$.
Else, $m = N_s$ (end of while loop);
for $i = 1, 2, \ldots., S$, a differential evolution mutation step is taken for i-th bacterium.

(b) Differential evolution mutation loop:

(i) For each $\theta(i, j+1, t)$ trial solution vector we choose randomly three other distinct vectors from the current population namely $\theta(l), \theta(m), \theta(n)$ such that $i \neq l \neq m \neq n$.

(ii) $V(i, j+1, t) = \theta(l) + F.(\theta(m) - \theta(n))$,

where, $V(i, j+1, t)$ is the donor vector corresponding to $\theta(i, j+1, t)$.

(iii) Then the donor and the target vector interchange components probabilistically to yield a trial vector $U(i, j+1, t)$ following:

$U_p(i, j+1, t) = \quad V_p(i, j+1, t)$ If $(rand_p(0,1) \leq CR)$ or $(p = rn(i))$

$\theta_p(i, j+1, t)$ If $(rand_p(0,1) > CR)$ or $(p \neq rn(i))$ for p-th dimension.

Where $rand_p(0,1) \in [0,1]$ is the p-th evaluation of a uniform random number generator. $rn(i) \in \{1, 2, \ldots, D\}$ is a randomly chosen index which ensures that $U(i, j+1, t)$ gets at least one component from $V(i, j+1, t)$.

(iv) $J(i, j+1, t)$ is computed for trial vector.

(v) If $J(U(i, j+1, t)) < J(\theta(i, j+1, t))$, $\theta(i, j+1, t+1) = U(i, j+1, t)$.

Original vector is replaced by offspring if value of objective function for it is smaller.

If $j < N_c$, start another chemotaxis loop.

4. The Experimental Setup

4.1 Benchmark functions used

The performance of the CDE algorithm has been evaluated on a test suite of six well-known benchmarks (Tab. I) [19]. In Tab. I, D represents the number of dimensions (we used n = 15, 30, 45 and 60). The first two test functions are unimodal, having only one minimum. The others are multimodal, with a considerable number of local minima in the region of interest. All benchmark functions except f_6 have the global minimum at the origin or very near to the origin [19]. For the Shekel's foxholes (f_6), the global minimum is at (-31.95, -31.95) and f_6(-31.95, -31.95) $\approx$ 0.998, and the function has only two dimensions. Tab. II summarizes the initialization and search ranges used for these functions. An asymmetrical initialization procedure has been used here following the work reported in [20].

4.2 The spread spectrum radar polyphase code design problem

A famous problem of optimal design arises in the field of spread spectrum radar poly-phase codes [21]. Such a problem suits very well for the application of global optimization algorithms like DE. The problem can be formally stated as:

$$\begin{aligned} &\text{global min } f(\vec{X}) = \max\{\varphi_1(\vec{X}), \dots, \varphi_{2m}(\vec{X})\} \\ &\vec{X} = \{(x_1, \dots, x_D) \in \Re^D | 0 \le x_j \le 2\pi, j = 1, \dots, D\} \end{aligned} \tag{8}$$

where $m = 2n - 1$ and

$$\begin{aligned} \varphi_{2i-1}(\vec{X}) &= \sum_{j=i}^{D} \cos\left(\sum_{k=|2i-j-1|-1}^{j} x_k\right),\ i = 1, \dots, D \\ \varphi_{2i}(\vec{X}) &= 0.5 + \sum_{j=i+1}^{D} \cos\left(\sum_{k=|2i-j-1|-1}^{j} x_k\right),\ i = 1, \dots, D-1 \\ \varphi_{m+i}(\vec{X}) &= -\varphi_i(\vec{X}), i = 1, \dots, m \end{aligned} \tag{9}$$

The work in [27] shows that the above problem is NP-hard. The objective function for $D = 2$ has been shown in Fig. 1.

