Article

TRANSACTIONS OF THE INSTITUTE OF MEASUREMENT & CONTROL

Transactions of the Institute of Measurement and Control
34(6) 691–704

DOI: 10.1177/0142331211403032
tim.sagepub.com

SAGE

A simplex differential evolution algorithm: development and applications

Musrrat Ali[1], Millie Pant[1] and Ajith Abraham[2]

Abstract

Population-based heuristic optimization methods like differential evolution (DE) depend largely on the generation of the initial population. The initial population not only affects the search for several iterations but often also has an influence on the final solution. The conventional method for generating the initial population is the use of computer-generated pseudo-random numbers, which may not be very effective. In the present study, we have investigated the potential of generating the initial population by integrating the non-linear simplex method of Nelder and Mead with pseudo-random numbers in a DE algorithm. The resulting algorithm named the non-linear simplex DE is tested on a set of 20 benchmark problems with box constraints and two real life problems. Numerical results show that the proposed scheme for generating the random numbers significantly improves the performance of DE in terms of fitness function value, convergence rate and average CPU time.

Keywords

Crossover, differential evolution, initial population, random numbers, stochastic optimization

Introduction

Nature-inspired algorithms (NIA) or bio-inspired algorithms are the techniques that work on the principles derived from natural or biological metaphors. Some popular NIA include genetic algorithms (GA) that work on Darwin's concept of survival of the fittest, particle swarm optimization (PSO) and ant colony optimization (ACO) that work on the collective co-operative behaviour of birds, ants etc. and differential evolution (DE) that uses mutation, crossover and selection to solve a particular problem. A brief review of some of the latest NIA can be found in (Tang and Wu, 2009). These algorithms typically work on a large population set, are gradient-free methods and do not depend on the auxiliary properties of the problem. Also, they are easy to program and have a wide applicability. Because of all these features, NIA have gained considerable attention in the global optimization community in the past few decades, as these are more flexible than the gradient-based classical techniques (which are problem specific) in dealing with a wide spectrum of problems.

However, it is important to note here that NIA are not completely flawless. Sometimes, while dealing with multimodal optimization functions (i.e. functions having several local and global optima), these algorithms may become stuck in a local optimum, or in worse cases may stop prematurely without even reaching a local optimum. Then, they may also face the problem of stagnation, a condition where an algorithm may accept new points but show no improvement in the function value. In NIA, two antagonist factors – exploration and exploitation (also known as diversification and intensification) – work simultaneously and an imbalance between them may result in the feeble working of NIA. Subsequently, several modifications have been suggested to improve the working of these algorithms (Ali et al., 2009; Li et al., 2008; Pant et al., 2008, 2009a,b,c; Wang et al., 2006; Zhu et al., 2009). These modifications mainly consist of improving parameter settings, development of new operators or hybridization of two methods.

Interestingly, generation of the initial population, which plays a prominent role in the NIA, is not discussed much in literature. It is natural to think that if the initial population itself has the information of the optimum regions of the search space, it will help the NIA in locating the global optimum more quickly. The most popular method of generating the initial population is the use of computer-generated pseudo-random numbers within some specified range. This method, though simple to use and easy to apply, may not be very efficient, as it does not provide the algorithm with any prior information about the search space.

In the present study, we have investigated the effect of using the Nelder–Mead non-linear simplex method (NSM) in conjugation with computer-generated pseudo-random numbers to generate the initial population. We have embedded the proposed scheme on DE and the resulting algorithm is named non-linear simplex DE (NSDE). Our algorithm is

[1]Department of Paper Technology, Indian Institute of Technology Roorkee, Saharanpur, India
[2]Machine Intelligence Research Labs (MIR Labs), Scientific Network for Innovation and Research Excellence, Auburn, WA, USA

Corresponding author:
Musrrat Ali, Department of Paper Technology, Indian Institute of Technology Roorkee, Saharanpur campus, Saharanpur -247001, India
Email: musrrat.iitr@gmail.com

different from that of Parsopoulas and Vrahatis (2002) in a twofold manner: firstly, we have applied it on DE, and secondly, instead of using pure NSM to generate the initial population, we have used NSM in conjugation with pseudo-random numbers. This is done to provide more diversity to the initial population. A pure NSM may result in creating a bias in the search towards some local region, whereas a pure random will have no prior information of the search space (implying no bias). Thus a combination of the two may help in proving a good initial solution.

