Applied Mathematics and Computation xxx (2012) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Applied Mathematics and Computation

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

Unconventional initialization methods for differential evolution

Musrrat Ali [a,*], Millie Pant [b], Ajith Abraham [c]

[a] Department of Computer Science, Sungkyunkwan University, Suwon 440467, Republic of Korea
[b] Department of Paper Technology, Indian Institute of Technology Roorkee, Saharanpur Campus, Saharanpur 247001, India
[c] Machine Intelligence Research Labs (MIR Labs), Scientific Network for Innovation and Research Excellence, P.O. Box 2259, Auburn, WA 98071-2259, USA

A R T I C L E   I N F O

Keywords:
Stochastic optimization
Differential evolution
Initial population
Random numbers
Quadratic interpolation
Nonlinear simplex method

A B S T R A C T

The crucial role played by the initial population in a population-based heuristic optimization cannot be neglected. It not only affects the search for several iterations but often also has an influence on the final solution. If the initial population itself has some knowledge about the potential regions of the search domain then it is quite likely to accelerate the rate of convergence of the optimization algorithm. In the present study we propose two schemes for generating the initial population of differential evolution (DE) algorithm. These schemes are based on quadratic interpolation (QI) and nonlinear simplex method (NSM) in conjugation with computer generated random numbers. The idea is to construct a population that is biased towards the optimum solution right from the very beginning of the algorithm. The corresponding algorithms named as QIDE (using quadratic interpolation) and NSDE (using non linear simplex method), are tested on a set of 20 traditional benchmark problems with box constraints and 7 shifted (non-traditional) functions taken from literature. Comparison of numerical results with traditional DE and opposition based DE (ODE) show that the proposed schemes considered by us for generating the random numbers significantly improves the performance of DE in terms of convergence rate and average CPU time.

1. Introduction

DE is comparatively a recent addition to class of population based search heuristics. Nevertheless, it has emerged as one of the techniques most favored by engineers for solving continuous optimization problems. DE [1] has several attractive features. Besides being an exceptionally simple evolutionary strategy, it is significantly faster and robust for solving numerical optimization problems and is more likely to find the function's true global optimum. Also, it is worth mentioning that DE has a compact structure with a small computer code and has fewer control parameters in comparison to other evolutionary algorithms. Originally Storn and Price proposed a single strategy for DE, which they later extended to ten different strategies [2].

DE has been successfully applied to a wide range of problems including optimization of process synthesis and design problems [3], application of DE in image processing [4–6], traveling salesman problem [7], multi class support vector machine [8], optimization of directional over-current relay settings [9], multi-objective optimization [10] etc.

Despite having several striking features and successful applications to various fields DE is sometimes criticized for its slow convergence rate for computationally expensive functions. By varying the control parameters the convergence rate of DE may be increased but it should be noted that it does not affect the quality of solution. Generally, in population based search techniques like DE an acceptable trade-off should be maintained between convergence and type of solution, which

* Corresponding author.
E-mail addresses: musrrat.iitr@gmail.com, ali82dpt@iitr.ernet.in (M. Ali).

0096-3003/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.amc.2012.10.053

even if not a global optimal solution should be satisfactory rather than converging to a suboptimal solution which may not even be a local solution. Several attempts have been made in this direction to fortify DE with suitable mechanisms to improve its performance. Most of the studies involve the tuning or controlling of the parameters of algorithm and improving the mutation, crossover and selection mechanism, some interesting modifications that helped in enhancing the performance of DE include introduction of greedy random strategy for selection of mutant vector [11], modifications in mutation and localization in acceptance rule [12], DE with preferential crossover [13], crossover based local search method for DE [14], self adaptive differential evolution algorithm [15], new donor schemes proposed for the mutation operation of DE [16], DE with Cauchy mutation [17]. There are also work have done on parameter analysis [18] and hybridization [19,20]. All the modified versions have shown that a slight change in the structure of DE can help in improving its performance. However, the role of the initial population, which is the topic of this paper, is widely ignored. The opening sentence of these algorithms is usually "generate an initial population" without indicating how this should be done. There is only few literature is available on this topic [21–24]. An interesting method for generating the initial population was suggested by Rahnamayan et al. [25,26] in which the initial population was generated using opposition based rule.

To further continue the research in this direction, in this paper we propose two schemes for generating the initial population of basic DE algorithm. These schemes are based on quadratic interpolation (QI) method and nonlinear simplex method in conjugation with random numbers. The corresponding modified DE versions are named (1) Quadratic interpolation method called QIDE and (2) non linear simplex method called NSDE. Both QI and NSM are well known local search methods. Their use provides additional information about the potential regions of the search domain.

In the present study our aim is to investigate the effect of initial population on payoff between convergence rate and solution quality. Our motivation is to encourage discussions on methods of initial population construction. Performances of the proposed algorithms are compared with Basic DE and differential evolution initialized by opposition based learning (ODE), which is a recently modified version of differential evolution [25], on a set of twenty unconstrained benchmark problems and 7 shifted functions.

Remaining of the paper is organized in following manner; in Section 2, we give a brief description of DE. In Section 3, we have given a brief description of the initialization schemes used in this paper. The proposed algorithms are explained in Section 4. Section 5 deals with experimental settings and parameter selection. Benchmark problems considered in the present study and the results are given in Section 6. The conclusions based on the present study are finally drawn in Section 7.

2. Differential evolution (DE)

DE starts with a population of NP candidate solutions which may be represented as $X_{i,G}$, $i = 1,\ldots,NP$, where i index denotes the population and G denotes the generation to which the population belongs. The working of DE depends on the manipulation and efficiency of three main operators; mutation, crossover and selection which briefly described in this section.

2.1. Mutation

Mutation operator is the prime operator of DE and it is the implementation of this operation that makes DE different from other Evolutionary algorithms. The mutation operation of DE applies the vector differentials between the existing population members for determining both the degree and direction of perturbation applied to the individual subject of the mutation operation. The mutation process at each generation begins by randomly selecting three individuals in the population. The most often used mutation strategies implemented in the DE codes are listed below.

$$\text{DE/rand/1}: \quad V_{i,G+1} = X_{r_1,G} + F \times (X_{r_2,G} - X_{r_3,G}), \tag{1a}$$
$$\text{DE/rand/2}: \quad V_{i,G+1} = X_{r_1,G} + F \times (X_{r_2,G} - X_{r_3,G}) + F \times (X_{r_4,G} - X_{r_5,G}), \tag{1b}$$
$$\text{DE/best/1}: \quad V_{i,G+1} = X_{best,G} + F \times (X_{r_1,G} - X_{r_2,G}), \tag{1c}$$
$$\text{DE/best/2}: \quad V_{i,G+1} = X_{best,G} + F \times (X_{r_1,G} - X_{r_2,G}) + F \times (X_{r_3,G} - X_{r_4,G}), \tag{1d}$$
$$\text{DE/rand-to-best/1}: \quad V_{i,G+1} = X_{r_1,G} + F \times (X_{best,G} - X_{r_2,G}) + F \times (X_{r_3,G} - X_{r_4,G}), \tag{1e}$$

where, $i = 1,\ldots,NP$, $r_1, r_2, r_3 \in \{1,\ldots,NP\}$ are randomly selected and satisfy:
$r_1 \neq r_2 \neq r_3 \neq i$, $F \in [0,1]$, F is the control parameter proposed by Storn and Price.

Throughout the paper we shall refer to the strategy (1a) which is apparently the most commonly used version and shall refer to it as basic version.

2.2. Crossover

Once the mutation phase is complete, the crossover process is activated. The perturbed individual, $V_{i,G+1} = (v_{1,i,G+1},\ldots,v_{n,i,G+1})$, and the current population member, $X_{i,G} = (x_{1,i,G},\ldots,x_{n,i,G})$, are subject to the crossover operation, that finally generates the population of candidates, or "trial" vectors, $U_{i,G+1} = (u_{1,i,G+1},\ldots,u_{n,i,G+1})$, as follows:

$$u_{j,i.G+1} = \begin{cases} v_{j,i.G+1} & \text{if } rand_j \leqslant C_r \vee j = k, \\ x_{j,i.G} & \text{otherwise}, \end{cases} \tag{2}$$

where, j = 1 . . . ,n, $k \in \{1, \ldots, n\}$ is a random parameter's index, chosen once for each *i*, and the crossover rate, $Cr \in [0,1]$, the other control parameter of DE, is set by the user.

2.3. Selection

The selection scheme of DE also differs from that of other EAs. The population for the next generation is selected from the individual in current population and its corresponding trial vector according to the following rule:

$$X_{i.G+1} = \begin{cases} U_{i.G+1} & \text{if } f(U_{i.G+1}) \leqslant f(X_{i.G}) \\ X_{i.G} & \text{otherwise} \end{cases} \tag{3}$$

Thus, each individual of the temporary (trial) population is compared with its counterpart in the current population. The one with the lower objective function value will survive from the tournament selection to the population of the next generation. As a result, all the individuals of the next generation are as good as or better than their counterparts in the current generation. In DE trial vector is not compared against all the individuals in the current generation, but only against one individual, its counterpart, in the current generation. A schematic of the DE algorithm is given in Fig. 1. The pseudo code of algorithm is given below.

