Journal of Computational Science 5 (2014) 312–323

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Self Balanced Differential Evolution

Harish Sharma [a,*], Jagdish Chand Bansal [b], K.V. Arya [a,1]

[a] ABV-Indian Institute of Information Technology and Management, Gwalior, India
[b] South Asian University, New Delhi, India

ARTICLE INFO

Article history:
Received 18 October 2012
Received in revised form
10 December 2012
Accepted 11 December 2012
Available online 28 December 2012

Keywords:
Evolutionary optimization
Differential Evolution
Cognitive Learning Factor
Meta-heuristics

ABSTRACT

Differential Evolution (DE) is a well known and simple population based probabilistic approach for global optimization. It has reportedly outperformed a few Evolutionary Algorithms (EAs) and other search heuristics like the Particle Swarm Optimization (PSO) when tested over both benchmark and real world problems. But, DE, like other probabilistic optimization algorithms, sometimes behave prematurely in convergence. Therefore, in order to avoid stagnation while keeping a good convergence speed for DE, two modifications are proposed: one is the introduction of a new control parameter, Cognitive Learning Factor (CLF) and the other is dynamic setting of scale factor. Both modifications are proposed in mutation process of DE. Cognitive learning is a powerful mechanism that adjust the current position of individuals by a means of some specified knowledge. The proposed strategy, named as Self Balanced Differential Evolution (*SBDE*), balances the exploration and exploitation capability of the DE. To prove efficiency and efficacy of *SBDE*, it is tested over 30 benchmark optimization problems and compared the results with the basic DE and advanced variants of DE namely, *SFLSDE*, *OBDE* and *jDE*. Further, a real-world optimization problem, namely, Spread Spectrum Radar Polly phase Code Design, is solved to show the wide applicability of the *SBDE*.

1. Introduction

Differential Evolution (DE) scheme is relatively a simple, fast and population based stochastic search technique, proposed by Storn and Price [33]. DE falls under the category of Evolutionary Algorithms (EAs). But in some sense it differs significantly from EAs, e.g., trial vector generation process (explained in Section 2) uses the information of distance and direction from current population to generate a new trial vector. Furthermore, in EAs, crossover is applied first to generate a trial vector, which is then used within the mutation operation to produce one offspring while, in DE, mutation is applied first and then crossover.

Researchers are continuously working to improve the performance of DE. Some of the recently developed versions of DE with appropriate applications can be found in [6]. Experiments over several numerical benchmarks [36] show that DE performs better than the Genetic algorithm (GA) [15] or the Particle Swarm Optimization (PSO) [17]. DE has successfully been applied to various areas of science and technology, such as chemical engineering [21], signal processing [9], mechanical engineering design [32], machine intelligence, and pattern recognition [27]. Recently, machine intelligence and cybernetics are most well-liked field in which DE algorithm has become a popular strategy.

There are two fundamental processes which drive the evolution of a DE population: the variation process, which enables exploring different areas of the search space, and the selection process, which ensures the exploitation of the previous experience. However, it has been shown that DE may occasionally stop proceeding towards the global optimum even though the population has not converged to a local optimum [19]. Therefore, to maintain the proper balance between exploration and exploitation behavior of DE, a new control parameter called Cognitive Learning Factor (*CLF*) is introduced in DE. Cognitive Learning is about enabling people to learn by using their reason, intuition and perception. This technique is often used to change people's behavior. The same phenomenon is also applied to the proposed DE in this paper. In the proposed DE, a weight factor (*CLF*) is associated with the individual's experience in the mutation operation. Furthermore, the range of scale factor F is also dynamically varied in the range devised after extensive experiments. The dynamic scale factor better controls the perturbation rate in mutation process. By varying *CLF* and F, exploration and exploitation capabilities of DE may be balanced. DE with these modifications is named as Self Balanced Differential Evolution (*SBDE*). Further, the proposed strategy is compared by

* Corresponding author. Tel.: +91 9479810157.
E-mail addresses: harish.sharma0107@gmail.com (H. Sharma), jcbansal@sau.ac.in (J.C. Bansal), kvarya@gmail.com (K.V. Arya).
[1] Tel.: +91 0751 2449819.

1877-7503/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.jocs.2012.12.002

experimenting on 30 test problems to the basic DE and its recent variants named, Scale Factor Local Search Differential Evolution (*SFLSDE*) [24], Opposition-Based Differential Evolution (*OBDE*) [31] and Dynamic Optimization using Self-Adaptive Differential Evolution (*jDE*) [5].

Rest of the paper is organized as follows: Section 2 describes brief overview of basic Differential Evolution algorithm. In Section 3 some basic Improvements in Differential Evolution algorithm are briefly reviewed. Self Balanced Differential Evolution algorithm (*SBDE*) is proposed and established in Section 4. In Section 5, a comprehensive set of experimental results are provided. In Section 6, a real-world optimization problem, Spread Spectrum Radar Polly phase Code Design problem is solved using *SBDE*. Finally, in Section 7, paper is concluded.

2. Brief overview of Differential Evolution algorithm

DE has several strategies based on method of selecting the target vector, number of difference vectors used and the type of crossover [28]. In this paper *DE/rand/1/bin* scheme is used where DE stands for differential evolution, 'rand' specifies that the target vector is selected randomly, '1' is for number of differential vectors and 'bin' notation is for*binomial* crossover. The popularity of Differential Evolution is due to its applicability to a wide class of problems and ease of implementation. Differential Evolution consists of the properties of both evolutionary algorithms and swarm intelligence. The detailed description of DE is as follows:

Like other population based search algorithms, in DE a population of potential solutions (individuals) searches the solution. In a D-dimensional search space, an individual is represented by a D-dimensional vector $(x_{i_1}, x_{i_2}, \ldots, x_{i_D})$, $i = 1, 2, \ldots, NP$ where NP is the population size (number of individuals).

In DE, there are three operators: mutation, crossover and selection. Initially, a population is generated randomly with uniform distribution then the mutation, crossover and selection operators are applied to generate a new population. Trial vector generation is a crucial step in DE process. The two operators mutation and crossover are used to generate the trial vectors. The selection operator is used to select the best trial vector for the next generation. DE operators are explained briefly in following subsections.

2.1. Mutation

A trial vector is generated by the DE mutation operator for each individual of the current population. For generating the trial vector, a target vector is mutated with a weighted differential. An offspring is produced in the crossover operation using the newly generated trial vector. If G is the index for generation counter, the mutation operator for generating a trial vector $u_i(G)$ from the parent vector $x_i(G)$ is defined as follows:

- Select a target vector $x_{i_1}(G)$ from the population such that $i \neq i_1$.
- Again, randomly select two individuals, x_{i_2} and x_{i_3}, from the population such that $i \neq i_1 \neq i_2 \neq i_3$.
- Then the target vector is mutated for calculating the trial vector as follows:

$$u_i(G) = x_{i_1}(G) + \underbrace{F \times \overbrace{(x_{i_2}(G) - x_{i_3}(G))}^{\text{Variation Component}}}_{\text{Step size}} \tag{1}$$

where $F \in [0, 1]$ is the mutation scale factor which is used in controlling the amplification of the differential variation [13].

2.2. Crossover

Offspring $x'_i(G)$ is generated using the crossover of parent vector, $x_i(G)$ and the trial vector, $u_i(G)$ as follows:

$$x'_{ij}(G) = \begin{cases} u_{ij}(G), & \text{if } j \in J \\ x_{ij}(G), & \text{otherwise.} \end{cases}$$

where J is the set of cross over points or the points that will go under perturbation, $x_{ij}(G)$ is the jth element of the vector $x_i(G)$.

Different methods may be used to determine the set J of which binomial crossover and exponential crossover are the most frequently used [13]. In this paper, the DE and its variants are implemented using the binomial crossover. In this crossover, the crossover points are randomly selected from the set of possible crossover points, $\{1, 2, \ldots, D\}$, where D is the problem dimension. Algorithm 1 shows the steps of binomial crossover to generate crossover points [13]. In this algorithm, CR is the probability that the considered crossover point will be included. The larger the value of CR, the more crossover points will be selected.

Algorithm 1 *(Binomial Crossover).*

```
J = $\phi$
$j^* \sim U(1, D)$;
$J \leftarrow J \cup j^*$;
for each $j \in 1 \ldots D$ do
    if $U(0, 1) < CR$ and $j \neq j^*$ then
        $J \leftarrow J \cup j$;
    end if
end if
```

Here, J is a set of crossover points, CR is crossover probability, $U(1, D)$ is a uniformly distributed random integer between 1 and D.

2.3. Selection

There are two functions of the selection operator: First it selects the individual for the mutation operation to generate the trial vector and second, it selects the best, between the parent and the offspring based on their fitness value for the next generation. If fitness of parent is greater than the offspring then parent is selected otherwise offspring is selected:

$$x_i(G+1) = \begin{cases} x'_i(G), & \text{if } f(x'_i(G)) > f(x_i(G)). \\ x_i(G), & \text{otherwise.} \end{cases}$$

This ensures that the population's average fitness does not deteriorate.