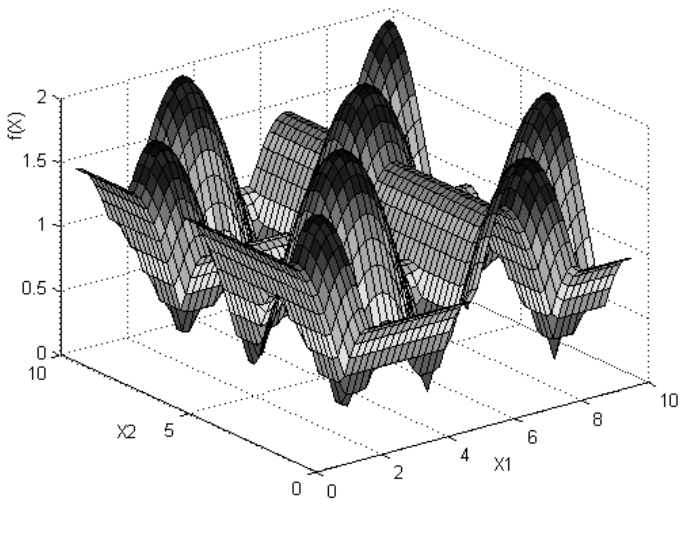

Fig. 1 $f(\vec{X})$ *of (9) for* $D = 2$.

4.3 Simulation strategy

Simulations were carried out to obtain a comparative performance analysis of the proposed CDE method with respect to: (a) DE/rand/1/bin [6] (b) BFOA and (c) BFOA-GA [13]. The first classical DE schemes was chosen because of its wide popularity in solving numerical optimization or engineering problems.

Function	Mathematical Representation
Sphere function	$f_1(\vec{X}) = \sum\limits_{i=1}^{D} x_i^2$
Rosenbrock	$f_2(\vec{X}) = \sum\limits_{i=1}^{D-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$
Rastrigin	$f_3(\vec{X}) = \sum\limits_{i=1}^{D} [x_i^2 - 10\cos(2\pi x_i) + 10]$
Griewank	$f_4(\vec{X}) = \frac{1}{4000} \sum\limits_{i=1}^{D} x_i^2 \prod\limits_{i=1}^{D} \cos(\frac{x_i}{\sqrt{i}}) + 1$
Ackley	$f_5(\vec{X}) = -20\exp(-0.2\sqrt{\left(\frac{1}{D}\sum\limits_{i=1}^{D} x_i^2\right)} - \exp\left(\frac{1}{D}\sum\limits_{i=1}^{D}\cos 2\pi x_i\right) + 20 + e$
Shekel's Foxholes	$f_6(\vec{X}) = \left[\frac{1}{500} + \sum\limits_{j=1}^{25} \frac{1}{j + \sum\limits_{i=1}^{2}(x_i - a_{ij})^6}\right]^{-1}$

Tab. I *Benchmark functions used.*

For classical DE/rand/1/bin the population size *NP* was taken as 10 times the dimension of the problem. To make the comparison fair, the populations for all the considered algorithms (for all problems tested) were initialized using the same random seeds. We choose the number of fitness evaluations (FE) as a measure of computation time instead of 'generations' or 'iterations'.

Thirty independent runs of each of the four algorithms were carried out and the average and the standard deviation of the best-of-run values were recorded. Different maximum number of FEs were used according to the complexity of the problem. For all benchmarks (excluding Shekel's Foxholes function f_6) the stopping criterion was set as reaching a fitness of 1.00e-05. However, for the Shekel's Foxholes function (f_6) it was fixed at 0.998. The spread spectrum radar polyphase code design problem was tested varying n from 2 to 20. However, we report results of just two of the most difficult problem instances in each case ($n = 19, 20$) owing to the space limitations.

f	Global Optimum	Range of search	Range of Initialization
f_1	$f_1(\vec{0}) = 0$	(-100, 100)^n	(50, 100)^n
f_2	$f_2(\vec{1}) = 0$	(-100, 100)^n	(15, 30)^n
f_3	$f_3(\vec{0}) = 0$	(-10, 10)^n	(2.56, 5.12)^n
f_4	$f_4(\vec{0}) = 0$	(-600, 600)^n	(300, 600)^n
f_5	$f_5(\vec{0}) = 0$	(-32, 32)^n	(15, 32)^n
f_6	$f_6(-31.95, -31.95) = 0.998$	(-65.536, 65.536)^2	(0, 65.536)^2

Tab. II *Search ranges of benchmark functions.*

For the DE/rand/1/bin and CDE the cross-over rate $Cr = 0.9$ and the scale factor $F = \lambda = 0.8$ were used. The parametric setup of the classical BFOA has been shown in Tab. III. The same population size S, N_c and N_s has been used for all the BFOA based algorithms compared here (BFOA, BFOA-GA and CDE). All the other parameters (if not explicitly mentioned here) have been fixed according to the best possible choice as described in relevent literature. All the algorithms discussed here have been developed from scratch in Visual C++ on a Pentium IV, 2.3 GHz PC, with 1024 KB cache and 2 GB of main memory in Windows XP environment.