DE pseudo code:

Step 1: The first step is the random initialization of the parent population. Randomly generate a population of (say) NP vectors, each of n dimensions: $x_{i,j} = x_{min,j} + \text{rand}(0,1)(x_{max,j} - x_{min,j})$, where $x_{min,j}$ and $x_{max,j}$ are lower and upper bounds for *j*th component respectively, rand (0, 1) is a uniform random number between 0 and 1.
Step 2: Calculate the objective function value $f(X_i)$ for all X_i.
Step 3: Select three points from population and generate perturbed individual V_i using Eq. (1a).

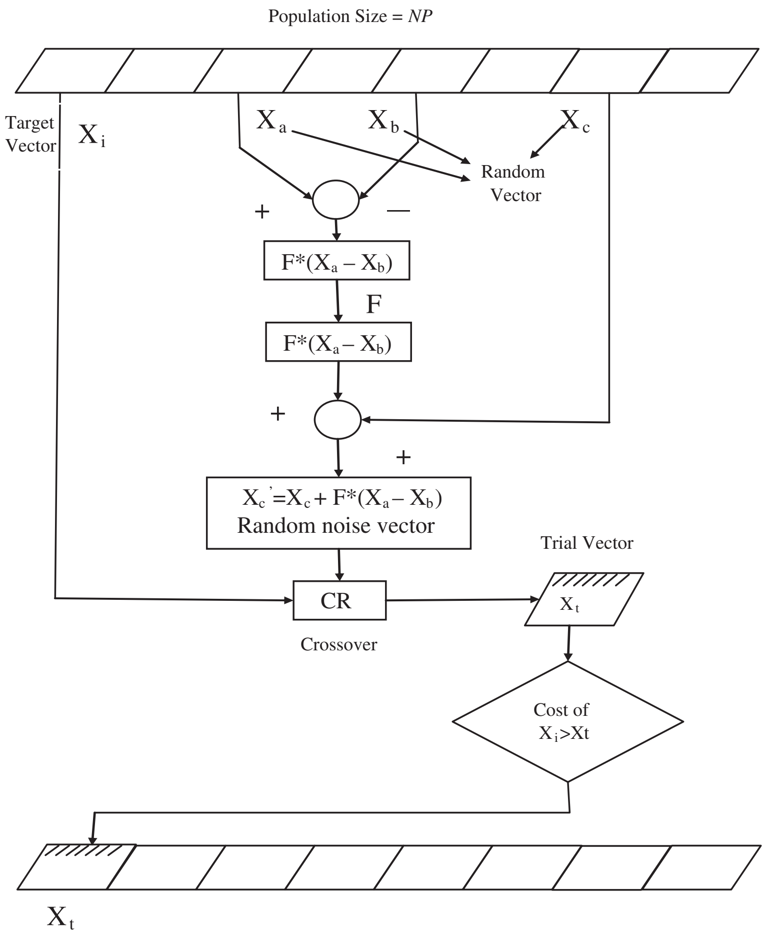

Fig. 1. A schematic of differential evolution algorithm.

Step 4: Recombine the each target vector X_i with perturbed individual generated in step 3 to generate a trial vector U_i using Eq. (2).
Step 5: Check whether each variable of the trial vector is within range. If yes, then go to step 6 else make it within range using $u_{i,j} = 2 \times x_{min,j} - u_{i,j}$, if $u_{i,j} < x_{min,j}$ and $u_{i,j} = 2 \times x_{max,j} - u_{i,j}$, if $u_{i,j} > x_{max,j}$, and go to step 6.
Step6: Calculate the objective function value for vector U_i.
Step 7: Choose better of the two (function value at target and trial point) using Eq. (3) for the next generation.
Step 8: Check whether the convergence criterion is met if yes then stop; otherwise go to step 3.

3. Initial population generation methods used in the present study

In the present study besides using the traditional method of generating the initial population i.e. besides using the computer generated pseudorandom numbers, we have considered three other methods of generating the initial population; these are quadratic interpolation (QI), nonlinear simplex method (NSM) and Opposition Based Method (OBM). A brief description of these methods is given as follows:

3.1. Opposition Based Method (OBM)

The use of OBM was suggested by Rahnamayan et al. [25,26], where they used *opposition based learning* (OBL) to generate the initial population. The basic idea behind OBL is the simultaneous consideration of an estimate and its corresponding opposite estimate in order to achieve a better approximation for the current candidate solution. Mathematically, it has been proven in [27] that the opposite numbers are more likely to be closer to the optimal solution than purely random ones. Opposite numbers may be defined as:

Opposite Number—If $x \in [l,u]$ is a real number, then the opposite number x' is defined by

$$x' = l + u - x. \tag{4}$$

Similarly, this definition can be extended to higher dimensions as follows [26].

Opposite n-Dimensional point—If $X = (x_1,x_2,\ldots,x_n)$ is a point in n-dimensional space, where $x_1,x_2,\ldots,x_n \in R$ and $x_i \in [l_i,u_i]$ $\forall i \in \{1,2,\ldots,n\}$. Then the opposite point $X' = (x'_1, x'_2, \ldots, x'_n)$ is completely defined by its components

$$x'_i = l_i + u_i - x_i \tag{5}$$

Rahmanayan et al. applied their method for generating the initial population for DE algorithm. Their algorithm called ODE (Opposition based differential evolution) reportedly gave good results on a set of benchmark problems in comparison to the traditional DE using computer generated random numbers.

3.2. Quadratic interpolation (QI)

QI method is one of the simplest and the oldest direct search method used for solving optimization problems that makes use of gradient in a numerical way. In this method we try to select the three distinct points (say a, b and c) randomly from the population. A parabolic curve is then fitted into the selected points and the point lying at the minimum of this quadratic curve is evaluated.

Mathematically, the new point r_i is produced as follows:

$$r_i = \frac{1}{2}\frac{(b^2 - c^2) \times f(a) + (c^2 - a^2) \times f(b) + (a^2 - b^2) \times f(c)}{(b - c) \times f(a) + (c - a) \times f(b) + (a - b) \times f(c)} \quad \text{where } i = 1, 2, \ldots, NP. \tag{6}$$

It is clear from Fig. 2 that the fitted function passing through these three points is parabola and new point is produced at minimum of this parabola.

QI has been used in conjugation with several variants of random search/evolutionary algorithms and has given good results. Zhang et al. [28] hybridized it with DE. Some other papers using QI approach are [29,30].

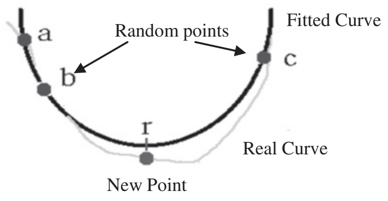

Fig. 2. Parabolic curve of quadratic interpolation.

3.3. Nonlinear simplex method (NSM)

The NSM search method was first introduced by Nelder and Mead in 1965 [31] and is perhaps one of the most widely used local direct search methods for nonlinear unconstrained optimization. NSM is a derivative-free line search method that was specially designed for solving traditional unconstrained problems of minimization type, like nonlinear least squares problem, nonlinear simultaneous equations, and function minimization. The NSM works through a sequence of four elementary geometric transformations namely; reflection, expansion, contraction and reduction. With the help of these transformations the simplex can improve itself and come closer to the optimum. To select the appropriate transformation, the method only uses the values of the function to be optimized at the vertices of the simplex considered. After each transformation, the current worst vertex is replaced by a better one. In case of a minimization problem, at the beginning of the algorithm, only that point of the simplex is moved where the objective function is worst and a point image of the worst point is generated. This operation is called reflection. If the reflected point is better than all other points, the method expands the simplex in this direction; otherwise, if it is at least better than the worst one, the algorithm performs again the reflection with the new worst point. The contraction step is performed when the worst point is at least as good as the reflected point, in such a way that the simplex adapts itself to the function landscape and finally surrounds the optimum. If the worst point is better than the contracted point, reduction is performed.

The sequence of transformations reflection, expansion, contraction and reduction are illustrated in Figs. 3a–3d respectively.

The procedure of NSM is outlined as follows:

Step 1: Select $n + 1$ point randomly from population and evaluate function at these points, and find out X_{best} and X_{worst}.
Step 2: Calculate the centroid of these points excluding the worst point, say X_{worst} at which function is maximum.

$$X_{cent} = \frac{1}{n+1}\left[\left(\sum_{i=1}^{n+1} X_{ij}\right) - X_{worst,j}\right] \quad j = 1, \ldots, n \tag{7}$$

Step 3: *Reflection*: Reflect X_{worst} through the centroid to a new point X_1. And calculate the function value at this point.

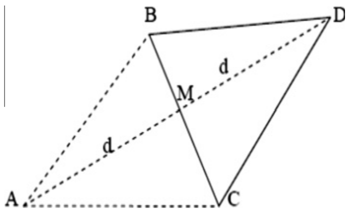

Fig. 3a. Reflection of A to D.

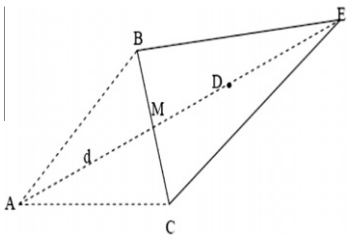

Fig. 3b. Expansion of D to E.

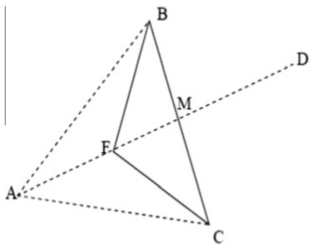

Fig. 3c. Contraction of D to F.