The Pseudo-code for Differential Evolutionary strategy, is described as follows [13]:

Algorithm 2 *(Differential Evolutionary Algorithm).*

```
Initialize the control parameters $F$ and $CR$;
Create and initialize the population $P(0)$ of $NP$ individuals;
while stopping condition(s) not true do
    for each individual, $x_i(G) \in P(G)$ do
        Evaluate the fitness, $f(x_i(G))$;
        Create the trial vector $u_i(G)$ by applying the mutation operator;
        Create an offspring $x'_i(G)$ by applying the crossover operator;
        if $f(x'_i(G))$ is better than $f(x_i(G))$ then
            Add $x'_i(G)$ to $P(G+1)$;
        else
            Add $x_i(G)$ to $P(G+1)$;
        end if
    end for
end while
Return the individual with the best fitness as the solution;
```

Here, F (scale factor) and CR (crossover probability) are the control parameters and influence the performance of the DE. P is the population vector.

3. Brief review on basic Improvements in Differential Evolution

In order to get rid of the drawbacks of basic DE, researchers have improved DE in many ways. The potentials where DE can be improved may be broadly classified into two categories:

1. Hybridization of DE with other population based probabilistic or deterministic algorithms
2. Introducing a new mechanisms for controlling the evolution, which may require new parameters and/or fine tuning of DE control parameters *NP*, *F*, *CR*.

This paper concentrates on the second category of DE research, i.e., the paper introduces a new mechanisms for controlling the evolution, which requires a new control parameter, namely, Cognitive Learning Factor in DE process. In past many efforts have been done for improving the performance of DE based on this category [10]. Some important strategies based on this category are briefly reviewed as follows:

Storn and Price [33] have observed that the value of *F* should be in the range of [0.5, 1]. The value of *NP* should be in the range of [5*D*, 10*D*], where *D* is the dimension of the problem.

Fuzzy Adaptive Differential Evolution (*FADE*) is introduced by Liu and Lampinen [20]. It is based on the fuzzy logic controllers, whose inputs incorporate the relative function values and individuals of successive generations to adapt the parameters for the mutation and crossover operation. They proved by the experimental results over a set of benchmark functions that the FADE algorithm performance is better than the conventional DE algorithm.

Gamperle et al. [14] determined different parameter values for DE specially for the Sphere, Rastrigin' and Rosenbrock' functions. They proved that the global optimum searching capability and the convergence speed are very sensitive for the values of the control parameters *NP*, *F*, and *CR*. They specified that the population size $NP \in [3D, 8D]$, with the scaling factor $F = 0.6$ and the crossover rate *CR* in [0.3, 0.9] are the good choice for the parameter setting.

Zaharie proposed a parameter adaptation strategy for DE (ADE) [40] which is based on controlling the population diversity. In ADE, multi-population approach is also implemented. Furthermore, Zaharie and Petcu introduced an adaptive Pareto DE algorithm, based on the same line of thinking, for multi-objective optimization [41].

Abbass [1] proposed a self-adapted strategy for crossover rate *CR* to solve multi-objective optimization problems. In Abbass strategy, *CR* value is encoded into each individual, simultaneously evolved with other search variables. There was a different scale factor *F*, uniform distributed in [0, 1], for each variable.

Furthermore, Qin et al. [30] introduced a Self-adaptive DE (*SaDE*) algorithm, in which all the control parameters that are used in the trial vector generation strategies and selection process are steadily self-adapted by learning from their previous experiences.

Brest et al. [4] investigate a Self-Adaptive Differential Evolution algorithm (*jDE*) where *F* and *CR* control parameters are self-adapted and a multi-population method with aging mechanism is used. Further, Zhang and Sanderson [42] proposed a new variant of DE named JADE: adaptive differential evolution with optional external archive, in which they introduced a new self adaptive strategy for *F* and *CR* control parameters.

Omran et al. [26] introduced a self-adaptive scaling factor *F*. They generated the value of *CR* for each individual from a normal distribution *N*(0.5, 0.15). This approach (called SDE) was tested on four benchmark functions and verified to be performed better than other versions of DE.

Besides, setting the control parameters (*F* and *CR*), some researchers also tuned the population size (*NP*) for improving the performance. Teo introduced a variant of DE which is based on the idea of Self Adapting Populations (DESAP) [35].

Noman and Iba [25] introduced a crossover-based local search method for DE called the Fittest Individual Refinement (FIR). An exploration capability of DE is hastened by the FIR scheme as it enhances DE's search capability in the neighborhood for the best solution in successive generations.

Furthermore, Yan et al. [39] proposed a new variant of DE called Simulated Annealing Differential Evolution (*SADE*). In SADE algorithm, each individual contains a set of *F* values instead of single value within the range [0.1, 1], control parameters *F* and *CR* are encoded into individual and their values are changed, based on the two new probability factors τ_1 and τ_2. *F* is reinitialized with the probability τ_1 by a random value otherwise it remains unchanged. The crossover probability *CR* also reinitialized with probability τ_2 and within the range [0, 1]. *CR* is assigned to each individual but in an identical fashion. *CR* changes its value with probability τ_2 with a random value otherwise it remains unchanged for the next generation.

Neri and Tirronen [24] proposed a self-adaptive strategy called Scale Factor Local Search Differential Evolution (*SFLSDE*) strategy. *SFLSDE* is a self-adaptive scheme with the two local search algorithms: Scale factor hill-climb and Scale factor golden section search. These local search algorithms are used for detecting the value of scale factor *F* corresponding to an offspring with a better performance. Therefore, the local search algorithms support in the global search (exploration process) and in generating offspring with high performance.

Das et al. [8] proposed a new variant of Differential Evolution algorithm called Differential Evolution Using a Neighborhood-Based Mutation Operator (*DEGL*). The proposed scheme balances the exploration and exploitation abilities of DE. *DEGL* introduces four new control parameters: α, β, w, and the neighborhood radius k. In *DEGL*, w is the most important parameter as it controls the balance between the exploration and exploitation capabilities. It is shown in the following expression.

$$V = w \times Global + (1 - w) \times Local$$

here, $w \in [0, 1]$. Small values of w favor the local neighborhood component, thereby resulting in better exploration. On the other hand, large values favor the global variant component, encouraging exploitation. Therefore, values of w near about 0.5 result the most balanced *DEGL* version.

Weber et al. [37] introduced scale factor inheritance mechanism in distributed differential evolution algorithm. In the proposed algorithm, the population is distributed over several sub-populations allocated according to a ring topology. Each sub-population having its own scale factor value. In addition, a perturbation mechanism also introduced which enhances the exploration feature of the algorithm.

4. Self Balanced Differential Evolution

4.1. A few drawbacks of DE

The inherent drawback with most of the population based stochastic algorithms is premature convergence. DE is not an exception. Any population based algorithm is regarded as an efficient algorithm if it is fast in convergence and able to explore the maximum area of the search space. In other words, if a population based algorithm is capable of balancing between exploration and exploitation of the search space, then the algorithm is regarded as an efficient algorithm. From this point of view, basic DE is not an efficient algorithm [22]. Also some studies proved that stagnation is another inherent drawback with DE i.e. DE sometimes stop proceeding towards the global optima even though the population has not converged to local optima or any other point [19]. Mezura-Montes et al. [22] compared the different variants of DE for global optimization and found that DE shows a poor performance and remains inefficient in exploring the search space, especially for multimodal functions. Price et al. [29] also drawn the same conclusions. The problems of premature convergence and stagnation is a matter of serious consideration for designing a comparatively efficient[2] DE algorithm.

4.2. Motivation for Cognitive Learning Factor and Dynamic Scale Factor

Exploration of the whole search space and exploitation of the near optimal solution region may be balanced by maintaining the diversity in early and later iterations of any random number based search algorithm. Mutation equation (1) in DE may be seen in the following way:

$$u_i(G) = A \times x_i(G) + B \times \overbrace{(x_{i_2}(G) - x_{i_3}(G))}^{\text{Variation Component}}$$

i.e., the trial vector $u_i(G)$ is the weighted sum of target vector $x_i(G)$ and the difference $(x_{i_2}(G) - x_{i_3}(G))$ of two random vectors. Here, A is the weight to target vector and B is the weight to the difference of random vectors. In basic DE, A is set to be 1, while B is the scaling factor F varied in range $(0, \infty)$. Studies have been carried out with varying scaling factor F [4] for better exploration and exploitation mechanism. In this paper, experiments are performed for finding an optimal strategy to set the weight A and B. The weight A is named as Cognitive Learning Factor (*CLF*) and denoted by '*C*' (for this study). *CLF* is the weight to individual's current position or in other words this is the weight to self confidence and therefore, it is named so.

The modified mutation operation of DE becomes:

$$u_i(G) = C \times x_{i_1}(G) + F \times (x_{i_2}(G) - x_{i_3}(G)) \tag{2}$$

Symbols have their usual meaning. It is clear from Eq. (2) that small value of C and large value of F, increase the exploration capability as the weight for individual's current position is low whereas weight for variation component is high. Furthermore, large value of C and small value of F, increase exploitation capability as in this case, weight to individual's current position is high whereas weight to variation component is low. Therefore, a proper managed C and F can balance the diversity in the population. So, it is expected that these modifications should improve the results. The proposed strategy dynamically balances the exploration and exploitation capability of DE and therefore the proposed variant of DE is named as Self Balanced Differential Evolution (*SBDE*).