S	N_c	N_s	N_{ed}	p_{ed}	$d_{attract}$	$w_{attract}$	$w_{repellant}$	$h_{repellant}$
50	100	4	1	0.25	0.1	0.2	10	0.1

Tab. III *Parameter setup for BFOA.*

5. Results

The following performance measures are used for our comparative study: (a) quality of the final solution, (b) speed of convergence towards the optimal solution, (c) success rate (frequency of hitting the optimum), and (d) scalability of the algorithms against the growth of problem dimensions. Tab. IV compares the algorithms on the quality of the optimum solution. The mean and the standard deviation (within parentheses) of the best-of-run values for 30 independent runs of each of the four algorithms are presented. Each algorithm was run up to a predetermined maximum number of FEs (depending upon the complexity of the problem). Missing values of standard deviation indicate a zero standard deviation. The best solution in each case has been shown in bold.

Tab. V shows the results of unpaired t-tests between CDE and the best of the three competing algorithms in each case (standard error of difference of the two means, 95% confidence interval of this difference, the t value, and the two-tailed P value). For all cases in Tab. V, sample size = 30 and degrees of freedom = 58. It is interesting to see from Tabs. IV and V that the proposed method meets or beats the nearest competitor in a statistically significant way.

Tab. VI shows, for all test functions and all algorithms, the number of runs (out of 30) that managed to find the optimum solution within a given tolerance or cutoff value. In Tab. VII we report the mean number of Function Evaluations (FEs) and standard deviations (within parentheses) required by each competitor algorithm to converge within the prescribed cut-off value. In each case, the mean is calculated over the corresponding number of runs that managed to converge within the cut-off value. The entries marked NA (Not Applicable) in this table, imply that no run of the corresponding algorithm converged to the cut-off within the maximum number of FEs allowed. Missing values of standard deviation here also indicate a zero standard deviation. In Figs. 2, 3 and 4 we have graphically presented the rate of convergence of all the methods for two most difficult test functions (f_2 and f_3) (in 30 dimensions) and also for the spread spectrum radar polyphase code design problem. We avoid showing plots for all the test cases in order to save space and time. These results show that the proposed method leads to significant improvements in most cases.

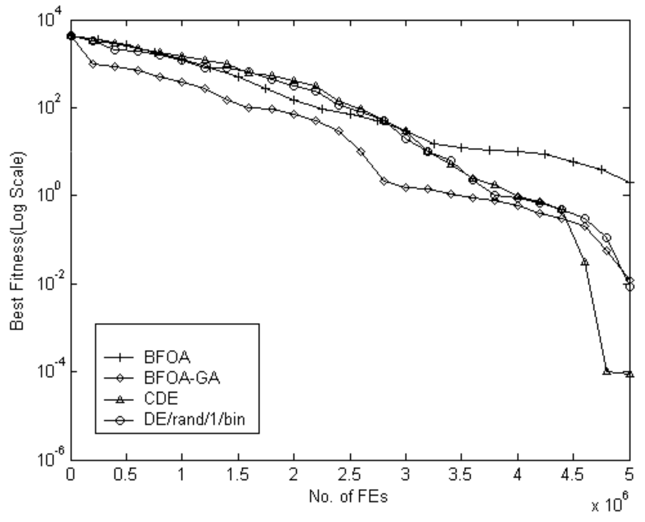

Fig. 2 *The convergence characteristics of contestant algorithms over the Generalised Rastrigin's Function (f_3) for D = 30.*