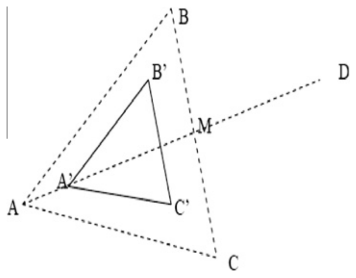

Fig. 3d. Reduction of ABC to A′B′C′.

$$X_1 = X_{cent} + \alpha(X_{cent} - X_{worst}) \quad \text{where } \alpha > 0 \text{ is reflection coefficient} \tag{8}$$

Step 4: Expansion: If f (X_1) < =f(X_{best}) then perform expansion to generate a new point X_2 in the expanded region otherwise go to step 5. If f(X_2) < f(X_{best}) then replace X_{worst} by X_2 and continue from step1. Otherwise, replace X_{worst} by X_1 and continue from step 1.

$$X_2 = X_{cent} + \gamma(X_1 - X_{cent}) \quad \text{where } \gamma > 1 \text{ is expansion coefficient.} \tag{9}$$

Step 5: Contractions: If f(X_1) < f(X_{worst}) then produce a new point X_3 by contraction otherwise go to step 6. If f(X_3) < f(X_{worst}) then replace X_{worst} by X_3 and return to step 1 to continue the search.

$$X_3 = X_{cent} + \beta(X_{worst} - X_{cent}) \quad \text{where } 0 < \beta < 1 \text{ is contraction coefficient.} \tag{10}$$

Step 6: Reduction: f(X_1) > f(X_{worst}) reduce all the vectors $(X_i - X_{best})$ by one half from X_{best} by computing.

$$X_i = X_{best} + 0.5(X_i - X_{best}) \quad i = 1, \ldots, n+1 \tag{11}$$

Return to step 1 to continue the search.

4. Proposed algorithms

The proposed algorithms, named as Nonlinear Simplex DE (NSDE) and quadratic interpolation DE (QIDE), are structured in a similar manner as that of basic DE, except in the initialization phase. Here we use NSM and QI in conjugation with computer generated random numbers to generate the initial population. We have made slight changes in the schemes described in the previous sections. In QI, instead of choosing three random points we have selected only two points randomly while the third point is chosen as the one having the best fitness value. Likewise, in case of NSM we have replaced the reduction equation by generating a random point between the lower and the upper bounds.

The initialization phase of NSDE and QIDE consists of the following four steps:

- Using computer generated random numbers construct a population set P having NP points.
- Using QI/NSM construct a population set Q having NP points.
- Merge the two sets (P and Q) to obtain 2*NP points.
- Select NP elite candidates to construct the initial population.

Both QI and NSM schemes are used in conjugation with random numbers. This is done to include an element of randomness in the population and to preserve the initial diversity of the population. A pure QI or NSM would have made the initial search greedy in nature and might have biased the algorithm towards a local region.

With the use of NSM and QI methods, the initial population is provided with the information of the good regions of the search domain. These methods are not computationally expensive, since for each particle of the initial population one function evaluation is done, which is inevitable even if we use a randomly distributed initial population.

Once the initialization is complete, the remaining of the algorithms is same as that of the basic DE, defined in Section 2. A schematic of the algorithms is given in Fig. 4.

4.1. Effects of using the proposed methods to generate initial population:

The initial generation of population applying quadratic interpolation and nonlinear simplex method makes use of the function value to determine a candidate point for the additional population. As a result in the initial step itself we get a collection of fitter individuals which may help in increasing the efficiency of the algorithm. Consequently, the chances of obtaining the optimum in fewer NFEs increases considerably or in other words the convergence rate of the algorithm becomes faster.

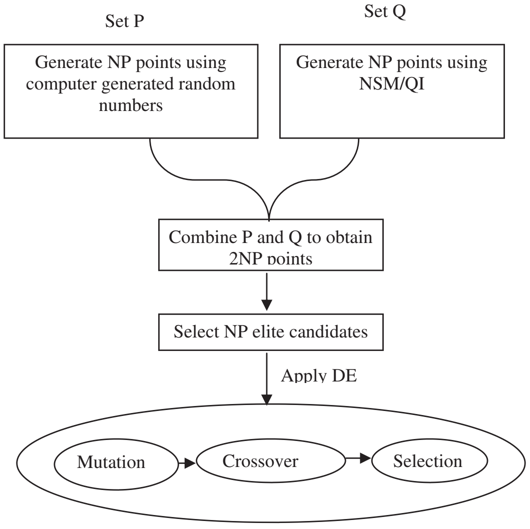
Set P
Set Q
Generate NP points using computer generated random numbers
Generate NP points using NSM/QI
Combine P and Q to obtain 2NP points
Select NP elite candidates
Apply DE
Mutation
Crossover
Selection

Fig. 4. A schematic of QIDE/NSDE algorithms.

The working of the proposed algorithms is shown with the help of two test functions; Rosenbrock and Griewank. We generated an initial population of 100 points within the range $[-2,2]$ for Rosenbrock function and within the range $[-600,600]$ for Griewank function using basic DE, ODE, QIDE and NSDE. The dispersion of points is illustrated graphically in Figs. 5a–5d and 6a–6d respectively.

From these illustrations we can observe that while using the proposed algorithms, the search space gets concentrated around the global optima which lies at $(1,1)$ with objective function value zero, for two dimensional Rosenbrock function and which lies at $(0,0)$ with objective function value 0, for Griewank function.

When the initial population is constructed using QIDE and NSDE, the large search domain, $[-600,600]$, of Griewank function is contracted to the range of around $[-400,400]$ while using NSDE whereas it is contracted further to around $[-200,200]$ when QIDE is used.

5. Experimental setup

With DE, the lower limit for population size, NP, is 4 since the mutation process requires at least three other chromosomes for each parent. While testing the algorithms, we began by using the optimized control settings of DE. Population size, NP can always be increased to help maintain population diversity. As a general rule, an effective NP is between $3 \times n$ and $5 \times n$, but can often be modified depending on the complexity of the problem. For the present study we performed several experiments with the population size as well as with the crossover rate and mutation probability rate and observed that for problems up to dimension 30, a population size of $3 \times n$ is sufficient. But here we have taken fixed population size NP = 100, which is slightly larger than $3 \times n$ and 500 for the shifted functions. Values of F, outside the range of 0.4–1.2 are rarely effective, so $F = 0.5$ is usually a good initial choice. In general higher value of C_r help in speeding up the convergence rate therefore

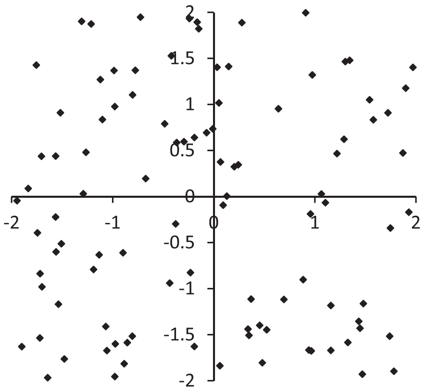

Fig. 5a. Initial population consisting of 100 points in the range [-2,2] for Rosenbrock function using basic DE.

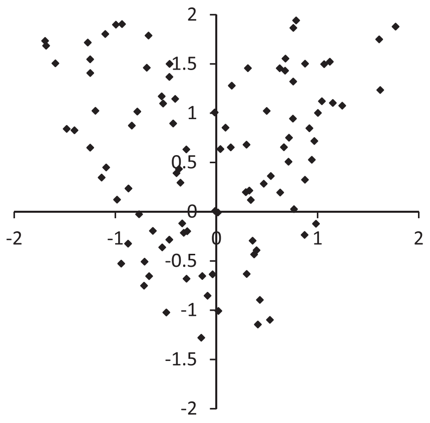

Fig. 5b. Initial population consisting of 100 points in the range [-2,2] for Rosenbrock function using ODE.

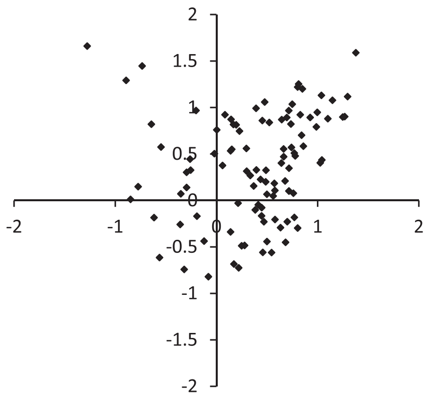

Fig. 5c. Initial population consisting of 100 points in the range [-2,2] for Rosenbrock function using QIDE.

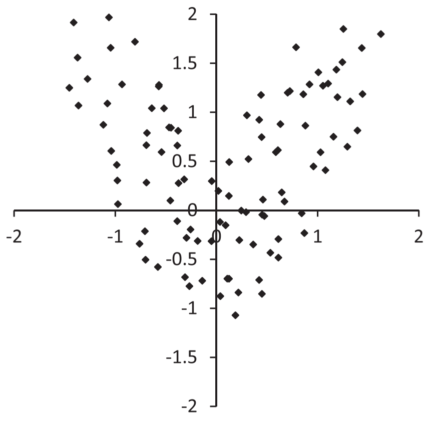

Fig. 5d. Initial population consisting of 100 points in the range [-2,2] for Rosenbrock function using NSDE.

in the present study we have taken $C_r = 0.9$. All the algorithms are executed on a PIV PC, using DEV C++, thirty times for each problem. Random numbers are generated using the inbuilt random number generator *rand* ( ) *function* available in DEVC++.