As mentioned in the previous paragraph, the focus of the present study is to apply the novel population generation method in the DE algorithm. We have chosen DE (Storn and Price, 1995, 1997) because it is comparatively a new addition to a class of population-based search algorithms, it has shown promising results in comparison with other population-based search algorithms and it has been successfully applied to a wide range of problems (Das and Konar, 2006; Das et al., 2008; Joshi and Sanderson, 1999; Lu and Wang, 2001; Omran et al., 2005; Rogalsky et al., 1999; Wang and Jang, 2000). Also it has undergone several modifications (Ali, 2007; Bergey and Ragsdale, 2005; Das et al., 2009: Kaleo and Ali, 2006). However, to the best of our knowledge, the approach by which we have integrated NSM and pseudo-random numbers has not been studied so far.

Performance of the proposed NSDE is compared with basic DE and DE initialized by opposition-based learning (ODE), which is a recently modified version of DE (Rahnamayan et al., 2007), on a set of 20 unconstrained benchmark problems and two real life problems.

The remainder of the paper is organized in following manner: in Section 2, we give a brief description of DE and NSM. In Section 3, the proposed algorithm is explained. Section 4 deals with experimental settings and parameter selection. Benchmark problems considered for the present study are given in Section 5. The performances of the proposed algorithms are compared with basic DE and ODE in Section 6. The conclusions based on the present study are finally drawn in Section 7.

Differential evolution and non-linear simplex method

In this section, we give a brief description of the basic DE and NSM.

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 are briefly described in this section.

Mutation. The mutation operator is the prime operator of DE and it is the implementation of this operation that makes DE different from other evolutionary algorithms (EAs). 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.

$$\text{DE/rand/1}: V_{i,G+1} = X_{r_1,G} + F^*(X_{r_2,G} - X_{r_3,G}) \tag{1a}$$

$$\text{DE/rand/2}: V_{i,G+1} = X_{r_1,G} + F^*(X_{r_2,G} - X_{r_3,G}) + F^*(X_{r_4,G} - X_{r_5,G}) \tag{1b}$$

$$\text{DE/best/1}: V_{i,G+1} = X_{best,G} + F^*(X_{r_1,G} - X_{r_2,G}) \tag{1c}$$

$$\text{DE/best/2}: V_{i,G+1} = X_{best,G} + F^*(X_{r_1,G} - X_{r_2,G}) + F^*(X_{r_3,G} - X_{r_4,G}) \tag{1d}$$

$$\text{DE/rand-to-best/1}: V_{i,G+1} = X_{r_1,G} + F^*(X_{best,G} - X_{r_2,G}) + F^*(X_{r_3,G} - X_{r_4,G}) \tag{1e}$$

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 we shall refer to it as the basic version.

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:

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

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

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:

$$X_{i.G+1} = \begin{cases} U_{i.G+1} \; iff(U_{i.G+1}) \le f(X_{i.G}) \\ X_{i.G} \qquad\qquad otherwise \end{cases} \tag{3}$$

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, the trial vector is not compared against all the individuals in the current generation, but only against one individual, its counterpart, in the current generation. The pseudo-code of algorithm is given here.

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 the *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 Equation (1a).

Step 4: Recombine each target vector X_i with perturbed individual generated in step 3 to generate a trial vector U_i using Equation (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.

Step 6: Calculate the objective function value for vector U_i.

Step 7: Choose the better of the two (function value at target and trial point) using Equation (3) for the next generation.

Step 8: Check whether the convergence criterion is met, if yes then stop; otherwise go to step 3.

An illustration of mutation, crossover and selection are shown in Figures 1, 2 and 3, respectively.

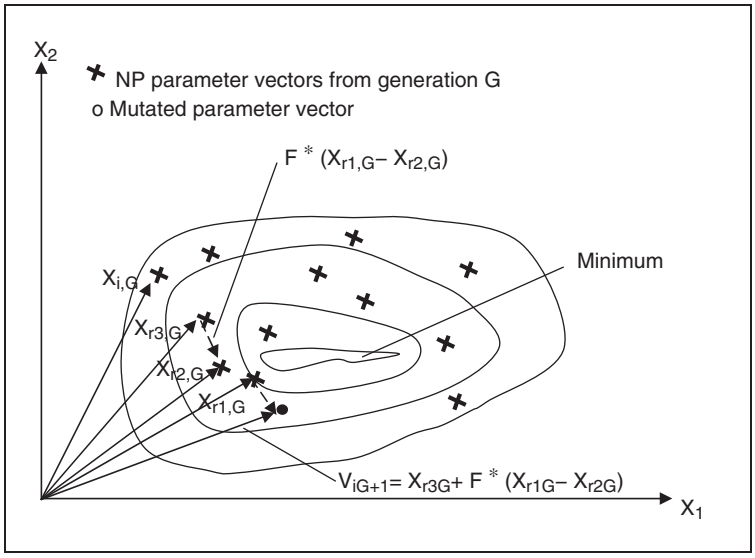

Figure 1 Illustration of process of mutation on a two-dimensional cost function showing its contour lines for generating $V_{i,G+1}$ by different vectors.