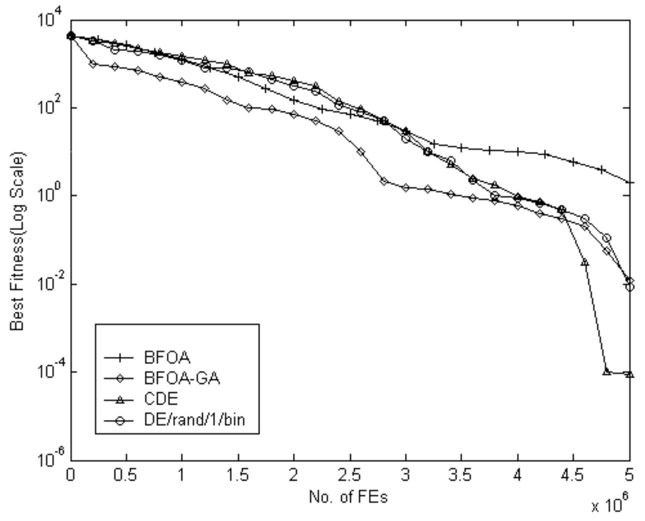

Fig. 3 *The convergence characteristics of contestant algorithms over the Generalised Rosenbrock's Function (f_2) for D = 30.*

Fun	D	Max^m FE	Mean Best Value (Standard Deviation)			
			DE/rand/1/bin	BFOA	BFOA-GA	CDE
f_1	15	50,000	**1.00e-05**	6.2112e-04 (5.342e-03)	**1.00e-05**	**1.00e-05**
	30	1×10^5	**1.00e-05**	9.2234e-03 (2.039e-05)	**1.00e-05**	**1.00e-05**
	45	5×10^5	**1.00e-05**	2.4432e-02 (3.327e-02)	**1.00e-05**	**1.00e-05**
	60	1×10^6	**1.00e-05**	7.2346e+00 (3.715e+00)	**1.00e-05**	**1.00e-05**
f_2	15	50,000	1.6342e-01 (4.253e-05)	9.7691e+00 (5.209e+00)	6.7672e-01 (1.465e-01)	**3.2113e-03 (4.028e-02)**
	30	1×10^5	8.8639e-01 (3.237e-05)	2.7862e+01 (9.282e-05)	8.2364e+00 (4.003e-01)	**2.6752e-01 (5.746e-02)**
	45	5×10^5	5.2318e+00 (3.911e-02)	1.8232e+01 (5.243e-03)	1.8562e+01 (3.461e-01	**4.4653e-02 (3.318e-03)**
	60	1×10^6	1.8704e+01 (6.282e-02)	3.5624e+00 (3.366e-03)	9.3534e+01 (4.568e-03)	**9.1219e-01 (4.346e-02)**
f_3	15	50,000	**1.00e-05**	2.2352e-01 (3.029e-02)	**1.00e-05**	**1.00e-05**
	30	1×10^5	**1.00e-05**	4.9382e+00 (4.732e-01)	6.7827e-05 (1.364e-08)	**1.00e-05**
	45	5×10^5	6.8734e-02 (5.329e-08)	1.2038e+01 (4.917e+00)	3.4039e-03 (2.112e-08)	**6.503e-03 (4.221e-02)**
	60	1×10^6	2.1935e-03 (1.186e-08)	8.7325e+01 (5.521e+00)	1.6341e-03 (6.271e-04)	**8.3627e-04 (2.635e-03)**
f_4	15	50,000	**1.00e-05**	**1.00e-05**	**1.00e-05**	**1.00e-05**
	30	1×10^5	**1.00e-05**	9.5043e-01 (4.895e-02)	2.1214e-04 (6.232e-05)	**1.00e-05**
	45	5×10^5	6.4543e-03 (3.434e-06)	2.1309e+00 (7.281e-01)	1.2651e-02 (3.862e-04)	**3.0071e-04 (1.832e-05)**
	60	1×10^6	8.3247e-02 (1.613e-06)	6.3298e+01 (1.217e+00)	1.3483e-01 (4.012e-02)	**3.2876e-03 (1.536e-03)**
f_5	15	50,000	**1.00e-05**	4.7382e-01 (4.045e-02)	4.0936e-04 (5.992e-03)	**1.00e-05**
	30	1×10^5	**1.00e-05**	2.1779e+00 (7.261e-03)	7.2582e-03 (5.002e-04)	**1.00e-05**
	45	5×10^5	1.0084e-03 (2.923e-06)	1.8271e+01 (4.654e-02)	9.7362e-02 (7.113e-04)	1.817e-04 (1.958e-07)
	60	1×10^6	3.6298e-02 (9.127e-04)	9.7309e+01 (1.682e-01)	3.2175e-01 (5.636e-02)	**2.2627e-04 (6.347e-05)**
f_6	2	50,000	**9.9980e-01**	9.99862e-01 (5.426-04)	9.99805e-01 (4.264e-08)	**9.9980e-01**