In every case, a run was terminated when the best function value obtained is less than a threshold for the given function or when the maximum number of function evaluation (NFE = 10^6) was reached 5×10^6 for shifted functions. In order to have a fair comparison, these settings are kept the same for all algorithms over all benchmark functions during the simulations.

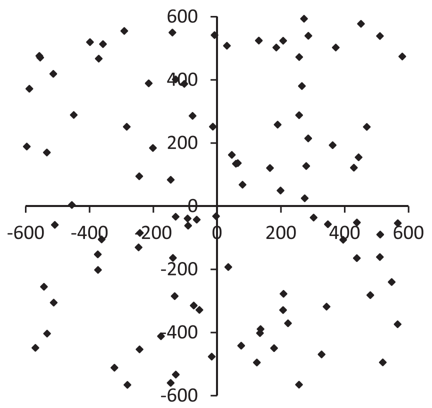

Fig. 6a. Initial population consisting of 100 points in the range [−600,600] for Griewanks function using DE.

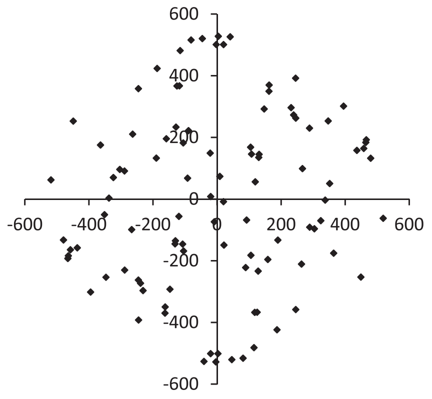

Fig. 6b. Initial population consisting of 100 points in the range [−600,600] for Griewank function using ODE.

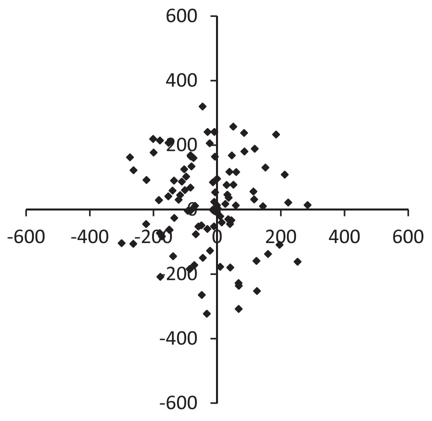

Fig. 6c. Initial population consisting of 100 points in the range [−600,600] for Griewank function using QIDE.

6. Numerical results and comparisons

6.1. Benchmark problems

The performance of proposed algorithms is evaluated on a test bed of twenty standard, benchmark problems with box constraints, taken from the literature [26]. Mathematical models of the benchmark problems along with the true optimum value are given in Appendix. Besides analyzing the performance of the proposed algorithms on traditional benchmark prob-

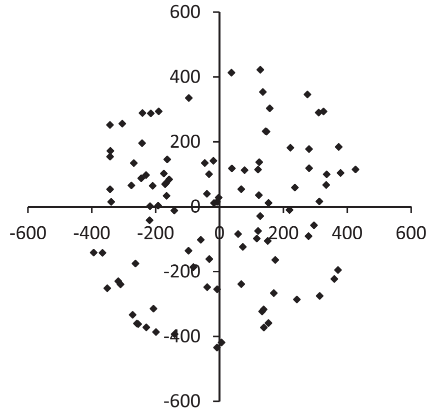

Fig. 6d. Initial population consisting of 100 points in the range [−600,600] for Griewank function using NSDE.

lems, we also validated it on a selected set of recently proposed benchmark test suite for CEC 2008 special session and competition on large scale global optimization [32]. We considered seven problems from this test suite and tested them for dimension 500. It includes the two unimodal (F1 and F2) and five multimodal (F3–F7) functions among which four are non-separable (F2, F3, F5, F7) and three separable (F1, F4, F6).

6.2. Performance measures

Average fitness function value and standard deviation (STD)

$$\text{Average NFE} = \frac{\sum_{i=1}^{n} \text{NFE}(f_i)}{n},$$

$$\text{Improvement (\%) in terms of NFE} = \frac{\text{Total NFE(basic DE algorithm)} - \text{Total NFE(Algorithm to be compared)}}{\text{Total NFE(basic DE algorithm)}} \times 100,$$

$$\text{Acceleration rate(AR)} = \frac{\text{Total NFE for basic DE}}{\text{Total NFE for algorithm to be compared}},$$

$$\text{Average CPU time} = \frac{\sum_{i=1}^{n} \text{Time}(f_i)}{n},$$

$$\text{Improvement (\%)in terms of CPU Time} = \frac{\text{Total time(basic DE algorithm)} - \text{Total time(Algorithm to be compared)}}{\text{Total NFE(basic DE algorithm)}} \times 100.$$

Besides the above mentioned performance measures, we have used student t-test to analyze the algorithms
The results corresponding to these performance measures are given in Tables 1–4, 5a–5c and 6.

6.3. Performance comparison of proposed QIDE and NSDE with basic DE and ODE

We have compared the proposed algorithms with the basic DE and ODE. Here we would like to mention that we have used ODE version given in [25] instead of [26] because in [26], the authors have used the additional features like opposition based generation jumping etc. while in the present study we just focusing on the effect of initial population generation on basic DE algorithm. From Table 1, which gives the average fitness function value, standard deviation, it can be observed that for the 20 benchmark problems taken in the present study all the algorithms gave more or less similar results, with marginal differences, which are comparable to true optimum. DE, ODE and NSDE performed better than other algorithms in 25% cases, while QIDE gave a better performance in comparison to other algorithms in 20% cases in term of average fitness function value. For the function f_{14} (Step function) all the algorithms gave same results. Further to ascertain the significance of the difference of mean of objective function values, we apply t-test at a 0.05 level of significance. A '$\sim$' sign indicates that there is no significant difference in the means and a '+' ('-') sign indicates that the mean objective function value obtained by DE (ODE, QIDE, NSDE) is significantly larger than the mean objective function value obtained ODE, QIDE, NSDE (DE). This is shown in the last columns of Table 1. The best and worst fitness function values obtained, in 30 runs, by all the algorithms for benchmark problems are given in Table 2.

In Table 3 we have recorded the total time taken by the algorithms considered in the present study. Basic DE took 32.68 s while ODE, QIDE and NSDE took 30.08, 26.78 and 27.89 s respectively to solve the 20 test problems. Thus we can say that the overall percentage improvement while applying ODE is around 8% in comparison to basic DE and there is an improvement of around 18% and 15% while applying QIDE and NSDE respectively, which is twice the improvement shown by ODE.

Table 1
Mean fitness, standard deviation of functions in 30 runs and t-test results.

Function	Dim.	Mean fitness, (Standard deviation)				t-Test results		
		DE	ODE	QIDE	NSDE	ODE	QIDE	NSDE
f_1	30	**0.0546854** (0.0131867)	0.0901626 (0.0077778)	0.0855299 (0.0133846)	0.0916686 (0.00721253)	–	–	–
f_2	30	**0.0560517** (0.0116127)	0.0918435 (0.00565233)	0.0932952 (0.00867206)	0.0866163 (0.00666531)	–	–	–
f_3	30	0.0957513 (0.00293408)	0.0952397 (0.00499586)	0.0951664 (0.00336872)	**0.0951172** (0.00405255)	~	~	~
f_4	10	0.0931511 (0.0145175)	0.0874112 (0.00699322)	**0.0778262** (0.0137472)	0.0851945 (0.0121355)	~	+	+
f_5	30	0.0915561 (0.012111)	**0.0885065** (0.00711877)	0.0915062 (0.00638926)	0.0916412 (0.00860403)	~	~	~
f_6	30	0.0942648 (0.00478545)	0.0933845 (0.00620528)	0.0930114 (0.00501949)	**0.0926704** (0.00735851)	~	~	~
f_7	2	**4.26112e−008** (2.5783e−008)	6.23824e−008 (2.75612e−008)	4.5905e−008 (3.3009e−08)	4.9999e−008 (2.95279e−08)	–	~	~
f_8	4	0.0620131 (0.0239495)	**0.0528597** (0.0276657)	0.067081 (0.0229266)	0.0591064 (0.0123711)	~	~	~
f_9	30	0.088998 (0.00880246)	0.092875 (0.00487147)	**0.0848135** (0.00898217)	0.0882776 (0.0103789)	~	+	~
f_{10}	10	−7.91444 (3.40729)	−9.61563 (0.024986)	−9.62706 (0.0198371)	−**9.62952** (0.0238362)	+	+	+
f_{11}	30	**0.0842833** (0.00897659)	0.0890837 (0.00961583)	0.0861358 (0.0102781)	0.0901177 (0.00969009)	~	~	+
f_{12}	30	0.0940407 (0.00501821)	**0.0931232** (0.00502023)	0.094328 (0.00476967)	0.0951981 (0.00373364)	~	~	~
f_{13}	30	0.0956696 (0.00352899)	0.0935369 (0.00397665)	**0.091846** (0.00707918)	0.0955274 (0.00495933)	+	+	~
f_{14}	30	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	~	~	~
f_{15}	30	**0.0730003** (0.0169434)	0.0880257 (0.0115251)	0.0807431 (0.0142388)	0.0890936 (0.00986588)	–	–	–
f_{16}	2	0.0645903 (0.0231492)	0.0545825 (0.0263629)	0.0721181 (0.0186806)	**0.0539806** (0.0226797)	~	~	~
f_{17}	30	0.0910662 (0.00428958)	**0.0845474** (0.0118228)	0.0891959 (0.00724487)	0.0923214 (0.00514694)	+	~	~
f_{18}	2	4.75455e−008 (2.9688e−008)	3.63292e−008 (3.10335e−008)	5.13125e−008 (3.2954e−08)	**2.657e−008** (2.657e−008)	~	~	~
f_{19}	5	0.067335 (0.025448)	0.0738969 (0.0209749)	**0.0664662** (0.0166898)	0.0769911 (0.0160823)	~	~	~
f_{20}	5	−3.99239 (0.00164918)	−**3.99398** (0.00235545)	−3.99297 (0.00183906)	−3.99297 (0.00184151)	+	~	~