SBDE introduces one new parameter, C called the Cognitive Learning Factor (CLF). Cognitive Learning Factor C is the most important parameter in *SBDE* as it controls the balance between the exploration and exploitation capabilities of the algorithm. C varies between 0.1 and 1 using Eq. (3):

$$C_i(G+1) = \begin{cases} C_i(G) + prob_i(G) & \text{if } \ fitness_i(G) > fitness_i(G-1) \ \text{ and } \ C_i(G) < 1 \\ 1 & \text{if } \ fitness_i(G) > fitness_i(G-1) \ \text{ and } \ C_i(G) \geq 1 \\ C_i(G) & \text{if } \ fitness_i(G) < fitness_i(G-1) \ \text{ and } \ trial_i \leq limit \\ 0.1 & \text{if } \ fitness_i(G) < fitness_i(G-1) \ \text{ and } \ trial_i > limit \end{cases} \tag{3}$$

In Eq. (3), C_i is the CLF for ith individual, G is the iteration counter, $fitness_i$ is the fitness of the ith individual, $trial_i$ is the counter to count number of times the individual is not updated, *limit* is the maximum allowable counts in which an individual is not updated and $prob_i(G)$ is calculated in Eq. (4). In Eq. (4) $maxfit(G)$ is the maximum fitness of the solutions in Gth iteration.

$$prob_i(G) = \frac{0.9 \times fitness_i(G)}{maxfit(G)} + 0.1, \tag{4}$$

It should be noted that at any time, coefficient C is the function of fitness of an individual X, simultaneously, applied to update individual X and therefore it is named as Cognitive Learning Factor.

SBDE also proposed a dynamic scale factor (F) whose value is calculated using Eq. (5).

$$F_i(G+1) = (rand(0,1) - 0.5) \times (1.5 - prob_i(G)), \tag{5}$$

In SBDE, in any iteration, the individuals of the population works two folded: exploration and exploitation. Usually, large step size produces exploration while small step size exploitation. From Eqs. (4) and (5), it can be seen that $prob_i$ and so CLF is proportional to the $fitness_i$ while F_i decreases if $fitness_i$ increases. Thus, at any time, if the weight to the individual's self confidence (first term of Eq. 2) increases, weight to the step size (second term) decreases and therefore at some level, maintains the overall ratio of weights.

Due to Eqs. (3) and (5), in SBDE, better individual will be allowed only small step sizes which force individual to exploit the search space, provided it is kept on updating itself. In other words, the task of exploitation is assigned to a better individual. In case, a better individual could not update itself for a certain number of trials (named as *limit*) then it starts to explore ($C_i(G+1) = 0.1$, if $fitness_i(G) < fitness_i(G-1)$ and $trial_i > limit$). Now the job of exploration is mainly, handled by worse fit individuals in SBDE. Because for worse individuals, the value of F_i will be large, hence the step size will be given higher weight as compare to the individual's current position. In short, the exploration and exploitation are maintained by worse and better fit individuals, respectively. There is no hard line which divides the exploring and exploiting individuals, it is the fitness which transit a exploring individual to exploiting individual and vice-versa.

The Self Balanced DE algorithm (*SBDE*) is similar to the basic DE algorithm except the mutation operation. The Pseudo-code for the *SBDE* algorithm is shown in Algorithm 3.

[2] As it is not possible to design a fully efficient population based stochastic algorithm.

Algorithm 3 *(Self Balanced Differential Evolution (SBDE)).*

```
Initialize the control parameters, $F$, $CR$ and $C$;
Initialize Generation counter $G = 1$;
Create and initialize the population, $P(0)$, of $NP$ individuals;
Calculate fitness and then probabilities of the individuals using Eq. (4).
while stopping condition(s) not true do
  for each individual, $x_i(G) \in P(G)$ do
    Evaluate the fitness, $f(x_i(G))$;
    Calculate the probability $prob_i(G)$ using Eq. (4).
    Create the trial vector $u_i(G)$ by applying the scale factor (mutation operator) $F$ and Cognitive Learning Factor $C$ as follows;
      $u_i(G) = C_i(G) \times x_{i_1}(G) + F_i(G) \times (x_{i_2}(G) - x_{i_3}(G))$;
    Create an offspring $x'_i(G)$ by applying the binomial crossover;
    if $f(x'_i(G))$ is better than $f(x_i(G))$ then
      $trial_i = 0$
      $C_i(G+1) = C_i(G) + prob_i(G)$;
      if $C_i(G+1) > 1$ then
        $C_i(G+1) = 1$;
      end if
      Add $x'_i(G)$ to $P(G+1)$;
    else
      $trial_i = trial_i + 1$
      if $trial_i \geq limit$ then
        Randomly initialize the individual $x_i(G)$;
        $C_i(G+1) = 0.1$;
      else
        $C_i(G+1) = C_i(G)$;
      end if
      Add $x_i(G)$ to $P(G+1)$;
    end if
    $F_i(G+1) = (rand(0, 1) - 0.5) \times (1.5 - prob_i(G))$;
  end for
end while
Return the individual with the best fitness as the solution;
```

It is worth mentioning that, SBDE has an additional advantage of avoiding stagnation due to re-initialization of individual's position when it shows no improvement for a certain number of trials, refer following in the Algorithm 3:

```
..................
$trial_i = trial_i + 1$
if $trial_i \geq limit$ then
  Randomly initialize the individual $x_i(G)$;
end if
..................
```

5. Experimental results and discussion

5.1. Test problems under consideration

In order to see the effect of C and dynamic scaling factor F on DE, 30 different global optimization problems (f_1 to f_{30}) are selected (listed in Table 1). These problems are minimization problems and have different degrees of complexity and multimodality. Test problems f_1–f_{20} and f_{27}–f_{30} are taken from [3], test problems f_{21}–f_{26} are taken from [34] with the associated offset values.

5.2. Experimental setting

To test *SBDE* over test problems, following experimental setting is adopted:

- The scale factor $F = 0.5$ [28],
- Population size $NP = 50$,
- The stopping criteria is either maximum number of function evaluations (which is set to be 2.0×10^5) is reached or the corresponding acceptable error (mentioned in Table 1) has been achieved.
- $limit = D \times NP/2$ inspired from [16,2],
- The number of simulations =100,
- In order to investigate the effect of the parameter CR on the performance of *SBDE*, its sensitivity with different values of CR in the range [0.1, 1], is examined in Fig. 1. This figure shows the graph between different values of CR and corresponding sum of average number of function evaluations for 30 problems in meeting the termination criteria for SBDE. It is clear from Fig. 1 that SBDE is very sensitive for CR and value 0.4 gives comparatively better results. Therefore $CR = 0.4$ is selected for the experiments in this paper.

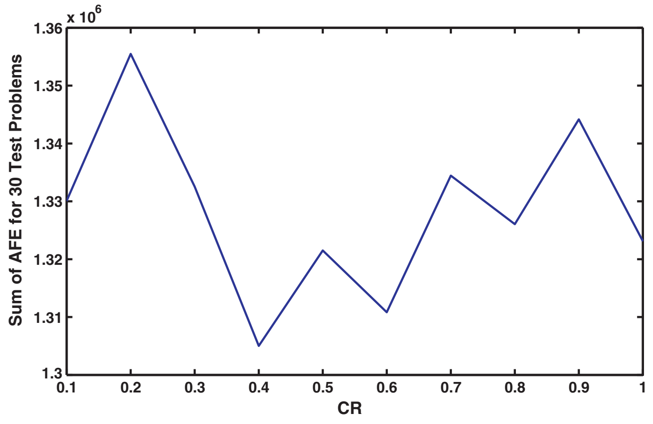

Fig. 1. Effect of parameter *CR* on average function evaluations.

- Parameters for the basic DE are $CR = 0.8$, $F = 0.5$ [28,33].
- Parameter settings for the algorithms *SFLSDE*, *OBDE* and *jDE* are similar to their original research papers.

5.3. Results and discussion

Numerical results with experimental setting of Section 5.2 are given in Table 2. In Table 2, standard deviation (*SD*), mean error (*ME*), average function evaluations (*AFE*), and success rate (*SR*) are reported. Table 2 shows that most of the time *SBDE* outperforms in terms of reliability, efficiency and accuracy as compare to the *DE/rand/bin/1*, *SFLSDE*, *OBDE* and *jDE*. Some more intensive statistical analysis based on *t* test, Acceleration Rate (AR) [31], Average Distance Around Swarm Center [18], Boxplot and Performance Index [12] have been carried out for results of *SBDE*, *DE*, *SFLSDE*, *OBDE* and *jDE*.

5.3.1. Statistical analysis

The *t*-test is quite popular among researchers in the field of evolutionary computing. In this paper Student's *t*-test is applied according to the description given in [7] for a confidence level of 0.95. Table 3 shows the results of the *t*-test for the null hypothesis that there is no difference in the mean number of function evaluations of 100 runs using *SBDE*, *DE*, *SFLSDE*, *OBDE* and *jDE*. Note that here '+' indicates the significant difference (or the null hypothesis is rejected) at a 0.05 level of significance, '–' implies that there is no significant difference while '=' indicates that comparison is not possible. In Table 3, *SBDE* is compared with the *DE*, *SFLSDE*, *OBDE* and *jDE*. The last row of Table 3, establishes the superiority of *SBDE* over *DE*, *SFLSDE*, *OBDE* and *jDE*.

Further, a comparison is made on the basis of convergence speed of the considered algorithms by measuring the average function evaluations (AFEs). A smaller AFEs means higher convergence speed. In order to minimize the effect of the stochastic nature of the algorithms, the reported function evaluations for each test problem is the average over 100 runs. In order to compare convergence speeds, we use Acceleration Rate (*AR*) which is defined as follows, based on the AFEs for the two algorithms *ALGO* and *SBDE*:

$$AR = \frac{AFE_{ALGO}}{AFE_{SBDE}}, \tag{6}$$

where $ALGO \in \{DE, SFLSDE, OBDE, jDE\}$ and $AR > 1$ means *SBDE* converges faster. Table 4 shows a clear comparison between *SBDE* – *DE*,

Table 1
Test problems.