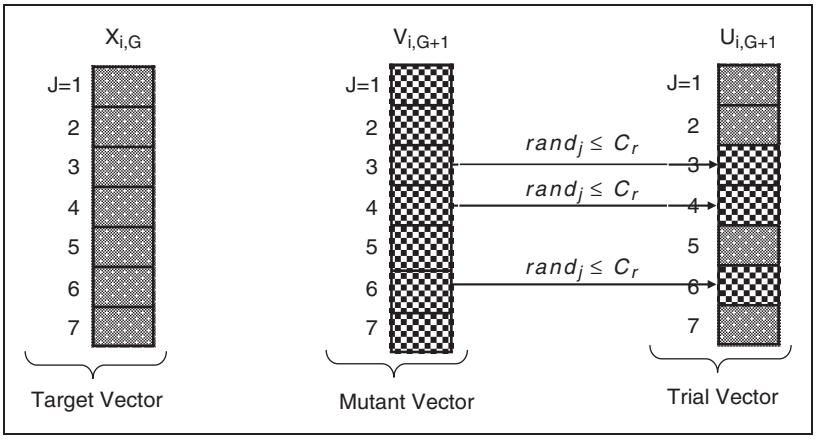

Figure 2 Illustration of the crossover process for $n=7$ parameters.

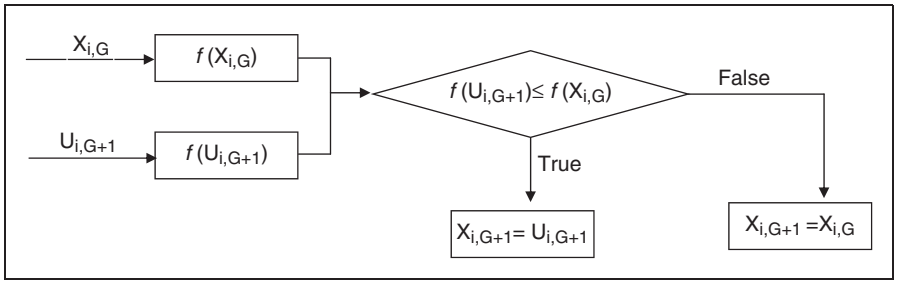

Figure 3 Illustration of the selection process for minimization problem.

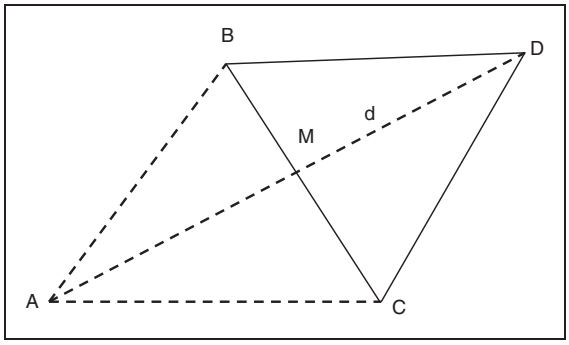

Figure 4 Reflection of A to D.

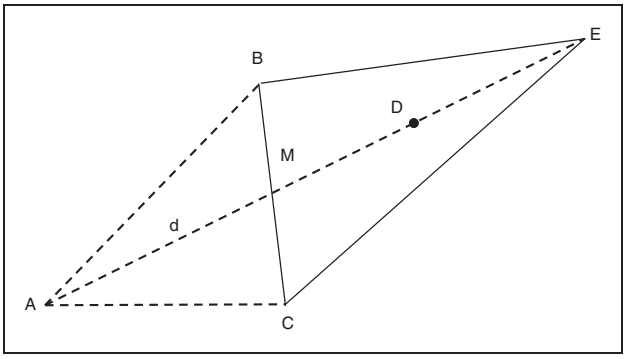

Figure 5 Expansion of D to E.