Table 2
Best and worst fitness function values obtained by all the algorithms

Function	Dim.	Best and Worst function values			
		DE	ODE	QIDE	NSDE
f_1	30	**0.0533706**	0.0710478	0.0579077	0.0801175
		0.0920816	0.0980475	0.0989701	0.0989173
f_2	30	**0.0469366**	0.0834493	0.0736855	0.0708503
		0.0852506	0.0994759	0.0995696	0.0971761
f_3	30	0.0912359	**0.0812952**	0.086481	0.085954
		0.099449	0.0991723	0.0987919	**0.0987729**
f_4	10	0.0555946	0.0782872	**0.0553663**	0.0586798
		0.0973456	0.0990834	0.0982010	0.0986525
f_5	30	**0.0550155**	0.0765341	0.0801072	0.0730851
		0.0985525	**0.0976009**	0.0986306	0.0988916
f_6	30	0.0811647	0.0799383	0.0846065	**0.0777488**
		0.0995538	0.0992613	0.0990139	**0.0979521**
f_7	2	**3.03242e−009**	1.9059e−008	7.22166e−009	5.64424e−009
		8.24678e−008	9.47894e−008	9.18626e−008	8.50966e−008
f_8	4	0.0139037	**0.00826573**	0.0132712	0.0333435
		0.0974824	0.09121890	0.0873360	**0.0790444**
f_9	30	0.0746445	0.0849655	0.0682601	**0.064171**
		0.0995713	0.098311	**0.0951216**	0.0992847
f_{10}	10	−9.64801	−9.65114	−9.64608	−**9.65368**
		−1.02642	−**9.59249**	−9.59216	−9.57805
f_{11}	30	**0.0627431**	0.0636232	0.0668308	0.0683468
		0.0944119	0.0989899	0.0976185	0.0994605
f_{12}	30	0.0849009	0.0819181	**0.0807365**	0.0887407
		0.0991914	0.0999306	**0.0989910**	0.0997806
f_{13}	30	0.0902771	0.0866648	**0.0778513**	0.0854635
		0.0996024	0.0988438	**0.0987097**	0.0998667
f_{14}	30	0.0	0.0	0.0	0.0
		0.0	0.0	0.0	0.0
f_{15}	30	**0.0441712**	0.0593778	0.0492061	0.067818
		0.0945989	0.0991945	0.0984191	0.0992816
f_{16}	2	0.0277478	**0.0129757**	0.037426	0.0297125
		0.0969767	0.0984134	0.0921157	**0.0918671**
f_{17}	30	0.0844465	**0.0593988**	0.0747125	0.0836182
		0.0975161	0.0997203	0.09992	0.0996487
f_{18}	2	**2.3063e−009**	4.64862e−009	4.43574e−009	1.11418e−008
		9.91416e−008	9.55725e−008	9.6909e−008	**8.51751e−008**
f_{19}	5	**0.00746793**	0.0291998	0.0320238	0.0483536
		0.099394	0.0995076	**0.0900442**	0.0999149
f_{20}	5	−3.99626	−**3.99737**	−3.99602	−3.99644
		−3.99019	−3.99035	−**3.99058**	−3.99027

However when we do the comparison in terms of average time taken and average number of function evaluations then the proposed QIDE emerges as a clear winner. It converges to the optimum at a faster rate in comparison to all other algorithms. Only for functions f_3 (Rosenbrock), f_{13} (Schwefel) and f_{19} (Pathalogical) QIDE took more NFE than other algorithms, whereas for the remaining 17 problems QIDE converged faster than the other algorithms. This fact is also evident from the overall acceleration rate which shows that ODE reduces the NFE up to 8.56%, whereas the reduction in NFE while using QIDE is up to 28.76% and for NSDE, the reduction is around 26.93%.

In Tables 5a and 5b we compare the performance of the proposed algorithms one by one in terms of improvement in NFE and CPU time for all the functions. From Table 5a, which gives the comparison of QIDE, we see that in comparison to basic DE, except for function f_3 (Rosenbrock), there is an improvement of more than 8% for all the test problems in terms of NFE. In case of f_{11} (Zharkhov), this improvement is more than 78%. Its comparison with ODE shows that except for f_3 and f_{13}, for every function there is an improvement of more than 6% in terms of NFE. As expected a similar behavior of QIDE can be observed in terms of average CPU time. In Table 5b, the performance of NSDE is compared with basic DE and ODE. Here also we see that in comparison to basic DE, except for function f_3, there is an improvement of more than 4% for all the test problems in terms of NFE. When compared with ODE, NSDE shows an improvement of around 2% for all the test problems except for f_3 and f_{13}. This behavior is displayed when NSDE is compared with basic DE and ODE in terms of average CPU time also. In Table 5c, we compare both the algorithms against each other. Here we see that QIDE outperforms NSDE (though marginally in some cases) for all the test problems except f_{19}. Performance curves of selected benchmark problems are illustrated in Figs. 7a–7h.

After evaluating the performance of the algorithms for solving traditional benchmark problems, their performance is further validated on a set of 7 nontraditional benchmark functions and the corresponding numerical results are reported in Ta-

Table 3
Average CPU time (in s) taken by the algorithms.

Fun.	Dim.	Average time (s)			
		DE	ODE	QIDE	NSDE
f_1	30	0.6	0.54	**0.42**	0.51
f_2	30	0.6	0.57	**0.53**	0.55
f_3	30	11.3	11.1	**11.2**	11.4
f_4	10	2.41	2.34	**2.01**	2.1
f_5	30	1.8	1.79	**1.5**	1.7
f_6	30	1.5	1.45	**1.3**	1.41
f_7	2	0.31	0.29	0.1	0.21
f_8	4	0.1	0.11	**0.06**	0.08
f_9	30	1.2	1.11	**1.1**	1.11
f_{10}	10	3.02	2.93	**2.4**	2.57
f_{11}	30	2.91	2.37	**1.5**	1.61
f_{12}	30	0.59	0.52	**0.41**	0.5
f_{13}	30	1.83	**1.24**	1.3	1.41
f_{14}	30	0.71	0.65	**0.48**	0.51
f_{15}	30	0.47	0.45	**0.39**	0.41
f_{16}	2	0.13	0.11	**0.08**	0.1
f_{17}	30	0.92	0.81	**0.71**	0.75
f_{18}	2	0.23	0.21	**0.13**	0.15
f_{19}	5	1.66	1.12	0.87	**0.51**
f_{20}	5	0.39	0.37	**0.29**	0.3
Total		32.68	30.08	**26.78**	27.89
Avg.		1.634	1.504	**1.339**	1.3945
%improv			7.955936	18.05386	14.65728

Table 4
Mean number of function evaluation of 30 runs and over all acceleration rates.

Fun.	Dim.	Mean function			
		DE	ODE	QIDE	NSDE
f_1	30	28020	26912	**24210**	26220
f_2	30	37312	36639	**33140**	35500
f_3	30	295232	**295112**	330840	332860
f_4	10	382454	361234	**264360**	265420
f_5	30	54503	53305	**49590**	52240
f_6	30	52476	51589	**48260**	49170
f_7	2	3845	3740	**3020**	3520
f_8	4	7902	7934	**6250**	6780
f_9	30	44034	41455	**33830**	35330
f_{10}	10	220356	196871	**165780**	172200
f_{11}	30	200924	196617	**43990**	44960
f_{12}	30	66154	63760	**42790**	57800
f_{13}	30	197069	**148742**	151150	155970
f_{14}	30	42423	41578	**27500**	32300
f_{15}	30	25903	24236	**20700**	22620
f_{16}	2	3913	3832	**3280**	3600
f_{17}	30	55029	52455	**46690**	47760
f_{18}	2	7367	7249	**5000**	5150
f_{19}	5	205398	150173	70860	**57540**
f_{20}	5	32419	31305	**27020**	27260
Total		1962733	1794738	**1398260**	1434200
AR			8.559239%	28.75954%	26.92842%

ble 6 in terms of best, median, worst and mean error and standard deviation. From these results QIDE performed better than DE for all the test cases with an improvement of up to 99% in the best function value for F1, F3 and F5 and an improvement up to 75% for F2, F4 and F6. For the last function F7, the improvement is around 8%. After QIDE, it was ODE that gave a good performance. Considering the complexity of the problems, NSDE also performed reasonably well though not as good as QIDE.