Tab. IV *Average and the standard deviation of the best-of-run solution for 30 runs tested on seven benchmark functions (for all cases except f_6, the cut-off value used is 1.00e-05, while for f_6 it is 0.998).*

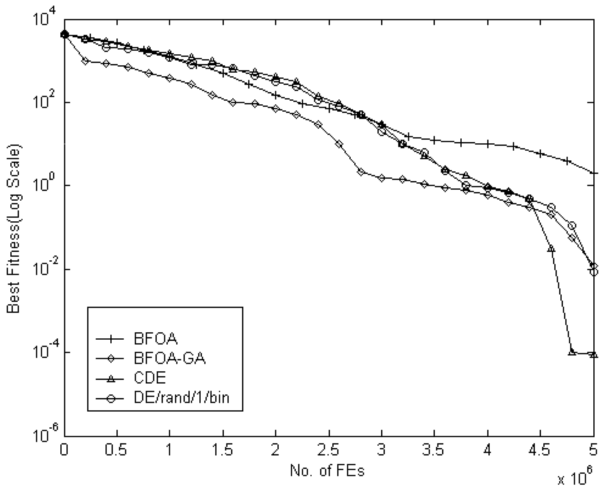

Fig. 4 *The convergence characteristics of contestant algorithms over the radar poly-phase code design problem ($D = 20$).*

Fn, Dim	Std. Err	t	95% Conf. Intvl	Two-tailed P	Significance
f_2, 25	0.000	21040.9635	-0.16340 to -0.16337	< 0.0001	**Extremely significant**
f_2, 50	0.000	13845.0859	-0.275984 to -0.275904	< 0.0001	**Extremely significant**
f_2, 75	0.007	726.4424	-5.20144 to -5.17285	< 0.0001	**Extremely significant**
f_2, 100	0.001	4761.122	-2.95145 to -2.94897	< 0.0001	**Extremely significant**
f_3, 75	0.000	6165.6383	-0.0000553 to -0.00005533	< 0.0001	**Extremely Significant**
f_3, 100	0.000	6.9515	-0.00102 to -0.00057	< 0.0001	**Extremely Significant**
f_4, 75	0.000	175.16	-0.012494 to -0.01221	< 0.0001	**Extremely significant**
f_4, 100	0.007	17.958	-0.1462 to -0.1168	< 0.0001	**Extremely significant**
f_5, 75	0.000	1545.6372	-0.000828 to -0.000825	< 0.0001	**Extremely significant**
f_5, 100	0.000	193.0175	-0.00302 to -0.00296	< 0.0001	**Extremely significant**

Tab. V *Results of unpaired t-tests on the data of Tab. IV.*

Function	Dim	No. of runs converging to the cut-off			
		DE/rand/1/bin	BFOA	BFOA-GA	CDE
f_1	15	**30**	0	**30**	**30**
	30	**30**	0	**30**	**30**
	45	**30**	0	**30**	**30**
	60	**30**	0	**30**	**30**
f_2	15	8	3	10	**17**
	30	0	0	0	**12**
	45	0	0	0	**6**
	60	0	0	0	**3**
f_3	15	**30**	0	**30**	**30**
	30	**30**	0	9	**30**
	45	14	11	15	**30**
	60	6	5	1	**30**
f_4	15	**30**	**30**	**30**	**30**
	30	15	12	13	**30**
	45	6	8	11	**15**
	60	8	4	9	**17**
f_5	15	13	15	15	**30**
	30	12	17	11	**30**
	45	5	8	8	**19**
	60	2	4	4	**11**
f_6	2	**30**	12	17	**30**

Tab. VI *Number of runs converging to the cut-off fitness value for six benchmark functions.*

6. Conclusions and Future Directions

This paper has presented an improved hybrid evolutionary algorithm by combining the DE based mutation operator with bacterial chemotaxis. The present scheme attempts to make a judicious use of exploration and exploitation abilities of the search space and therefore likely to avoid false and premature convergence in many cases. The overall performance of the proposed algorithm is definitely better than a standalone BFOA at least on the numerical benchmarks tested. The performance also appears to be at least comparable with the classical DE/rand/1/bin scheme. The future research effort should focus on reducing the number of user-defined parameters for the BFOA and its variants. Also an empirical study on the effects of these parameters on the convergence behavior of the hybrid algorithms may be worthy to undertake.