Non-linear simplex method (NSM)

The NSM search method was first introduced by Nelder and Mead in 1965 and is perhaps one of the most widely used local direct search methods for non-linear unconstrained optimization. The NSM is a derivative-free line search method that was specially designed for solving traditional unconstrained problems of minimization type, like the non-linear least-squares problem, non-linear simultaneous equations and function minimization (Olsson and Nelson, 1975). 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, and the algorithm again performs 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.

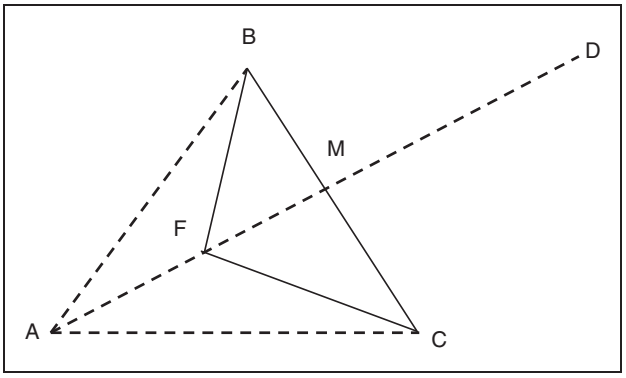

Figure 6 Contraction of D to F.

The sequence of transformations reflection, expansion, contraction and reduction are illustrated in Figures 4, 5, 6 and 7, respectively, and the steps are explained in more detail in Section 3.1.

Proposed NSDE algorithm

In this section, we describe the proposed NSDE algorithm and discuss the effect of embedding the proposed scheme in basic DE (as given in Section 2) on two simple benchmark examples taken from the literature.

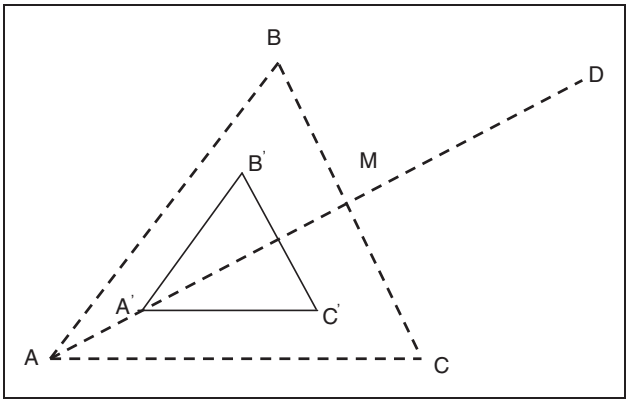

Figure 7 Reduction of ABC to A'B'C'.

Non-linear simplex differential evolution (NSDE)

The NSDE uses the NSM developed by Nelder and Mead (1965), in conjugation with a uniform random number to construct the initial population. It starts like a usual population-based search method by generating a set (say P) of size NP with the help of pseudo-random numbers within the given range. On the set P, NSM is applied NP times to generate another population set Q. The total population of size now becomes $2\times NP$, from which the initial population is constructed by selecting the NP fittest (elite) points. Once the initialization process is complete, the NSDE algorithm works like the basic DE.

The computational steps of NSDE are outlined as follows:

1. Generate a population set P of size NP uniformly as in step 1 of DE and set $k=1$.
2. Generate a point by the NSM method, which has the following main operations.

   2.1 Select $n+1$ points from population P randomly and evaluate function at these points.
   2.2 Calculate the centroid of these points excluding the worst point, say $X_{\max}$ at which function is maximum.
   2.3 *Reflection*: Reflect $X_{\max}$ through the centroid to a new point X_1, and calculate the function value at this point.
   2.4 *Expansion:* If $f(X_1)<=f(X_{\min})$ then perform expansion to generate a new point X_2 in the expanded region otherwise go to step 2.5. If $f(X_2)<f(X_{\min})$ then include point X_2 to the population Q otherwise point X_1 is included to population Q and go to step 3.
   2.5 *Contraction:* If $f(X_1)<f(X_{\max})$ then produce a new point X_3 by contraction otherwise go to step 2.6. If $f(X_3)<f(X_{\max})$ then add point X_3 to the population otherwise include point X_1 to the population Q and go to step 3.
   2.6 *Reduction:* Reduction is performed when either of the conditions mentioned above (from step 2.3 to step 2.5) are not satisfied. In this step we have replaced the original reduction method of Nelder–Mead, by generating uniformly distributed random points say X_{rand} within the specified range and include these in the population Q and go to step 3.