6.4. Analysis of control parameters F and Cr

The crucial parameters that influence the performance of DE algorithms are the scale factor (F) and the crossover rate (Cr). In this subsection we will study the effect of these parameters on proposed algorithms taking function f_6 with dimension 30.

Table 5a
QIDE vs. basic DE and ODE: % improvement in terms of NFE and average CPU time.

Function	Improvement in NFE		Improvement in time	
	QIDEvsDE	QIDEvsODE	QIDEvsDE	QIDEvsODE
f_1	13.5974304	10.0401308	30	22.22222222
f_2	11.1813894	9.549933131	11.6666667	7.01754386
f_3	−12.0610232	−12.10659004	0.88495575	−0.900900901
f_4	30.8779618	26.81751995	16.5975104	14.1025641
f_5	9.01418271	6.969327455	16.6666667	16.20111732
f_6	8.03414894	6.452926011	13.3333333	10.34482759
f_7	21.4564369	19.2513369	67.7419355	65.51724138
f_8	20.9060997	21.22510713	40	45.45454545
f_9	23.1730027	18.39343867	8.33333333	0.900900901
f_{10}	24.7671949	15.79257483	20.5298013	18.0887372
f_{11}	78.1061496	77.62655315	48.4536082	36.70886076
f_{12}	35.3175923	32.88895859	30.5084746	21.15384615
f_{13}	23.3009758	−1.618910597	28.9617486	−4.838709677
f_{14}	35.176673	33.85925249	32.3943662	26.15384615
f_{15}	20.0864765	14.58986631	17.0212766	13.33333333
f_{16}	16.1768464	14.40501044	38.4615385	27.27272727
f_{17}	15.153828	10.9903727	22.826087	12.34567901
f_{18}	32.1297679	31.02496896	43.4782609	38.0952381
f_{19}	65.5011246	52.8144207	53.6144578	31.25
f_{20}	16.6538141	13.68790928	25.6410256	21.62162162

Table 5b
NSDE vs. basic DE and ODE: % improvement in terms of NFE and average CPU time.

Function	Improvement in NFE		Improvement in time	
	NSDEvsDE	NSDEvsODE	NSDEvsDE	NSDEvsODE
f_1	6.42398287	2.57134364	15	5.55555556
f_2	4.85634648	3.1087093	8.3333333	3.50877193
f_3	−12.7452309	−12.7910759	−0.884956	−2.7027027
f_4	30.6008043	26.5240813	12.863071	10.2564103
f_5	4.15206502	1.9979364	5.5555556	5.02793296
f_6	6.30002287	4.68898409	6	2.75862069
f_7	8.45253576	5.88235294	32.258065	27.5862069
f_8	14.198937	14.5449962	20	27.2727273
f_9	19.766544	14.7750573	7.5	0
f_{10}	21.8537276	12.5315562	14.900662	12.2866894
f_{11}	77.62338	77.1332082	44.67354	32.0675105
f_{12}	12.6281102	9.34755332	15.254237	3.84615385
f_{13}	20.855132	−4.85942101	22.95082	−13.7096774
f_{14}	23.862056	22.3146857	28.169014	21.5384615
f_{15}	12.6742076	6.66776696	12.765957	8.88888889
f_{16}	7.99897777	6.05427975	23.076923	9.09090909
f_{17}	13.2093987	8.95052902	18.478261	7.40740741
f_{18}	30.0936609	28.955718	34.782609	28.5714286
f_{19}	71.9860953	61.6841909	69.277108	54.4642857
f_{20}	15.9135075	12.9212586	23.076923	18.9189189

To check the effect of the scaling factor F, it is varied from 0.1 to 0.9 in steps of 0.2 at crossover rate $Cr = 0.9$ that is shown in Fig. 8. It manages the two operations, *exploration* and *exploitation*. Larger the F the exploration is high but the convergence is slow down while F is small exploitation is high but there is a chance to stuck in a local optimum.

To investigate the sensitivity of the proposed algorithms to variations of crossover rate Cr, we experimented with different values of Cr ranging from 0.1 to 0.9 with steps of 0.2 at scaling factor $F = 0.5$. Results, plotted in Fig. 9, show how the performance of algorithms changes with the different values of Cr. Larger the value of Cr faster the convergence while the value of Cr is small the convergence is slow. So, the values of control parameters of the algorithms should be carefully chosen.

7. Discussion and conclusions

In the present paper we have proposed two schemes based on Quadratic Interpolation (QI) and nonlinear simplex method (NSM) to generate the initial population of DE. These schemes provide the information of the potential regions of the search space and their conjugation with random numbers prevents them from becoming a totally greedy search in nature. Conse-

Table 5c
QIDE vs. NSDE: %improvement in terms of NFE and average CPU time.

Function	Improvement in NFE	Improvement in time
f_1	7.665904	17.6470588
f_2	6.647887	3.63636364
f_3	0.606862	1.75438596
f_4	0.399367	4.28571429
f_5	5.072741	11.7647059
f_6	1.850722	7.80141844
f_7	14.20455	52.3809524
f_8	7.817109	25
f_9	4.245684	0.9009009
f_{10}	3.728223	6.61478599
f_{11}	2.157473	6.83229814
f_{12}	25.96886	18
f_{13}	3.090338	7.80141844
f_{14}	14.86068	5.88235294
f_{15}	8.488064	4.87804878
f_{16}	8.888889	20
f_{17}	2.240369	5.33333333
f_{18}	2.912621	13.3333333
f_{19}	−23.1491	−50.9803922
f_{20}	0.880411	3.33333333

Table 6
Comparison of proposed algorithms with DE and ODE for nontraditional shifted functions in terms of error (best median, worst and mean) and standard deviation (std.).

Problem	Dim	Error value	DE	ODE [33]	NSDE	QIDE
F_1	500	Best	2, 636.54	15.66	4.98	**3.48**
		Median	3, 181.45	36.61	5.89	**5.32**
		Worst	4, 328.80	292.65	8.28	**7.57**
		Mean	3, 266.24	80.17	5.03	**4.86**
		Std.	409.68	79.24	5.78	**4.34**
F_2	500	Best	79.74	3.60	**2.78**	19.82
		Median	82.39	**4.86**	4.93	11.88
		Worst	85.92	11.91	**7.32**	12.26
		Mean	82.93	5.78	**3.92**	11.87
		Std.	2.09	**2.37**	2.49	1.93
F_3	500	Best	76, 615, 772.08	**39, 718.90**	763,323.45	727,996.00
		Median	119, 733, 49.20	**137, 279.03**	782,301.74	731, 546.21
		Worst	169, 316,779.50	**407, 661.64**	789,873.56	732, 763.93
		Mean	123, 184, 755.70	**154, 306.34**	779,289.90	730, 473.25
		Std.	29, 956, 737.58	**114, 000.53**	117,328.93	116,325.43
F_4	500	Best	5, 209.99	2, 543.51	1,183.84	**1, 155.15**
		Median	5, 324.57	4, 279.56	4,382.63	**3, 243.87**
		Worst	5, 388.24	6, 003.94	4,829.43	**4, 478.90**
		Mean	5, 332.59	4, 216.34	**4,132.54**	4, 212.76
		Std.	43.82	1, 017.94	67.32	**58.60**
F_5	500	Best	24.29	1.25	1.09	**0.31**
		Median	24.71	1.55	1.32	**0.87**
		Worst	27.59	2.13	1.58	**0.96**
		Mean	25.16	1.75	1.38	**0.56**
		Std.	1.10	0.37	0.18	**0.05**
F_6	500	Best	4.66	2.49	2.37	**1.18**
		Median	4.97	4.12	3.83	**1.47**
		Worst	5.15	6.73	5.18	**1.56**
		Mean	4.94	4.51	4.27	**1.25**
		Std.	0.17	1.44	1.06	**0.07**
F_7	500	Best	−3683.07	−3957.85	−3983.32	−**3992.76**
		Median	−3575.13	−3834.07	−**3889.49**	−3836.65
		Worst	−3565.73	−3830.36	−3738.54	−**3833.21**
		Mean	−3593.75	−3851.82	−**3883.85**	−3863.59
		Std.	32.74	38.80	34.54	**29.31**

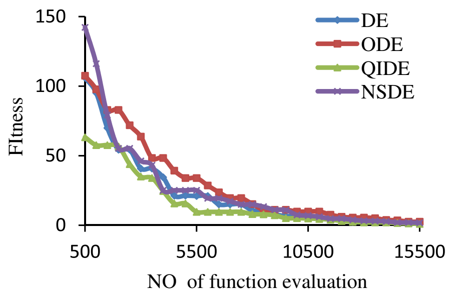

Fig. 7a. Sphere (f_1) function.

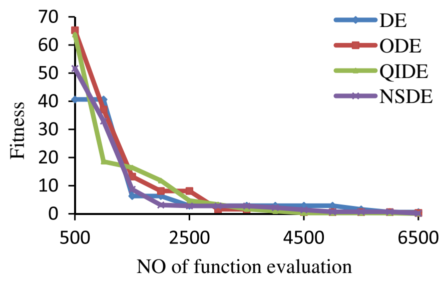

Fig. 7b. Colville (f_8) function.

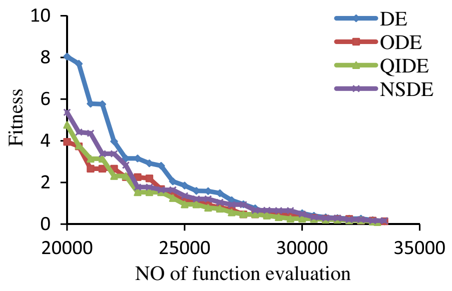

Fig. 7c. Axis parallel (f_2) function.