Test problem	Objective function	Search range	Optimum value	D	Acceptable error
Sphere	$f_1(x)=\sum_{i=1}^{n} x_i^2$	[−5.12,5.12]	$f(\mathbf{0})=0$	30	1.0E−05
Griewank	$f_2(x)=1+\frac{1}{4000}\sum_{i=1}^{D} x_i^2-\prod_{i=1}^{D}\cos\left(\frac{x_i}{\sqrt{i}}\right)$	[−600,600]	$f(\mathbf{0})=0$	30	1.0E−05
Rosenbrock	$f_3(x)=\sum_{i=1}^{D}(100(x_{i+1}-{x_i}^2)^2+(x_i-1)^2)$	[−30,30]	$f(\mathbf{1})=0$	30	1.0E−02
Rastrigin	$f_4(x)=10D+\sum_{i=1}^{D}[x_i^2-10\cos(2\pi x_i)]$	[−5.12,5.12]	$f(\mathbf{0})=0$	30	1.0E−05
Ackley	$f_5(x)=-20+e+\exp\left(-\frac{0.2}{D}\sqrt{\sum_{i=1}^{D}{x_i}^3}\right)-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)x_i\right)$	[−1,1]	$f(\mathbf{0})=0$	30	1.0E−05
Alpine	$f_6(x)=\sum_{i=1}^{n}\lvert x_i\sin x_i+0.1x_i\rvert$	[−10,10]	$f(\mathbf{0})=0$	30	1.0E−05
Cosine Mixture	$f_7(x)=\sum_{i=1}^{D}{x_i}^2-0.1\left(\sum_{i=1}^{D}\cos 5\pi x_i\right)+0.1D$	[−1,1]	$f(\mathbf{0})=-D\times 0.1$	30	1.0E−05
Exponential	$f_8(x)=-\left(\exp\left(-0.5\sum_{i=1}^{D}{x_i}^2\right)\right)+1$	[−1,1]	$f(\mathbf{0})=-1$	30	1.0E−05
Cigar	$f_9(x)={x_0}^2+100,000\sum_{i=1}^{D}{x_i}^2$	[−10,10]	$f(\mathbf{0})=4$	30	1.0E−05
brown3	$f_{10}(x)=\sum_{i=1}^{D-1}\left({x_i}^{2(x_{i+1})^2+1}+{x_{i+1}}^{2{x_i}^2+1}\right)$	[−1,4]	$f(\mathbf{0})=0$	30	1.0E−05
Schewel	$f_{11}(x)=\sum_{i=1}^{D}\lvert x_i\rvert+\prod_{i=1}^{D}\lvert x_i\rvert$	[−10,10]	$f(\mathbf{0})=0$	30	1.0E−05
Axis parallel hyper-ellipsoid	$f_{12}(x)=\sum_{i=1}^{D} ix_i^2$	[−5.12,5.12]	$f(\mathbf{0})=0$	30	1.0E−05
Sum of different powers	$f_{13}(x)=\sum_{i=1}^{D}\lvert x_i\rvert^{i+1}$	[−1,1]	$f(\mathbf{0})=0$	30	1.0E−05
Step function	$f_{14}(x)=\sum_{i=1}^{D}(\lfloor x_i+0.5\rfloor)^2$	[−100,100]	$f(-0.5\le x\le 0.5)=0$	30	1.0E−05
Inverted cosine wave	$f_{15}(x)=-\sum_{i=1}^{D-1}\left(\exp\left(\frac{-(x_i^2+x_{i+1}^2+0.5x_ix_{i+1})}{8}\right)\times \mathrm{I}\right)$, where $I=\cos\left(4\sqrt{x_i^2+x_{i+1}^2+0.5x_ix_{i+1}}\right)$	[−5,5]	$f(\mathbf{0})=-D+1$	10	1.0E−05

Table 1 (*Continued*)

Test problem	Objective function	Search range	Optimum value	D	Acceptable error
Rotated hyper-ellipsoid	$f_{16}(x)=\sum_{i=1}^{D}\sum_{j=1}^{i}x_j^2$	[−65.536,65.536]	$f(\mathbf{0})=0$	30	1.0E−05
Levy montalvo 1	$f_{17}(x)=\frac{\pi}{D}(10\sin^2(\pi y_1)+\sum_{i=1}^{D-1}(y_i-1)^2\times(1+10\sin^2(\pi y_{i+1}))+(y_D-1)^2),\quad \text{where}\quad y_i=1+\frac{1}{4}(x_i+1)$	[−10,10]	$f(-\mathbf{1})=0$	30	1.0E−05
Levy montalvo 2	$f_{18}(x)=0.1(\sin^2(3\pi x_1)+\sum_{i=1}^{D-1}(x_i-1)^2\times(1+\sin^2(3\pi x_{i+1}))+(x_D-1)^2(1+\sin^2(2\pi x_D))$	[−5,5]	$f(\mathbf{1})=0$	30	1.0E−05
Ellipsoidal	$f_{19}(x)=\sum_{i=1}^{D}(x_i-i)^2$	[−D,D]	$f(1,2,3,\ldots,D)=0$	30	1.0E−05
2D Tripod	$f_{20}(x)= p(x_2)(1+p(x_1))+\lvert(x_1+50p(x_2)(1-2p(x_1)))\rvert+\lvert(x_2+50(1-2p(x_2)))\rvert$	[−100,100]	$f(0,-50)=0$	2	1.0E−04
Shifted Rosenbrock	$f_{21}(x)=\sum_{i=1}^{D-1}(100(z_i^2-z_{i+1})^2+(z_i-1)^2)+f_{bias},\quad z=x-o+1,x=[x_1,x_2,\ldots,x_D],o=[o_1,o_2,\ldots,o_D]$	[−100,100]	$f(o)=f_{bias}=390$	10	1.0E−01
Shifted Sphere	$f_{22}(x)=\sum_{i=1}^{D}z_i^2+f_{bias},z=x-o,x=[x_1,x_2,\ldots,x_D],o=[o_1,o_2,\ldots,o_D]$	[−100,100]	$f(o)=f_{bias}=-450$	10	1.0E−05
Shifted Rastrigin	$f_{23}(x)=\sum_{i=1}^{D}(z_i^2-10\cos(2\pi z_i)+10)+f_{bias},\quad z=(x-o),x=(x_1,x_2,\ldots,x_D),o=(o_1,o_2,\ldots,o_D)$	[−5,5]	$f(o)=f_{bias}=-330$	10	1.0E−02
Shifted Schwefel	$f_{24}(x)=\sum_{i=1}^{D}\left(\sum_{j=1}^{i}z_j\right)^2+f_{bias},\quad z=x-o,x=[x_1,x_2,\ldots,x_D],o=[o_1,o_2,\ldots,o_D]$	[−100,100]	$f(o)=f_{bias}=-450$	10	1.0E−05
Shifted Griewank	$f_{25}(x)=\sum_{i=1}^{D}\frac{z_i^2}{4000}-\prod_{i=1}^{D}\cos\left(\frac{z_i}{\sqrt{i}}\right)+1+f_{bias},\quad z=(x-o),x=[x_1,x_2,\ldots,x_D],o=[o_1,o_2,\ldots,o_D]$	[−600,600]	$f(o)=f_{bias}=-180$	10	1.0E−05
Shifted Ackley	$f_{26}(x)=-20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}z_i^2}\right)-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi z_i)\right)+20+e+f_{bias},\quad z=(x-o),x=(x_1,x_2,\ldots,x_D),o=(o_1,o_2,\ldots,o_D)$	[−32,32]	$f(o)=f_{bias}=-140$	10	1.0E−05
Six-hump camel back	$f_{27}(x)=\left(4-2.1x_1^2+\frac{x_1^4}{3}\right)x_1^2+x_1x_2+(-4+4x_2^2)x_2^2$	[−5,5]	$f(-0.0898,0.7126)=-1.0316$	2	1.0E−05
Easom's function	$f_{28}(x)=-\cos x_1\cos x_2 e^{((-(x_1-\pi)^2-(x_2-\pi)^2))}$	[−10,10]	$f(\pi,\pi)=-1$	2	1.0E−13
Dekkers and Aarts	$f_{29}(x)=10^5x_1^2+x_2^2-(x_1^2+x_2^2)^2+10^{-5}(x_1^2+x_2^2)^4$	[−20,20]	$f(0,15)=f(0,-15)=-24,777$	2	5.0E−01
Moved axis parallel hyper-ellipsoid	$f_{30}(x)=\sum_{i=1}^{D}5i\times x_i^2$	[−5.12,5.12]	$f(x)=0\,;x(i)=5\times i,i=1:D$	30	1.0E−15

Table 2
Comparison of the results of *SBDE*, *DE*, *SFLSDE*, *OBDE* and *jDE* problems.