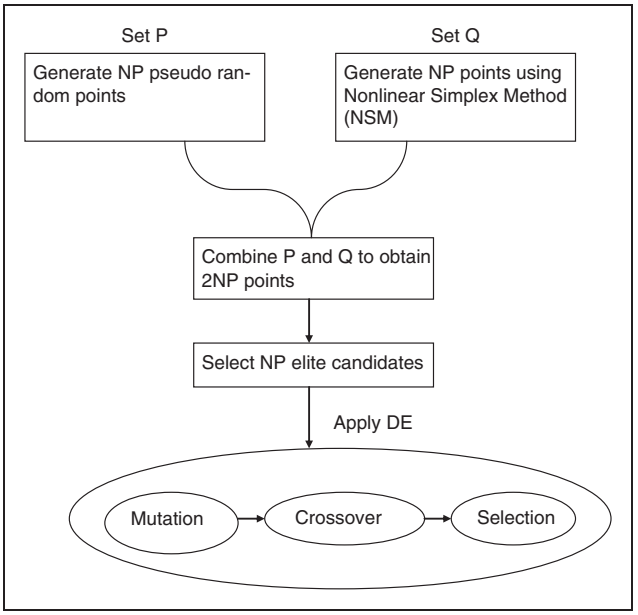

Figure 8 A schematic of the non-linear simplex differential evolution (NSDE) algorithm.

3. If $k<NP$ go to step 2.1 with $k=k+1$, else stop.

With the use of the NSM method, the initial population is provided with the information of the potential regions that possess each particle as a vertex of the NSM simplex in each step. The algorithm is 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.

A schematic of the NSDE is shown in Figure 8.

Effects of using the proposed NSDE to generate initial population

In the NSM, the simplex can varies its shape, size and orientation to adapt itself to the local contour of the objective function, so it is extremely flexible and suitable for exploring difficult terrains. By applying the NSM in the initial phase, the exploratory capacity of the algorithm is enhanced and we get a collection of fitter individuals, which in turn increases the efficiency of the algorithm. Consequently, the probability of obtaining the optimum in fewer numbers of function evaluations (NFEs) increases implying faster convergence rate.

The working of the proposed NSDE algorithm is shown with the help of two benchmark problems – the Rosenbrock function and the Griewank function. Both are well known and challenging optimization problems. Whereas for the Rosenbrock function, the optimum lies in a narrow valley, Griewank is a highly multimodal problem having several local and global minima.

The initial generation of 100 points within the range $[-2, 2]$ for Rosenbrock function and within the range $[-600, 600]$ for the Griewank function using basic DE, ODE and the proposed

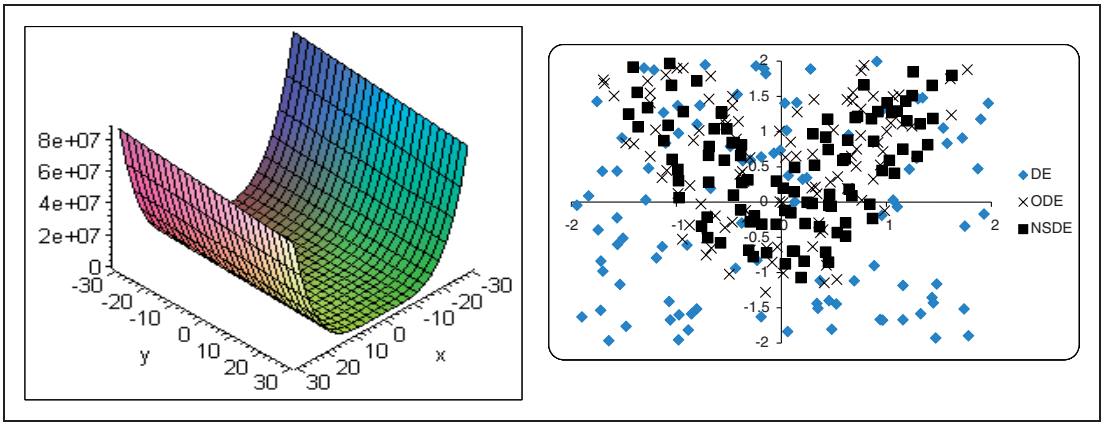

Figure 9 Three-dimensional model of the Rosenbrock function and initial population consisting of 100 points in the range [−2, 2] using basic differential evolution (DE), opposition-based learning DE (ODE) and non-linear simplex DE (NSDE).