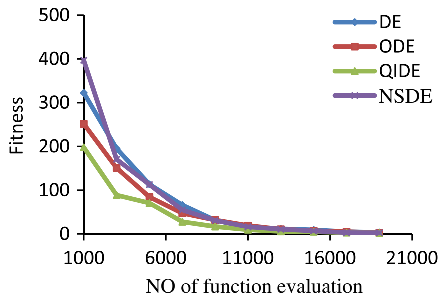

Fig. 7d. Griewenk (f_5) function.

quently the initial population, though being biased towards the solution preserves diversity also. The only structural difference between the proposed DE algorithms (QIDE and NSDE) and the basic DE lies is the initialization phase only.

From the empirical studies and graphic illustrations we can say that the proposed schemes enhance the working of basic DE in terms of average CPU time and NFEs without compromising with the quality of solution. We see that the improvement in average CPU time while using QIDE and NSDE is 18% and 14% respectively in comparison to basic DE. This is twice the improvement obtained by using ODE (7%). This evidently shows that the proposed schemes significantly improve the per-

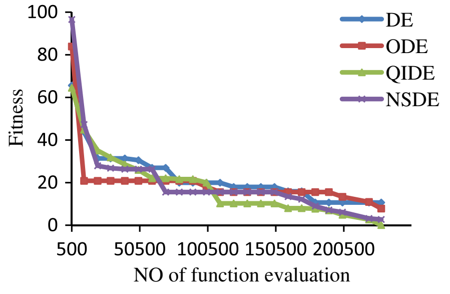

Fig. 7e. Restrigin (f_4) function.

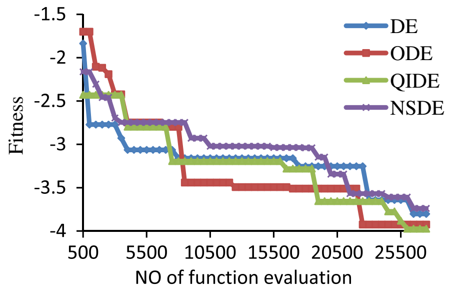

Fig. 7f. Inverted (f_{20}) cosine.

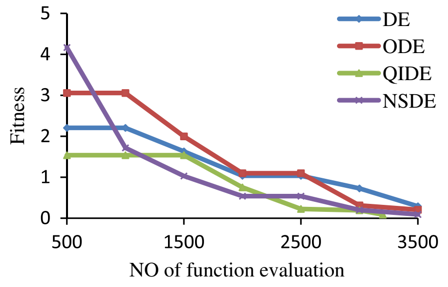

Fig. 7g. Tripod (f_{16}) function.

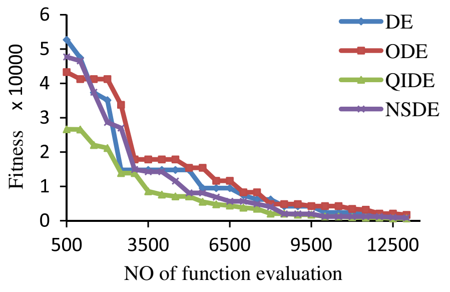

Fig. 7h. Step (f_{14}) function.

formance of basic DE. Such a situation is very useful in real world scenarios where sometimes the optimum has to be located in smallest possible time.

The only functions where QIDE and NSDE did not perform as expected were Rosenbrock (f_3) and f_{13} (Schwefel) and require further investigation.

Also we would like to mention that other than the process of initial population construction, we have not made use of any other additional feature/parameter in the basic structure of DE. Though we have applied the proposed schemes in basic DE,

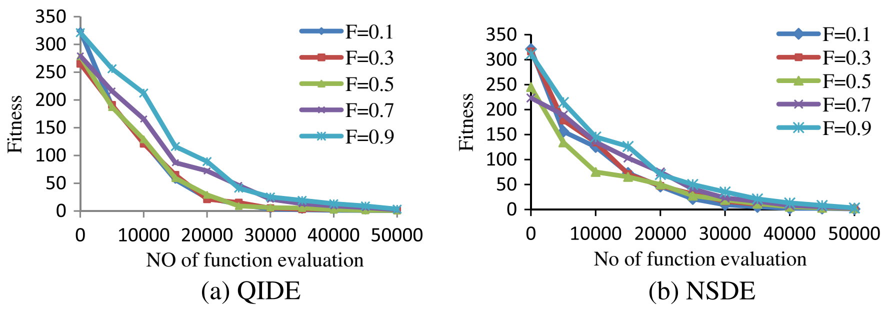

Fig. 8. Convergence graphs of proposed algorithms with different values of F and $Cr = 0.9$.

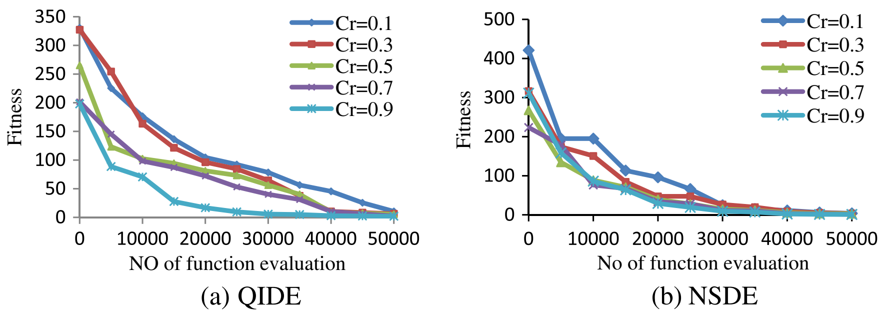

Fig. 9. Convergence graphs of proposed algorithms with different values of Cr and $F = 0.5$.

they can be applied in any evolutionary algorithm which makes use of randomly generated initial points. As a concluding statement it can be said that providing the initial population with some extra information of search space is an important help for the DE algorithm, since it may lead to faster convergence and improve the quality of the solutions provided by the algorithm.

Acknowledgments

The authors thank the anonymous reviewers and the Editor for their valuable comments and suggestions to improve the quality of the paper.

Appendix A

1. Sphere function:

$$f_1(x) = \sum_{i=1}^{n} x_i^2, \quad \text{with } -5.12 \leqslant x_i \leqslant 5.12, \ \min f_1(0,\ldots,0) = 0.$$

2. Axis parallel hyper-ellipsoid:

$$f_2(x) = \sum_{i=1}^{n} i x_i^2, \quad \text{with } -5.12 \leqslant x_i \leqslant 5.12, \ \min f_2(0,\ldots,0) = 0.$$

3. Rosenbrock's valley:

$$f_3(x) = \sum_{i=1}^{n-1} \left[100(x_{i+1}^2 - x_i^2) + (1 - x_i^2)\right] \quad \text{with } -2 \leqslant x_i \leqslant 2, \ \min f_3(1,\ldots,1) = 0.$$

4. Rastrigin's function:

$$f_4(x) = 10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i)) \quad \text{with } -5.12 \leqslant x_i \leqslant 5.12, \ \min f_4(0,\ldots,0) = 0.$$

5. Griewank function:

$$f_5(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \quad \text{with } -600 \leqslant x_i \leqslant 600,\ \min f_5(0,\ldots,0) = 0.$$

6. Ackley's function:

$$f_6(X) = -20 * \exp\left(-.2\sqrt{1/\mathrm{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(1/n\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e, \quad \text{with } -32 \leqslant x_i \leqslant 32,$$
$$\min f_6(0,\ldots,0) = 0.$$

7. Beale function:

$$f_7(x) = [1.5 - x_1(1 - x_2)]^2 + \left[2.25 - x_1\left(1 - x_2^2\right)\right]^2 + \left[2.625 - x_1\left(1 - x_2^3\right)\right]^2 \quad \text{with } -4.5 \leqslant x_i \leqslant 4.5,\ \min f_7(3, 0.5) = 0.$$

8. Colville function:

$$f_8(x) = 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2 + 90\left(x_4 - x_3^2\right)^2 + (1 - x_3)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2)19.8(x_2 - 1)(x_4 - 1) \quad \text{with,}$$
$$-10 \leqslant x_i \leqslant 10,\ \min f_8(1,1,1,1) = 0.$$

9. Levy function:

$$f_9(x) = \sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)(1 + \sin^2(3\pi x_{i+1})) + (x_n - 1)(1 + \sin^2(2\pi x_n)) \quad \text{with } -10 \leqslant x_i \leqslant 10,$$
$$\min f_9(1,\ldots,1) = 0$$

10. Michalewicz function:

$$f_{10}(x) = -\sum_{i=1}^{n}\sin(x_i)\left(\sin(ix_i^2/\pi)\right)^{2m} \quad \text{with } 0 \leqslant x_i \leqslant \pi, m = 10,\ \min f_{10(n=10)} = -9.66015.$$

11. Zakharov function:

$$f_{11}(x) = \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} 0.5ix_i\right)^2 + \left(\sum_{i=1}^{n} 0.5ix_i\right)^4 \quad \text{with } -5 \leqslant x_i \leqslant 10,\ \min f_{11}(0,\ldots,0) = 0.$$