Test problem	Algorithm	SD	ME	AFE	SR
f_1	SBDE	8.12E−07	9.03E−06	15,745.5	100
	DE	8.34E−07	9.06E−06	22,869.5	100
	SFLSDE	8.65E−07	9.00E−06	24,626.15	100
	OBDE	9.01E−06	8.70E−06	19,743	100
	jDE	9.13E−06	8.07E−06	23,772.5	100
f_2	SBDE	7.44E−07	9.10E−06	23,261.5	100
	DE	7.39E−07	8.90E−06	34,032	100
	SFLSDE	0.001034175	1.57E−04	39,382.59	98
	OBDE	2.20E−03	4.78E−03	63,289.5	80
	jDE	3.29E−04	1.58E−03	40,953.5	96
f_3	SBDE	2.63E+01	4.16E+01	200,050.04	0
	DE	1.87E+01	3.05E+01	200,050	0
	SFLSDE	28.22646992	23.45233068	199,640.34	1
	OBDE	3.53E+01	2.97E+01	200,006	0
	jDE	3.65E+01	2.49E+01	200,050	0
f_4	SBDE	1.39E−01	1.99E−02	97,521.67	98
	DE	4.72E+00	4.43E+01	200,050	0
	SFLSDE	0.098996267	0.009958634	154,474.92	98
	OBDE	1.16E+01	4.68E+00	168,575	28
	jDE	9.00E−06	7.71E−06	169,175	100
f_5	SBDE	4.46E−07	9.50E−06	29,567	100
	DE	4.59E−07	9.53E−06	42,954.5	100
	SFLSDE	4.20E−07	9.51E−06	46,037.98	100
	OBDE	9.51E−06	4.13E−06	38,855.5	100
	jDE	9.50E−06	4.97E−06	44,502.5	100
f_6	SBDE	5.26E−07	9.41E−06	70,845	100
	DE	1.87E−03	3.36E−03	199,143.5	3
	SFLSDE	3.08E−05	1.89E−05	169,798.17	74
	OBDE	1.80E−05	8.55E−04	71,880	99
	jDE	3.84E−05	4.61E−04	182,548.5	49
f_7	SBDE	6.85E−07	9.16E−06	14,882	100
	DE	8.06E−07	9.02E−06	22,833.5	100
	SFLSDE	6.42E−07	9.09E−06	24,039	100
	OBDE	3.00E+00	8.69E−06	21,317	100
	jDE	3.00E+00	7.89E−06	23,189	100
f_8	SBDE	8.38E−07	9.05E−06	12,007.5	100
	DE	8.61E−07	8.96E−06	17,381.5	100
	SFLSDE	7.03E−07	9.18E−06	18,628.31	100
	OBDE	1.00E+00	7.57E−06	14,757.5	100
	jDE	1.00E+00	9.62E−06	18,144.5	100
f_9	SBDE	8.11E−07	8.98E−06	28,020	100
	DE	7.25E−07	9.12E−06	39,903	100
	SFLSDE	8.68E−07	9.04E−06	43,018.14	100
	OBDE	8.92E−06	8.81E−06	35,306	100
	jDE	9.11E−06	8.01E−06	41,512.5	100
f_{10}	SBDE	8.02E−07	8.98E−06	14,539	100
	DE	8.06E−07	9.01E−06	22,506.5	100
	SFLSDE	8.75E−07	8.99E−06	25,713.07	100
	OBDE	3.50E−05	2.59E−04	21,923	99
	jDE	9.07E−06	6.42E−06	23,349.5	100
f_{11}	SBDE	4.89E−07	9.44E−06	25,199.5	100
	DE	4.01E−07	9.44E−06	39,516	100
	SFLSDE	5.60E−07	9.42E−06	42,894.88	100
	OBDE	9.36E−06	4.93E−06	44,695.5	100
	jDE	9.45E−06	5.31E−06	41,454	100
f_{12}	SBDE	7.49E−07	9.04E−06	18,264.5	100
	DE	8.48E−07	8.90E−06	26,248.5	100
	SFLSDE	1.00E−06	8.91E−06	28,160.88	100
	OBDE	8.90E−06	8.62E−06	22,715	100
	jDE	9.14E−06	7.39E−06	27,346.5	100
f_{13}	SBDE	1.93E−06	7.25E−06	4541.5	100
	DE	1.93E−06	7.34E−06	8613	100
	SFLSDE	2.08E−06	7.24E−06	8987.38	100
	OBDE	6.85E−06	2.25E−06	5244	100
	jDE	7.63E−06	1.65E−06	8451	100
f_{14}	SBDE	1.00E−06	1.00E−06	10,038.5	100
	DE	1.00E−06	1.00E−06	15,277.5	100
	SFLSDE	1.00E−06	1.00E−06	16,291.31	100
	OBDE	1.10E−01	3.43E−01	32,885	90
	jDE	1.00E−06	1.00E−06	15,684	100
f_{15}	SBDE	1.40E−06	8.20E−06	31,051.95	100
	DE	1.52E−06	7.97E−06	78,966.5	100
	SFLSDE	0.833647153	0.687539673	124,007.97	48
	OBDE	8.38E+00	6.82E−01	128,968	40
	jDE	8.98E+00	8.94E−02	67,464	97
f_{16}	SBDE	7.18E−07	9.09E−06	23,261.5	100
	DE	7.57E−07	9.15E−06	33,244.5	100
	SFLSDE	8.59E−07	9.04E−06	35,789.49	100
	OBDE	8.98E−06	8.30E−06	28,976.5	100
	jDE	9.10E−06	8.24E−06	34,582	100
f_{17}	SBDE	8.02E−07	9.03E−06	13,372.5	100
	DE	8.00E−07	9.02E−06	21,650.5	100
	SFLSDE	7.23E−07	9.12E−06	22,077.77	100
	OBDE	9.26E−06	6.79E−06	18,018.5	100
	jDE	9.02E−06	8.47E−06	21,161.5	100
f_{18}	SBDE	5.81E−07	9.26E−06	14,177.5	100
	DE	7.82E−07	9.02E−06	20,690	100
	SFLSDE	7.97E−07	9.03E−06	22,316.15	100
	OBDE	1.19E−04	1.09E−03	20,251.5	99
	jDE	9.02E−06	9.47E−06	21,361.5	100
f_{19}	SBDE	8.11E−07	9.00E−06	19,446	100
	DE	7.75E−07	9.12E−06	26,477.5	100
	SFLSDE	8.23E−07	9.05E−06	30,151.57	100
	OBDE	9.12E−06	7.75E−06	26,386	100
	jDE	8.96E−06	9.36E−06	28,052	100
f_{20}	SBDE	2.29E−07	6.57E−07	10,626.54	100
	DE	4.21E−01	2.30E−01	54,221.5	77
	SFLSDE	3.57E−01	1.50E−01	34,966.26	85
	OBDE	2.00E−02	1.40E−01	8251	98
	jDE	1.40E−01	3.47E−01	34,095.5	86
f_{21}	SBDE	3.02E+00	1.52E+00	131,145.1	56
	DE	5.70E−01	2.93E−01	135,066	79
	SFLSDE	3.87E−01	1.32E−01	58,047.64	99
	OBDE	3.95E+02	1.30E+00	138,293.5	89
	jDE	3.90E+02	9.24E−01	76,226	94
f_{22}	SBDE	1.42E−06	8.02E−06	8968	100
	DE	1.70E−06	7.93E−06	10,415	100
	SFLSDE	1.59E−06	7.91E−06	12,199.55	100
	OBDE	4.50E−04	1.54E−06	9840.5	100
	jDE	4.50E−04	1.35E−06	11,777.5	100
f_{23}	SBDE	1.40E+01	1.21E+02	200,676.6	0
	DE	1.48E+01	1.05E+02	200,050	0
	SFLSDE	1.46E+01	1.17E+02	199,769.38	0
	OBDE	2.50E+02	1.34E+02	200,004	0
	jDE	2.33E+02	1.29E+02	200,050	0
f_{24}	SBDE	6.77E+03	2.51E+04	200,590.27	0
	DE	6.03E+03	1.86E+04	200,050	0
	SFLSDE	6.15E+03	2.33E+04	199,772.01	0
	OBDE	9.18E+03	4.14E+03	200,005.5	0
	jDE	1.45E+04	4.73E+03	200,050	0
f_{25}	SBDE	1.79E−06	7.96E−06	19,403.16	100
	DE	1.60E−06	8.11E−06	27,278	100
	SFLSDE	7.35E−04	8.19E−05	44,467.26	99
	OBDE	1.80E−03	1.94E−02	168,312.5	19
	jDE	1.45E−03	9.80E−04	43,070	99
f_{26}	SBDE	1.01E−06	8.92E−06	13,160	100
	DE	7.52E−07	8.95E−06	15,643	100
	SFLSDE	9.46E−07	8.93E−06	18,070.59	100
	OBDE	1.40E−06	8.02E−06	14,615.5	100
	jDE	1.40E−06	8.05E−06	17,456	100

Table 2 (*Continued*)

Test problem	Algorithm	SD	ME	AFE	SR
f_{27}	SBDE	5.71E−06	5.00E−06	21,148.43	100
	DE	1.45E−05	1.54E−05	93,734	54
	SFLSDE	1.44E−05	1.62E−05	99,126.96	51
	OBDE	1.03E−04	1.48E−05	98,745	51
	jDE	1.03E−04	1.41E−05	113,105.5	44
f_{28}	SBDE	2.82E−14	4.16E−14	7969.85	100
	DE	2.68E−14	4.86E−14	11,910	100
	SFLSDE	2.72E−14	4.71E−14	8442.01	100
	OBDE	1.00E+00	2.97E−14	4557.5	100
	jDE	1.00E+00	3.01E−14	7744.5	100
f_{29}	SBDE	5.28E−03	4.90E−01	2915.5	100
	DE	5.26E−03	4.90E−01	3553.5	100
	SFLSDE	0.005272695	0.491507428	3234	100
	OBDE	2.48E+04	5.03E−03	2091.5	100
	jDE	2.48E+04	5.32E−03	3025	100
f_{30}	SBDE	9.14E−17	8.89E−16	42,209.5	100
	DE	7.04E−17	9.20E−16	59,972	100
	SFLSDE	8.66E−17	9.08E−16	64,399.61	100
	OBDE	9.01E−16	8.09E−16	53,106.5	100
	jDE	9.22E−16	6.71E−16	62,221	100

SBDE – *SFLSDE*, *SBDE* – *OBDE* and *SBDE* – *jDE* in terms of AR. It is clear from Table 4 that, for most of the test problems, convergence speed of *SBDE* is faster among all the considered algorithms.