Function	Dim	Mean and Std Dev of FEs required to reach the Cut-off value			
		DE/rand/1/bin	BFOA	BFOA-GA	CDE
f_1	15	8682.05 (11.725)	NA	8836.90 (120.287)	**5232.85 (43.232)**
	30	22193. 00 (3.961)	NA	26373. 50 (90.873)	**19282. 30 (29.732)**
	45	45092.45 (291.49)	NA	38274.75 (176.32)	**34789.73 (27.632)**
	60	989173.65 (23.88)	NA	938633.6 (183.98)	**581921.50 (73.818)**
f_2	15	7621.125 (24.821)	8915.35 (4.122)	8372.30 (51.876)	**6832.29 (12.342)**
	30	NA	NA	NA	**15628.83 (43.932)**
	45	NA	NA	NA	**33515.33 (113.432)**
	60	NA	NA	NA	**641234.67 (37.69)**
f_3	15	**34884.45 (4.841)**	46828.05 (5.583)	39522.30 (26.145)	35775.75 (14.858)
	30	67124.05 (34.991)	95614.95 (45. 526)	42313.67 (112.622)	**34322.15 (53.232)**
	45	416454.67 (7.806)	349772.57 (56.92)	**263222.00 (154.34)**	395144.72 (34.95)
	60	892452.50 (15.00)	752754.40 (4.975)	749991	**702321.43 (56.53)**
f_4	15	8725.55 (4.806)	12365.75 (7.039)	7973.25 (26.207)	**4875.15 (15.232)**
	30	17934.85 (18.910)	27568.33 (12.876)	12564.30 (62.863)	**9382.36 (3.982)**
	45	409281.33 (30.663)	447328.15 (35.723)	381305.67 (83.58)	**302615.13 (22.637)**
	60	**726121.59 (143.65)**	837261.25 (3326.78)	962311..35 (31.74)	809182.00 (18.753)
f_5	15	**37261.67 (9.57)**	45930.57 (30.56)	21905.00 (28.48)	60720.50 (19.547)
	30	**62918.57 (22.546)**	90382 (32.822)	87483.46 (56.465)	56483.05 (22.849)
	45	267211.25 (7.558)	389222.40 (29.804)	169023.66 (137.632)	**56472.45 (32.118)**
	60	890332.50 (13.540)	990321.25 (21.525)	680491.25 (230.327)	**502317.25 (42.238)**
f_6	2	16472.85 (24.242)	27502.40 (14.803)	18291.86 (23.238)	**12237.60 (9.851)**

Tab. VII *Mean no. of FEs and standard deviation required to converge to the cut-off fitness over the successful runs.*

D	Mean best-of-run solution (Std Dev)			
	DE/rand/1/bin	BFOA	BFOA-GA	CDE
19	7.4849e-01 (8.93e-05)	7.5745e-01 (6.83e-02)	7.5543e-01 (3.32e-02)	**7.4431e-01 (6.89e-03)**
20	9.5746e-01 (4.23e-03)	9.1932e-01 (4.69e-02)	8.4932e-01 (5.72e-02)	**8.0844e-01 (7.93e-03)**

Tab. VIII *Average and the standard deviation of the best-of-run solution for 30 runs for spread spectrum radar poly-phase code design problem (number of dimensions D = 19 and D = 20). For all cases each algorithm was run for 100,000 FEs.*

D	Std. Err	t	95% Conf. Intvl	Two-tailed P	Significance
19	0.002	2.5024	-0.00734 to -0.00081	0.0152	Significant
20	0.010	3.5880	-0.05792 to -0.01644	0.0007	**Extremely significant**

Tab. IX *Results of unpaired t-tests on the data of Tab. VII.*

D	Cut off Value	Mean No. of FE required to reach the cut-off value			
		DE/rand/1 /bin	BFOA	BFOA-GA	CDE
19	7.60e-01	78378.50	89129.15	75361.30	**71876.60**
20	9.10e-01	**57973.60**	88932.85	67654.50	63131.30

Tab. X *Average no of FEs (for 30 runs) and standard deviations for spread spectrum radar poly-phase code design problem to converge to a given cut-off value of the objective function.*

References

[1] Passino K. M.: Biomimicry of Bacterial Foraging for Distributed Optimization and Control, IEEE Control Systems Magazine, 2002, pp. 52-67.