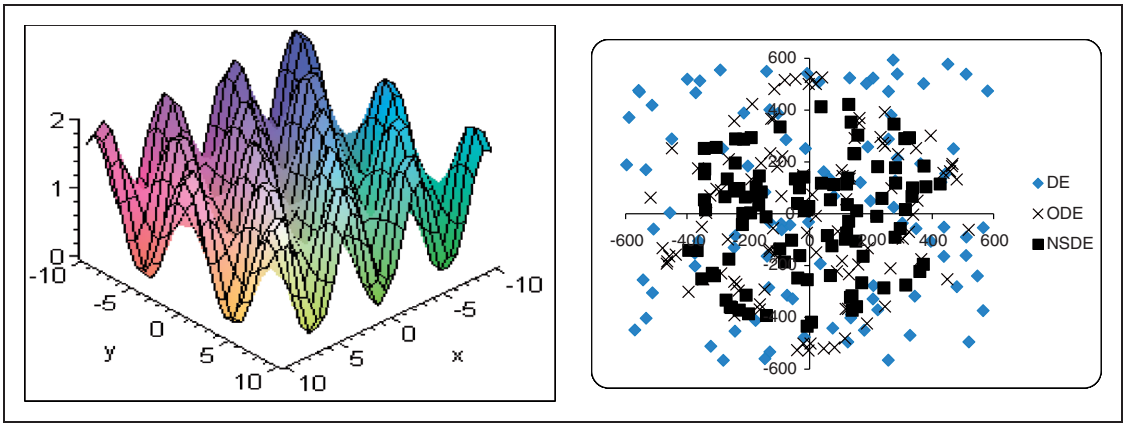

Figure 10 Three-dimensional model of the Griewank function and initial population consisting of 100 points in the range [−600, 600] using basic differential evolution (DE), opposition-based learning DE (ODE) and non-linear simplex DE (NSDE).

NSDE are depicted in Figures 9 and 10, respectively. From these illustrations, we can observe that the search space gets concentrated around the global optima, which lie at (1, 1) with objective function value zero, for the two-dimensional Rosenbrock function, and at (0, 0) with objective function value 0, for the Griewank function when the initial population is constructed using the NSDE. The large search domain, [−600, 600], of the Griewank function is contracted to the range of around [−400, 400] while using the NSDE.

Experimental set-up

DE has four control parameters, namely population size (say NP), scaling factor F and crossover rate C_r. As a general rule, for an 'n'-dimensional problem, an effective NP is between $3\times n$ and $5\times n$, but it can always be modified depending on the complexity of the problem. For the present study, we performed several experiments to fine tune the parameters. We observed that for problems up to dimension 30, a population size of $3\times n$ is sufficient. However, here we have taken fixed population size NP=100, which is slightly larger than $3\times n$. Values of scaling factor F, outside the range of 0.4–1.2 are rarely found to be effective, so we have considered F=0.5. In general, a higher value of C_r helps in speeding up the convergence rate; therefore in the present study, we have taken C_r=0.9. All the algorithms are executed on a PIV PC, using DEV C++, 30 times for each problem. In order to have a fair competition, we have kept the same parameter settings for all the algorithms. Random numbers are generated using the inbuilt random number generator *rand* () *function* available in DEVC++.

Overall acceleration rate AR, which is taken for the purpose of comparison, is defined as (Rahnamayan et al., 2007)

$$\mathrm{AR} = \left(1 - \frac{\sum_{j=1}^{\mu} \mathrm{NFE}\ (\text{by one algo})_j}{\sum_{j=1}^{\mu} \mathrm{NFE}\ (\text{by other algo})_j}\right) \times 100$$

where μ is number of functions.

In every case, a run was terminated when the best function value obtained was less than a threshold for the given

function or when the maximum number of function evaluation (NFE=10^6) was reached. In order to have a fair comparison, these settings are kept the same for all algorithms over all benchmark functions during the simulations.

Benchmark problems and real life problems

The performance of the proposed NSDE is evaluated on a test bed of 20 standard, benchmark problems with box constraints, taken from the literature (Rahnamayan et al., 2008). The test bed consists of a wide range of problems with dimension varying from 2 to 30. Mathematical models of the benchmark problems along with the theoretic optimum value are given in Appendix A.