Further, diversity of the considered algorithms are compared based on a diversity measure, *Average Distance around Swarm Center*. This measure is given in [18] and defined in Eq. (7)

$$D_A = \frac{1}{NP}\sum_{i=1}^{NP}\left(\sqrt{\sum_{k=1}^{D}(x_{ik}-\bar{x}_k)^2}\right) \tag{7}$$

Table 3
Results of the Student's *t* test.

Test problems	SBDE vs DE	SBDE vs SFLSDE	SBDE vs OBDE	SBDE vs jDE
f_1	+	+	+	+
f_2	+	+	+	+
f_3	=	=	=	=
f_4	+	+	+	+
f_5	+	+	+	+
f_6	+	+	+	+
f_7	+	+	+	+
f_8	+	+	+	+
f_9	+	+	+	+
f_{10}	+	+	+	+
f_{11}	+	+	+	+
f_{12}	+	+	+	+
f_{13}	+	+	+	+
f_{14}	+	+	+	+
f_{15}	+	+	+	+
f_{16}	+	+	+	+
f_{17}	+	+	+	+
f_{18}	+	+	+	+
f_{19}	+	+	+	+
f_{20}	+	+	–	+
f_{21}	+	–	+	–
f_{22}	+	+	+	+
f_{23}	=	=	=	=
f_{24}	=	=	=	=
f_{25}	+	+	+	+
f_{26}	+	+	+	+
f_{27}	+	+	+	+
f_{28}	+	+	–	–
f_{29}	+	+	–	+
f_{30}	+	+	+	+
Total number of +sign	27	26	24	25

Table 4
Acceleration Rate (AR) of *SBDE* compare to the basic *DE*, *SFLSDE*, *OBDE* and *jDE*.

Test problems	DE	SFLSDE	OBDE	jDE
f_1	1.452446731	1.56401194	1.253882062	1.50979645
f_2	1.463018292	1.693037422	2.720783268	1.760570041
f_3	0.9999998	0.997952012	0.999779855	0.9999998
f_4	2.051338949	1.584006098	1.728590169	1.734742647
f_5	1.4527852	1.557073088	1.314150911	1.505140867
f_6	2.810974663	2.396755876	1.014609358	2.576730891
f_7	1.534303185	1.615307082	1.432401559	1.558191103
f_8	1.447553612	1.551389548	1.229023527	1.511097231
f_9	1.424089936	1.535265525	1.260028551	1.481531049
f_{10}	1.548008804	1.76855836	1.50787537	1.605990783
f_{11}	1.568126352	1.702211552	1.773666144	1.64503264
f_{12}	1.437132142	1.541836897	1.243669413	1.497248761
f_{13}	1.896509964	1.978945282	1.154684576	1.86083893
f_{14}	1.521890721	1.622882901	3.275887832	1.562384818
f_{15}	2.543044801	3.993564655	4.153297941	2.172617179
f_{16}	1.429164069	1.538571889	1.24568493	1.486662511
f_{17}	1.619031595	1.650982987	1.347429426	1.582464012
f_{18}	1.459354611	1.574053959	1.428425322	1.506718392
f_{19}	1.361591073	1.550528129	1.356885735	1.442558881
f_{20}	5.102460443	3.290465194	0.776452166	3.208523188
f_{21}	1.029897419	0.442621493	1.054507565	0.581234068
f_{22}	1.161351472	1.360342328	1.097290366	1.313280553
f_{23}	0.996877563	0.995479194	0.996648339	0.996877563
f_{24}	0.997306599	0.995920739	0.997084754	0.997306599
f_{25}	1.40585348	2.291753508	8.674489104	2.219741527
f_{26}	1.188677812	1.373145137	1.110600304	1.326443769
f_{27}	4.432196622	4.68720184	4.669140924	5.348174782
f_{28}	1.494381952	1.059243273	0.571842632	0.971724687
f_{29}	1.218830389	1.109243697	0.717372663	1.03755788
f_{30}	1.420817588	1.525713643	1.258164631	1.474099433

where *NP* is the population size, *D* is the dimensionality of the problem, x_{ik} is the *k*th dimension of the *i*th individual position and $\bar{x}_k$ is the *k*th dimension of the swarm center (best individual) at any time. A low value of this measure shows swarm convergence around the swarm center while a high value shows large dispersion of individuals from the swarm center. A diversity comparison of the considered algorithms on the basis of this diversity measure is shown in Fig. 2. The diversity is calculated for the median run of each algorithm for first six test functions (f_1, f_5, f_7, f_8, f_9 and f_{11}) on which all the considered algorithms achieved 100% success. It can be observed that *SBDE* performs better than the basic *DE*, *SFLSDE*, *OBDE* and *jDE*.

For the purpose of comparison in terms of consolidated performance, boxplot analyses have been carried out for all the considered algorithms. The empirical distribution of data is efficiently represented graphically by the boxplot analysis tool [38]. The boxplots for *SBDE*, *DE*, *SFLSDE*, *OBDE* and *jDE* are shown in Fig. 3. It is clear from this figure that *SBDE* is better than the considered algorithms as interquartile range and median are comparatively low.

Further, to compare the considered algorithms, by giving weighted importance to the success rate, the mean error and the average number of function evaluations, performance indices (*PI*) are calculated [12]. The values of *PI* for the *SBDE*, *DE*, *SFLSDE*, *OBDE* and *jDE* are calculated by using following equations:

$$PI = \frac{1}{N_p}\sum_{i=1}^{N_p}(k_1\alpha_1^i + k_2\alpha_2^i + k_3\alpha_3^i)$$

where

$$\alpha_1^i = \frac{Sr^i}{Tr^i}; \quad \alpha_2^i = \begin{cases} \dfrac{Mf^i}{Af^i}, & \text{if } Sr^i > 0. \\ 0, & \text{if } Sr^i = 0. \end{cases} \text{; and}$$

$$\alpha_3^i = \frac{Mo^i}{Ao^i}, \quad i = 1, 2, \ldots, N_p$$

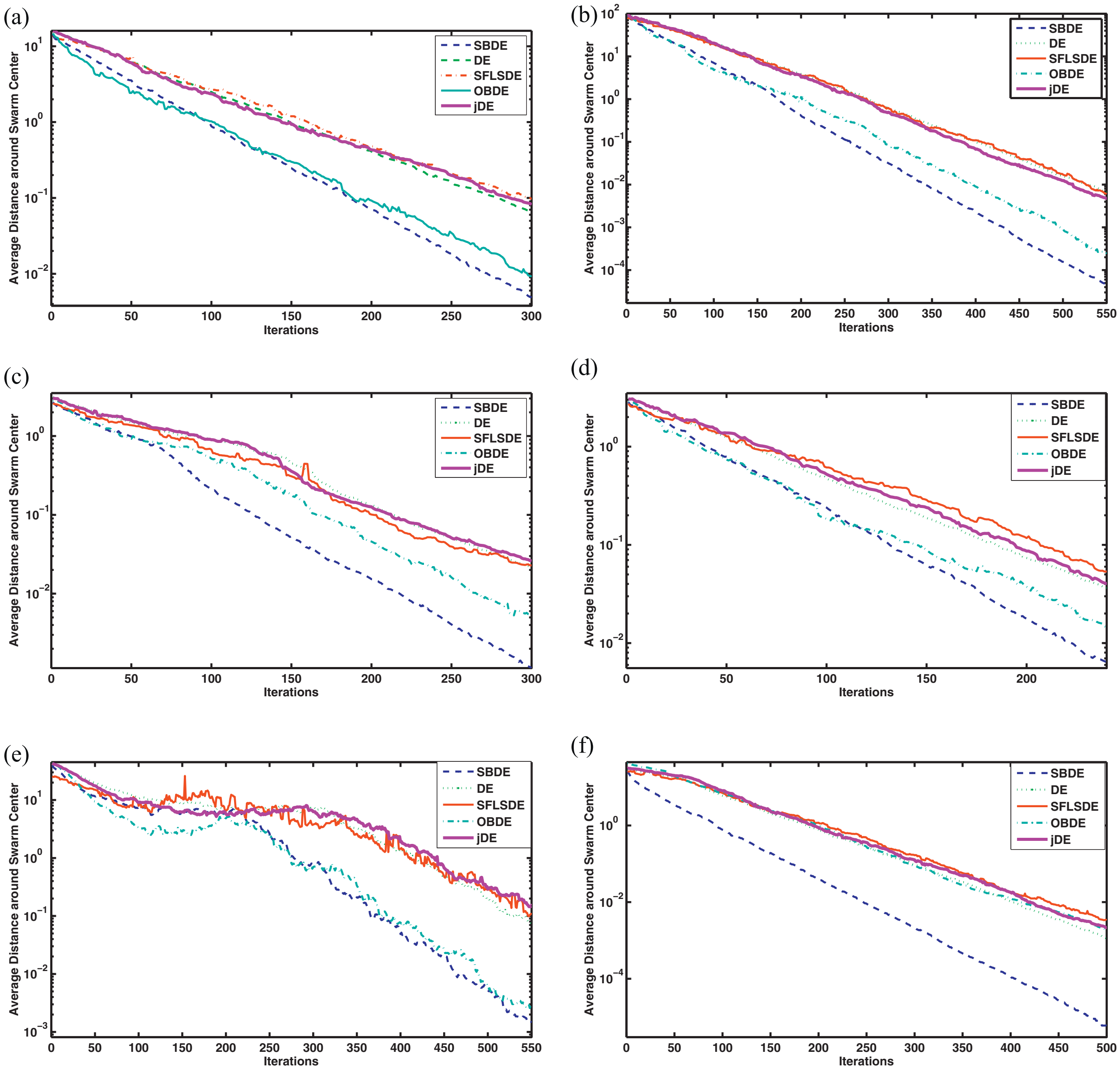

Fig. 2. Average Distance around Swarm Center; (a) for f_1, (b) for f_5, (c) for f_7, (d) for f_8, (e) for f_9, (f) for f_{11}.