12. Schawefel's problem 2.22:

$$f_{12}(x) = \sum_{i=1}^{n}|x_i| + \prod_{i=1}^{n}|x_i| \quad \text{with } -10 \leqslant x_i \leqslant 10,\ \min f_{12}(0,\ldots,0) = 0.$$

13. Schwefel's problem 2.21:

$$f_{13}(x) = \max_i\{|x_i|, 1 \leqslant i \leqslant n\} \quad \text{with } -100 \leqslant x_i \leqslant 100,\ \min f_{13}(0,\ldots,0) = 0.$$

14. Step function:

$$f_{14}(x) = \sum_{i=1}^{n}(\lfloor x_i + 0.5\rfloor)^2 \quad \text{with } -100 \leqslant x_i \leqslant 100,\ \min f_{14}(-0.5 \leqslant x_i \leqslant 0.5) = 0.$$

15. Quartic function:

$$f_{15}(x) = \sum_{i=1}^{n} ix_i^4 + \mathit{random}[0,1) \quad \text{with } -1.28 \leqslant x_i \leqslant 1.28,\ \min f_{15}(0,\ldots,0) = 0.$$

16. Tripod function:

$$f_{16}(x) = p(x_2)(1 + p(x_1)) + |(x_1 + 50p(x_2)(1 - 2p(x_1)))| + |(x_2 + 50(1 - 2p(x_2)))| \quad \text{with}$$
$$-100 \leqslant x_i \leqslant 100,\ \min f_{16}(0, -50) = 0 \text{ where } p(x) = 1 \text{ for } x > 0 \text{ otherwise } p(x) = 0.$$

17. Alpine function:

$$f_{17}(x) = \sum_{i=1}^{n}|x_i\sin(x_i) + 0.1x_i| \quad \text{with } -10 \leqslant x_i \leqslant 10,\ \min f_{17}(0,\ldots,0) = 0.$$

18. Cshaffer's function 6:

$$f_{18}(x) = 0.5 + \frac{\sin^2\sqrt{(x_1^2 + x_2^2)} - 0.5}{1 + 0.01(x_1^2 + x_2^2)^2} \quad \text{with } -10 \leqslant x_i \leqslant 10,\ \min f_{18}(0,0) = 0.$$

19. Pathological function:

$$f_{19}(x) = \sum_{i=1}^{n-1}\left(0.5 + \frac{\sin^2\sqrt{(100x_i^2 + x_{i+1}^2)} - 0.5}{1 + 0.001(x_i^2 + x_{i+1}^2 - 2x_i x_{i+1})^2}\right) \quad \text{with } -100 \leqslant x_i \leqslant 100,\ \min f_{19}(0,\ldots,0) = 0.$$

20. Inverted cosine wave function:

$$f_{20}(x) = -\sum_{i=1}^{n-1}\left(\exp\left(\frac{-(x_i^2 + x_{i+1}^2 + 0.5x_i x_{i+1})}{8}\right)\cos\left(4\sqrt{x_i^2 + x_{i+1}^2 + 0.5x_i x_{i+1}}\right)\right) \quad \text{with } -5 \leqslant x_i \leqslant 5,$$
$$\min f_{20}(0,\ldots,0) = -n + 1.$$

References

[1] R. Storn, K. Price, DE-a simple and efficient heuristic for global optimization over continuous space, Journal of Global Optimization 11 (4) (1997) 341–359.
[2] K. Price, An introduction to DE, in: D. Corne, D. Marco, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, London (UK), 1999, pp. 78–108.
[3] R. Angira, B.V. Babu, Optimization of process synthesis and design problems: a modified differential evolution approach, Chemical Engineering Science 61 (2006) 4707–4721.
[4] E. Cuevas, D. Zaldivar, M.P. Cisneros, A novel multi-threshold segmentation approach based on differential evolution optimization, Expert Systems with Applications 37 (2010) 5265–5271.
[5] W. Kwedlo, A clustering method combining differential evolution with the *K*-means algorithm, Pattern Recognition Letters 32 (2011) 1613–1621.
[6] K. Hammouche, M. Diaf, P. Siarry, A comparative study of various meta-heuristic techniques applied to the multilevel thresholding problem, Engineering Applications of Artificial Intelligence 23 (2010) 676–688.
[7] M.F. Tasgetiren, P.N. Suganthan, Q.-K. Pan, An ensemble of discrete differential evolution algorithms for solving the generalized traveling salesman problem, Applied Mathematics and Computation 215 (2010) 3356–3368.
[8] X. Zhang, J. Zhou, C. Wang, C. Li, L. Song, Multi-class support vector machine optimized by inter-cluster distance and self-adaptive deferential evolution, Applied Mathematics and Computation 218 (2012) 4973–4987.
[9] R. Thangaraj, M. Pant, A. Abraham, New mutation schemes for differential evolution algorithm and their application to the optimization of directional over-current relay settings, Applied Mathematics and Computation 216 (2010) 532–544.
[10] M. Ali, P. Siarry, M. Pant, An efficient differential evolution based algorithm for solving multi-objective optimization problems, European Journal of Operational Research 217 (2012) 404–416.
[11] P.K. Bergey, C. Rgsdale, Modified differential evolution: a greedy random strategy for genetic recombination, Omega 33 (2005) 255–265.
[12] P. Kaleo, M.M. Ali, A numerical study of some modified differential evolution algorithms, European Journal of Operational Research 169 (2006) 1176–1184.
[13] M.M. Ali, Differential evolution with preferential crossover, European Journal of Operational Research 181 (2007) 1137–1147.
[14] N. Noman, H. Iba, Enhancing differential evolution performance with local search for high dimensional function optimization, in: GECCO'05, Washington, DC, USA, 2005, pp. 967–974.
[15] A.K. Qin, P.N. Suganthan, Self Adaptive Differential Evolution Algorithm for Numerical Optimization, IEEE, 2005. pp. 1785–1791.
[16] H.Y Fan, J. Lampinen, G.S. Dulikravich, Improvements to mutation donor formulation of differential evolution, in: International Congress on Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems, EUROGEN, 2003, pp. 1–12.
[17] Musrrat Ali, Millie Pant, Improving the performance of differential evolution algorithm using Cauchy mutation, Soft Computing 15 (2011) 991–1007.
[18] L. Wang, F.-Z. Huang, Parameter analysis based on stochastic model for differential evolution algorithm, Applied Mathematics and Computation 217 (2010) 3263–3273.
[19] R. Thangaraj, M. Pant, A. Abraham, P. Bouvry, Particle swarm optimization: hybridization perspectives and experimental illustrations, Applied Mathematics and Computation 217 (2011) 5208–5226.
[20] V.C. Mariani, L.G.J. Luvizotto, F.A. Guerra, L.S. Coelho, A hybrid shuffled complex evolution approach based on differential evolution for unconstrained optimization, Applied Mathematics and Computation 217 (2011) 5822–5829.
[21] S. Kimura, K. Matsumura, Genetic Algorithms using low discrepancy sequences, in: Proc of GEECO, 2005, pp. 1341–1346.
[22] X.H. Nguyen, Q.Uy. Nguyen, R.I. Mckay, P.M. Tuan, Initializing PSO with randomized low-discrepancy sequences: the comparative results, in: Proc. of IEEE Congress on Evolutionary Algorithms, 2007, pp. 1985–1992.
[23] K.E. Parsopoulos, M.N. Vrahatis, Initializing the particle swarm optimization using nonlinear simplex method, in: Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation, WSEAS press, 2002, pp. 216–221.
[24] Millie Pant, Radha Thangaraj, Crina Grosan, Ajith Abraham, Improved particle swarm optimization with low-discrepancy sequences, in: Proc. of IEEE Congress on Evolutionary Algorithms, 2008, pp. 3011–3018.
[25] Shahryar Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, A novel population initialization method for accelerating evolutionary algorithms, Computer and Applied Mathematics with Application 53 (2007) 1605–1614.
[26] S. Rahnamayan, H.R. Tizhoosh, M.A. Salman, Opposition based-differential evolution, IEEE Transaction on Evolutionary Computation 12 (1) (2008) 64–79.
[27] Shahryar Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, Opposition versus randomness in soft computing techniques, Applied Soft Computing 8 (2) (2008) 906–918.
[28] Li Zhang, Yong-Chang Jiao, Hong Li, Fu-Shun Zhang, Hybrid differential evolution and the simplified quadratic interpolation for global optimization, in: GEC'09, ACM, Shanghai, China, 2009. 978-1-60558-326-6/09/06.
[29] M.M. Ali, A. Torn, Population set based global optimization algorithms: some modifications and numerical studies, 2003, Available from: <www.ima.umn.edu/preprints/>.
[30] C. Mohan, K. Shanker, A controlled random search technique for global optimization using quadratic approximation, Asia-Pacific Journal of Operational Research 11 (1994) 93–101.

[31] J.A. Nelder, R. Mead, A simplex method for function minimization, Computer Journal 7 (1965) 308–313.
[32] K. Tang, X. Yao, P.N. Suganthan, C. MacNish, Y.P. Chen, C.M. Chen, Z. Yang, Benchmark functions for the CEC 2008 special session and competition on large scale global optimization, Technical Report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2007, Online available at: <http://nical.ustc.edu.cn/cec08ss.php>.
[33] S. Rahnamayan, G.G. Wang, Solving large scale optimization problems by opposition based differential evolution (ODE), WSEAS Transaction on Computers 7 (2008) 1792–1804.