The effectiveness of an algorithm can be justified, if it is able to solve the real life problems with equal ease with which it solved the test problems. Therefore, beside considering the benchmark functions, we have also taken two real life application problems, namely the transistor modelling problem and the frequency modulation sound parameter identification problem from Price (1983) and Das et al. (2009), respectively. Mathematical models of real life problems are given in Appendix B.

Numerical results and comparisons

Numerical results of benchmark problems

We have compared the proposed NSDE with the basic DE and ODE. Here we would like to mention that we have used the ODE version given in Rahnamayan et al. (2007) instead of that in Rahnamayan et al. (2008) because in Rahnamayan et al. (2008), the authors have used additional features like opposition-based generation jumping, etc., whereas in the present study, we just focusing on the effect of initial population generation on the DE algorithm. Comparisons of the algorithms are done in terms of average fitness function value, standard deviation and the corresponding *t*-test value, the average number of function evaluations, and the average time taken by every algorithm to solve a particular problem.

From Table 1, which gives the average fitness function value, standard deviation and *t*-values, it can be observed that for the 20 benchmark problems taken in the present study, all the algorithms gave more or less similar results in terms of average fitness function value, with marginal difference, which are comparable with true optimum. For the function f_{14} (Step function) all the algorithms gives same results. Table 2 shows the average CPU time taken by the algorithms, the mean number of function evaluations of 30 runs and overall acceleration rates. The best and worst fitness function values obtained, in 30 runs, by all the algorithms for benchmark problems are given in Table 3.

However, when we do the comparison in terms of average time taken and average number of function evaluations, then the proposed NSDE emerges as a clear winner. It converges to the optimum at a faster rate in comparison with all other algorithms. Only for functions f_{13} (Schwefel) did the NSDE take more NFE than the ODE, whereas for the remaining 19 problems NSDE converged faster than the basic DE and ODE. To solve 20 problems, the average NFE taken by NSDE are 1,334,200 while ODE took 1,794,738 NFE and

Table 1 Mean fitness, standard deviation of functions in 30 runs and *t*-valve

		Mean fitness (standard deviation); *t*-value		
Function	Dim.	DE	ODE	NSDE
f_1	30	**0.0546854** (0.0131867); –	0.0901626 (0.0077778); 12.48	0.0916686 (0.00721253); 13.25
f_2	30	**0.0560517** (0.0116127); –	0.0918435 (0.00565233); 14.92	0.0866163 (0.00666531); 12.29
f_3	30	0.0957513 (0.00293408); –	0.0952397 (0.00499586); 0.48	**0.0951172** (0.00405255); 0.68
f_4	10	0.0931511 (0.0145175); –	0.0874112 (0.00699322); 1.92	**0.0851945** (0.0121355); 2.26
f_5	30	0.0915561 (0.012111); –	**0.0885065** (0.00711877); 1.17	0.0916412 (0.00860403); 0.03
f_6	30	0.0942648 (0.00478545); –	0.0933845 (0.00620528); 0.60	**0.0926704** (0.00735851); 0.98
f_7	2	**4.26112e-008** (2.5783e-008); –	6.23824e-008 (2.75612e-008); 2.82	4.9999e-008 (2.95279e-008); 1.01
f_8	4	0.0620131 (0.0239495); –	**0.0528597** (0.0276657); 1.35	0.0591064 (0.0123711); 0.58
f_9	30	0.088998 (0.00880246); –	0.092875 (0.00487147); 2.08	**0.0882776** (0.0103789); 0.29
f_{10}	10	−7.91444 (3.40729); –	−9.61563 (0.024986); 2.69	−**9.62952** (0.0238362); 2.71
f_{11}	30	**0.0842833** (0.00897659); –	0.0890837 (0.00961583); 1.97	0.0901177 (0.00969009); 2.38
f_{12}	30	0.0940407 (0.00501821); –	**0.0931232** (0.00502023); 0.69	0.0951981 (0.00373364); 0.99
f_{13}	30	0.0956696 (0.00352899); –	**0.0935369** (0.00397665); 2.16	0.0955274 (0.00495933); 0.13
f_{14}	30	0.0 (0.0); –	0.0 (0.0); –	0.0 (0.0); –
f_{15}	30	**0.0730003** (0.0169434); –	0.0880257 (0.0115251); 3.95	0.0890936 (0.00986588); 4.42
f_{16}	2	0.0645903 (0.0231492); –	0.0545825 (0.0263629); 1.54	**0.0539806** (0.0226797); 1.76
f_{17}	30	0.0910662 (0.00428958); –	**0.0845474** (0.0118228); 2.79	0.0923214 (0.00514694); 1.01
f_{18}	2	4.75455e-008 (2.9688e-008); –	3.63292e-008 (3.10335e-008); 1.41	**2.657e-008** (2.657e-008); 1.49
f_{19}	5	0.067335 (0.025448); –	**0.0738969** (0.0209749); 1.07	0.0769911 (0.0160823); 1.73
f_{20}	5	−3.99239 (0.00164918); –	−**3.99398** (0.00235545); 2.98	−3.99297 (0.00184151); 1.27

DE, differential evolution; ODE, opposition-based learning DE; NSDE, non-linear simplex DE.