- Sr^i = Successful simulations/runs of *i*th problem.
- Tr^i = Total simulations of *i*th problem.
- Mf^i = Minimum of average number of function evaluations used for obtaining the required solution of *i*th problem.
- Af^i = Average number of function evaluations used for obtaining the required solution of *i*th problem.
- Mo^i = Minimum of mean error obtained for the *i*th problem.
- Ao^i = Mean error obtained by an algorithm for the *i*th problem.
- N_p = Total number of optimization problems evaluated.

The weights assigned to the success rate, the average number of function evaluations and the mean error are represented by k_1, k_2 and k_3 respectively where $k_1 + k_2 + k_3 = 1$ and $0 \le k_1, k_2, k_3 \le 1$. To calculate the *PIs*, equal weights are assigned to two variables while weight of the remaining variable vary from 0 to 1 as given in [12]. Following are the resultant cases:

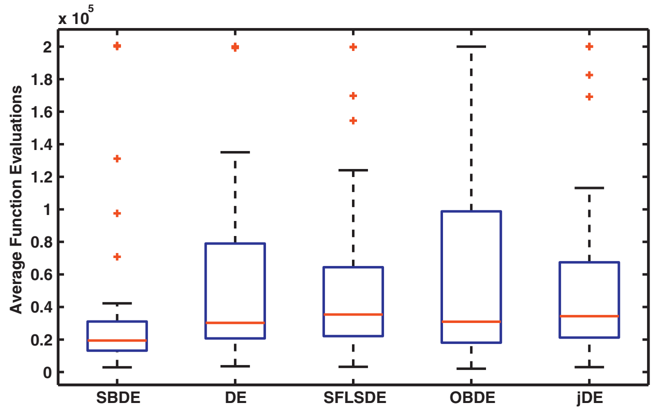

Fig. 3. Boxplot graph for Average Function Evaluation: (1) *SBDE*, (2) *DE*, (3) *SFLSDE*, (4) *OBDE*, (5) *jDE*.

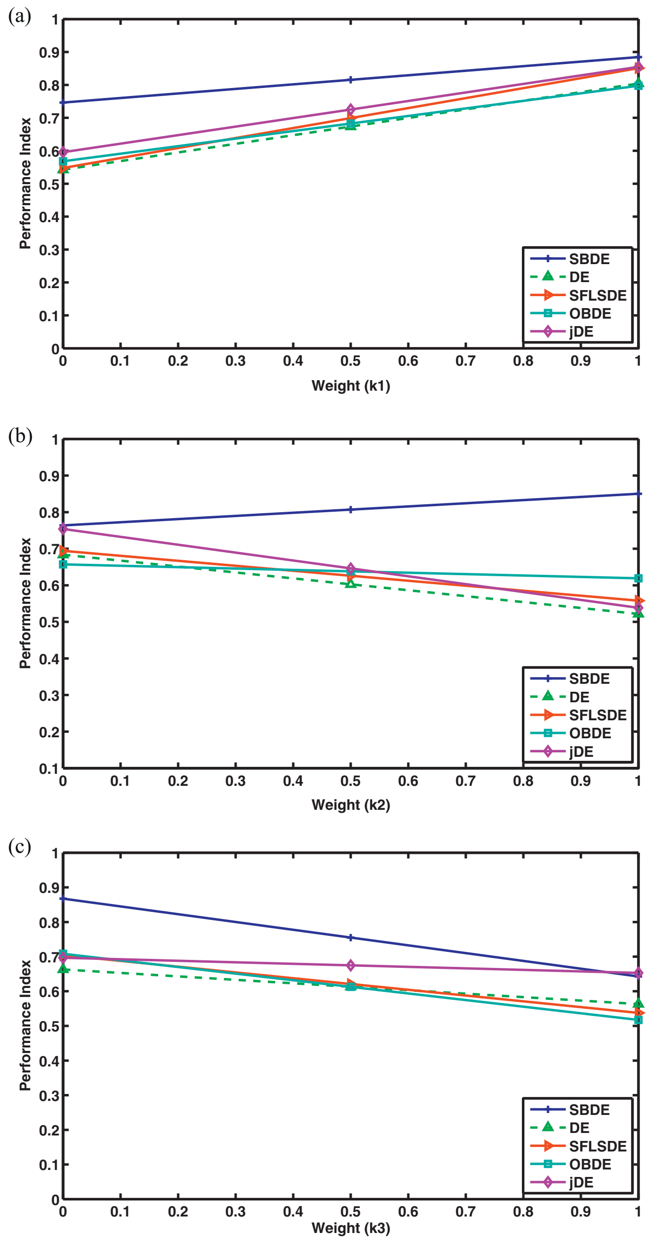

Fig. 4. Performance index for test problems; (a) for case (1), (b) for case (2) and (c) for case (3).

1. $k_1 = W, k_2 = k_3 = \frac{1-W}{2}, \quad 0 \le W \le 1;$
2. $k_2 = W, k_1 = k_3 = \frac{1-W}{2}, \quad 0 \le W \le 1;$
3. $k_3 = W, k_1 = k_2 = \frac{1-W}{2}, \quad 0 \le W \le 1.$

The graphs corresponding to each of the cases (1), (2) and (3) for *SBDE*, *DE*, *SFLSDE*, *OBDE* and *jDE* are shown in Fig. 4(a), (b), and (c) respectively. In these figures the weights k_1, k_2 and k_3 are represented by horizontal axis while the *PI* is represented by the vertical axis.

In case (1), average number of function evaluations and the mean error are given equal weights. *PIs* of the considered algorithms are superimposed in Fig. 4(a) for comparison of the performance. It is observed that *PI* of *SBDE* are higher than the considered algorithms. In case (2), equal weights are assigned to the success rate and the mean error and in case (3), equal weights are assigned to the success rate and the average function evaluations. It is clear from Fig. 4(b) and (c) that, the algorithms perform same as in case (1).

Table 5
Results of Spread Spectrum Radar Polly phase Code Design problem. The best result is highlighted in boldface.

Dimension	DE/rand/1/bin		SBDE		*t* test
	MOFV	SD	MOFV	*SD*	
10	0.53	0.07	0.53	0.07	–
15	1.08	6.66E−16	**0.69**	5.12E−3	+
20	1.66	0.09	**0.93**	0.11	+

6. Application of *SBDE* to Spread Spectrum Radar Polly phase Code Design

The problems is selected from CEC 2011 [11] based on the level of difficulty that they present to the *SBDE*. The problem is modeled as a min-max nonlinear non-convex optimization problem in continuous variables and with numerous local optima. It can be expressed as follows:

$$\text{global } \min_{x \in X} f(x) = \max\{\phi_1(x), \ldots, \phi_{2m}(x)\},$$

$$X = \{(x_1, \ldots, x_n) \in R^n | 0 \le x_j \le 2\pi, j = 1, \ldots, n\},$$

where $m = 2n - 1$ and

$$\phi_{2i-1}(x) = \sum_{j=i}^{n} \cos\left(\sum_{k=|2i-j-1|+1}^{j} x_k\right), \quad i = 1, \ldots, n,$$

$$\phi_{2i}(x) = 0.5 + \sum_{j=i+1}^{n} \cos\left(\sum_{k=|2i-j|+1}^{j} x_k\right), \quad i = 1, \ldots, n-1,$$

$$\phi_{m+i}(x) = -\phi_i(x), \quad i = 1, \ldots, m.$$

Here the objective is to minimize the module of the biggest among the samples of the so called auto-correlation function which is related to the complex envelope of the compressed radar pulse at the optimal receiver output, while the variables represent symmetrized phase differences. According to [23] the above problem has no polynomial time solution. The considered problem is solved by *DE* and *SBDE*. Table 5 shows the mean and the standard deviation of the best-of-run values for 100 independent runs of the *DE* and *SBDE* over the three difficult instances of the radar polyphase code design problem. Each algorithm used 2×10^5 function evaluations in each run. The 6th column in Table 5 indicates the statistical significance level obtained from a paired Student's *t* test between *DE* and *SBDE*. This can be observed that the proposed strategy *SBDE* improves the performance of *DE* from mean objective function value point of view. Therefore, this study recommend to implant the proposed strategy not only with the basic *DE* but with advanced variants of *DE* also.