[2] Liu Y., Passino K. M.: Biomimicry of Social Foraging Bacteria for Distributed Optimization: Models, Principles, and Emergent Behaviors, Journal of Optimization Theory And Applications: **115**, 3, December 2002, pp. 603–628.

[3] Back T., Fogel D. B., Michalewicz, Z.: Handbook of Evolutionary Computation, IOP and Oxford University Press, Bristol, UK, 1997.

[4] Fogel D. B.: Evolutionary Computation, IEEE Press, Piscataway, NJ, 1995.

[5] Storn R., Price K.: Differential evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, Journal of Global Optimization, **11**, 4, 1997, pp. 341–359.

[6] Price K., Storn R., Lampinen J.: Differential Evolution – A Practical Approach to Global Optimization, Springer, ISBN: 3-540-20950-6, 2005.

[7] Holland J. H.: Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, 1975.

[8] Goldberg D. E., Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.

[9] Mishra S.: A hybrid least square-fuzzy bacterial foraging strategy for harmonic estimation. IEEE Trans. on Evolutionary Computation, **9**, 1, 2005, pp. 61-73.

[10] Tripathy M., Mishra S., Lai L. L., Zhang Q. P.: Transmission Loss Reduction Based on FACTS and Bacteria Foraging Algorithm. PPSN, 2006, pp. 222-231.

[11] Kim D. H., Cho C. H.: Bacterial Foraging Based Neural Network Fuzzy Learning. IICAI 2005, pp. 2030-2036.

[12] Mishra S., Bhende C. N.: Bacterial Foraging Technique-Based Optimized Active Power Filter for Load Compensation, IEEE Transactions on Power Delivery, **22**, 1, Jan. 2007, pp. 457–465.

[13] Kim D. H., Abraham A., Cho J. H.: A hybrid genetic algorithm and bacterial foraging approach for global optimization, Information Sciences, **177**, 18, 2007, pp. 3918-3937.

[14] Vesterstrøm J., Thomson R.: A Comparative Study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on Numerical Benchmark Problems, Proc. Sixth Congress on Evolutionary Computation (CEC-2004), IEEE Press.

[15] Lampinen J., Zelinka I.: On Stagnation of the Differential Evolution Algorithm, In: Ošmera, Pavel (ed.) (2000). Proceedings of MENDEL 2000, 6th International Mendel Conference on Soft Computing, June 7.–9. 2000, Brno, Czech Republic.

[16] Biswas A., Dasgupta S., Das S., Abraham A.: Synergy of PSO and Bacterial Foraging Optimization: A Comparative Study on Numerical Benchmarks, Second International Symposium on Hybrid Artificial Intelligent Systems (HAIS 2007), Advances in Soft computing Series, Springer Verlag, Germany, Corchado, E. et al. (Eds.): Innovations in Hybrid Intelligent Systems, ASC 44, 2007, pp. 255-263.

[17] Noman N., Iba H.: Enhancing differential evolution performance with local search for high dimensional function optimization, in Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, June 2005, pp. 967–974.

[18] Das S., Konar A., Chakraborty U. K.: An Improved Particle Swarm Optimization Algorithm for Faster Global Search, In: ACM-SIGEVO Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2005), Washington DC, June, 2005.

[19] Yao X., Liu Y., Lin G.: Evolutionary programming made faster, IEEE Transactions on Evolutionary Computation, **3**, 2, 1999, pp. 82-102.

[20] Angeline P. J.: Evolutionary optimization versus particle swarm optimization: Philosophy and the performance difference, Lecture Notes in Computer Science, Proceedings of 7^{th} International Conference on. Evolutionary Programming – Evolutionary Programming VII, **1447**, 1998, pp. 84-89.

[21] Mladenovic P., Kovacevic-Vuijcic C.: Solving spread-spectrum radar polyphase code design problem by tabu search and variable neighbourhood search, European Journal of Operational Research, **153**, 2003, pp. 389-399.