7. Conclusion

In this paper, SBDE is proposed, analyzed and validated with the help of test problems and an engineering optimization problem. With the introduction of CLF and Dynamic scaling factor, SBDE has improved the performance as compare to *DE*, *SFLSDE*, *OBDE* and *jDE*. Through intensive statistical analysis, improvement is shown in terms of reliability, efficiency and accuracy. Overall, authors recommend SBDE as a better candidate in the field of nature inspired algorithms for function optimization due to its ability to explore and exploit in a better way.

The future scope of this work is the implementation of the proposed strategy to other nature inspired algorithms.

Acknowledgements

Second author acknowledges ABV-Indian Institute of Information Technology and Management Gwalior for providing research grant to carry out this work.

References

[1] H.A. Abbass, The self-adaptive pareto differential evolution algorithm, in: IEEE Proceedings of the 2002 Congress on Evolutionary Computation, 2002, CEC'02, vol. 1, 2002, pp. 831–836.
[2] B. Akay, D. Karaboga, A modified artificial bee colony algorithm for real-parameter optimization, Information Sciences 192 (2012) 120–142.
[3] M.M. Ali, C. Khompatraporn, Z.B. Zabinsky, A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems, Journal of Global Optimization 31 (4) (2005) 635–672.
[4] J. Brest, S. Greiner, B. Boskovic, M. Mernik, V. Zumer, Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems, IEEE Transactions on Evolutionary Computation 10 (6) (2006) 646–657.
[5] J. Brest, A. Zamuda, B. Boskovic, M.S. Maucec, V. Zumer, Dynamic optimization using self-adaptive differential evolution, in: IEEE Congress on Evolutionary Computation, 2009, CEC'09, 2009, pp. 415–422.
[6] U.K. Chakraborty, Advances Differential Evolution, vol. 143, Springer-Verlag, Heidelberg, Germany, 2008.
[7] C. Croarkin, P. Tobias, NIST/SEMATECH e-handbook of Statistical Methods, 2010.
[8] S. Das, A. Abraham, U.K. Chakraborty, A. Konar, Differential evolution using a neighborhood-based mutation operator, IEEE Transactions on Evolutionary Computation 13 (3) (2009) 526–553.
[9] S. Das, A. Konar, Two-dimensional IIR filter design with modern search heuristics: a comparative study, International Journal of Computational Intelligence and Applications 6 (3) (2006) 329–355.
[10] S. Das, P.N. Suganthan, Differential evolution: a survey of the state-of-the-art., IEEE Transactions on Evolutionary Computation 99 (2010) 1–28.
[11] S. Das, P.N. Suganthan, Problem definitions and evaluation criteria for CEC 2011 competition on testing evolutionary algorithms on real world optimization problems, Tech. Report, Jadavpur University, Kolkata, India, and Nangyang Technological University, Singapore, 2010.
[12] M. Thakur, K. Deep, A new crossover operator for real coded genetic algorithms, Applied Mathematics and Computation 188 (1) (2007) 895–911.
[13] A.P. Engelbrecht (Ed.), Computational Intelligence: An Introduction, John Wiley and Sons, England, 2002.
[14] R. Gamperle, S.D. Muller, A. Koumoutsakos, A parameter study for differential evolution, Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation 10 (2002) 293–298.
[15] J.H. Holland, Adaptation in Natural and Artificial systems, vol. 53, University of Michigan Press, Ann Arbor, 1975.
[16] D. Karaboga, B. Akay, A modified artificial bee colony (abc) algorithm for constrained optimization problems, Applied Soft Computing 11 (2011) 3021–3031.
[17] J. Kennedy, R. Eberhart, Particle swarm optimization, in: IEEE International Conference on Neural Networks, 1995. Proceedings, volume 4, 1995, pp. 1942–1948.
[18] T. Krink, J.S. VesterstrOm, J. Riget, Particle swarm optimisation with spatial particle extension, in: IEEE Proceedings of the 2002 Congress on Evolutionary Computation, 2002, CEC'02, vol. 2, 2002, pp. 1474–1479.
[19] J. Lampinen, I. Zelinka, On stagnation of the differential evolution algorithm, in: Proceedings of MENDEL, 2000, pp. 76–83.
[20] J. Liu, J. Lampinen, A fuzzy adaptive differential evolution algorithm, Soft Computing – A Fusion of Foundations, Methodologies and Applications 9 (6) (2005) 448–462.
[21] P.K. Liu, F.S. Wang, Inverse problems of biological systems using multi-objective optimization, Journal of the Chinese Institute of Chemical Engineers 39 (5) (2008) 399–406.
[22] E. Mezura-Montes, J. Velázquez-Reyes, C.A. Coello Coello, A comparative study of differential evolution variants for global optimization, in: Proceedings of the 8th Annual Conference on Genetic and evolutionary computation, ACM, 2006, pp. 485–492.
[23] N. Mladenovic, J. Petrovic, V. Kovacevic-Vujcic, M. Cangalovic, Solving spread spectrum radar polyphase code design problem by tabu search and variable neighbourhood search, European Journal of Operational Research 151 (2) (2003) 389–399.
[24] F. Neri, V. Tirronen, Scale factor local search in differential evolution, Memetic Computing 1 (2) (2009) 153–171.
[25] N. Noman, H. Iba, Enhancing differential evolution performance with local search for high dimensional function optimization, in: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, ACM, 2005, pp. 967–974.
[26] M. Omran, A. Salman, A. Engelbrecht, Self-adaptive differential evolution, Computational Intelligence and Security (2005) 192–199.
[27] M.G.H. Omran, A.P. Engelbrecht, A. Salman, Differential evolution methods for unsupervised image classification, in: IEEE Congress on Evolutionary Computation, 2005, vol. 2, 2005, pp. 966–973.
[28] K.V. Price, Differential evolution: a fast and simple numerical optimizer, in: Fuzzy Information Processing Society, 1996. NAFIPS. 1996 Biennial Conference of the North American, IEEE, 1996, pp. 524–527.
[29] K.V. Price, R.M. Storn, J.A. Lampinen, Differential Evolution: A Practical Approach to Global Optimization, Springer, Berlin, 2005.
[30] A.K. Qin, V.L. Huang, P.N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, IEEE Transactions on Evolutionary Computation 13 (2) (2009) 398–417.
[31] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, Opposition-based differential evolution, IEEE Transactions on Evolutionary Computation 12 (1) (2008) 64–79.
[32] T. Rogalsky, S. Kocabiyik, R.W. Derksen, Differential evolution in aerodynamic optimization, Canadian Aeronautics and Space Journal 46 (4) (2000) 183–190.
[33] R. Storn, K. Price, Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces, Journal of Global Optimization 11 (1997) 341–359.
[34] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.P. Chen, A. Auger, S. Tiwari, Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization, KanGAL Report, 2005, May, pp. 341–357.
[35] J. Teo, Exploring dynamic self-adaptive populations in differential evolution, Soft Computing – A Fusion of Foundations, Methodologies and Applications 10 (8) (2006) 673–686.
[36] J. Vesterstrom, R. Thomsen, A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems, in: IEEE Congress on Evolutionary Computation, 2004, CEC2004, vol. 2, 2004, pp. 1980–1987.
[37] M. Weber, V. Tirronen, F. Neri, Scale factor inheritance mechanism in distributed differential evolution, Soft Computing – A Fusion of Foundations, Methodologies and Applications 14 (11) (2010) 1187–1207.
[38] D.F. Williamson, R.A. Parker, J.S. Kendrick, The box plot: a simple visual method to interpret data, Annals of internal medicine 110 (11) (1989) 916.
[39] J.Y. Yan, Q. Ling, D.M. Sun, A differential evolution with simulated annealing updating method, in: IEEE International Conference on Machine Learning and Cybernetics, 2006, pp. 2103–2106.
[40] D. Zaharie, Control of population diversity and adaptation in differential evolution algorithms, in: Proceedings of MENDEL, 2003, pp. 41–46.
[41] D. Zaharie, D. Petcu, Adaptive pareto differential evolution and its parallelization, Parallel Processing and Applied Mathematics (2004) 261–268.
[42] J. Zhang, A.C. Sanderson, Jade: adaptive differential evolution with optional external archive, IEEE Transactions on Evolutionary Computation 13 (5) (2009) 945–958.

Harish Sharma received his B.Tech, M.Tech degree in Computer Engineering from Government Engineering College, Kota and Rajasthan Technical University, Rajasthan in 2003 and 2009 respectively. He is currently a Research Scholar at ABV-Indian Institute of Information Technology and Management, Gwalior, India.

Dr. Jagdish Chand Bansal is an Assistant Professor at South Asian University Delhi, India. He has worked as an Assistant Professor at ABV-Indian Institute of Information Technology and Management Gwalior. He has obtained his Ph.D. in Mathematics from IIT Roorkee. He is the editor in chief of "International Journal of Swarm Intelligence (IJSI)" published by Inderscience. His primary area of interest is Nature Inspired Optimization Techniques.

Dr. Karm Veer Arya is working as an Associate Professor at ABV-Indian Institute of Information Technology & Management, Gwalior, India. He earned Ph.D. degree in Computer Science & Engineering from Indian Institute of Technology (I.I.T. Kanpur), Kanpur, India. He has more than 20 years of experience to teach the undergraduate and postgraduate classes. He has published more than 75 journal and conference papers in the area of Information Security, image processing, biometrics, wireless ad hoc networks and soft computing.