Table 2 Average CPU time (in seconds) taken by the algorithms, mean number of function evaluation of 30 runs and over all acceleration rates (AR)

		Average Time (s)			Mean function		
Function	Dim.	DE	ODE	NSDE	DE	ODE	NSDE
f_1	30	0.60	0.54	**0.51**	28,020	26,912	**26,220**
f_2	30	0.60	0.57	**0.55**	37,312	36,639	**35,500**
f_3	30	11.30	11.10	**10.40**	295,232	295,112	**232,860**
f_4	10	2.41	2.34	**2.13**	382,454	361,234	**265,420**
f_5	30	1.80	1.79	**1.70**	54,503	53,305	**52,240**
f_6	30	1.50	1.45	**1.41**	52,476	51,589	**49,170**
f_7	2	0.31	0.29	**0.23**	3845	3740	**3520**
f_8	4	0.10	0.11	**0.09**	7,902	7934	**6780**
f_9	30	1.20	1.11	1.11	44,034	41,455	**35,330**
f_{10}	10	3.02	2.93	**2.67**	220,356	196,871	**172,200**
f_{11}	30	2.91	2.37	**1.81**	200,924	196,617	**44,960**
f_{12}	30	0.59	0.52	**0.50**	66,154	63,760	**57,800**
f_{13}	30	1.83	**1.24**	1.42	197,069	**148,742**	155,970
f_{14}	30	0.71	0.65	**0.53**	42,423	41,578	**32,300**
f_{15}	30	0.47	0.45	**0.43**	25,903	24,236	**22,620**
f_{16}	2	0.13	0.11	**0.10**	3913	3832	**3600**
f_{17}	30	0.92	0.81	**0.76**	55,029	52,455	**47,760**
f_{18}	2	0.23	.21	**0.16**	7367	7249	**5150**
f_{19}	5	1.66	1.12	**0.61**	205,398	150,173	**57,540**
f_{20}	5	0.39	0.37	**0.31**	32,419	31,305	**27,260**
Total		32.68	30.08	27.43	1,962,733	1,794,738	**1,434,200**
AR			7.955%	16.064%		8.5592%	26.928%

DE, differential evolution; ODE, opposition-based learning DE; NSDE, non-linear simplex DE.

Table 3 Best and worst fitness function values obtained by all the algorithms

		Best; worst function values		
Function	Dim.	DE	ODE	NSDE
f_1	30	0.0533706; 0.0920816	0.0710478; 0.0980475	0.0801175; 0.0989173
f_2	30	0.0469366; 0.0852506	0.0834493; 0.0994759	0.0708503; 0.0971761
f_3	30	0.0912359; 0.099449	0.0812952; 0.0991723	0.085954; 0.0987729
f_4	10	0.0555946; 0.0973456	0.0782872; 0.0990834	0.0586798; 0.0986525
f_5	30	0.0550155; 0.0985525	0.0765341; 0.0976009	0.0730851; 0.0988916
f_6	30	0.0811647; 0.0995538	0.0799383; 0.0992613	0.0777488; 0.0979521
f_7	2	3.03242e-009; 8.24678e-008	1.9059e-008; 9.47894e-008	5.64424e-009; 8.50966e-008
f_8	4	0.0139037; 0.0974824	0.00826573; 0.0912189	0.0333435; 0.0790444
f_9	30	0.0746445; 0.0995713	0.0849655; 0.098311	0.064171; 0.0992847
f_{10}	10	−9.64801; −1.02642	−9.65114; −9.59249	−9.65368; −9.57805
f_{11}	30	0.0627431; 0.0944119	0.0636232; 0.0989899	0.0683468; 0.0994605
f_{12}	30	0.0849009; 0.0991914	0.0819181; 0.0999306	0.0887407; 0.0997806
f_{13}	30	0.0902771; 0.0996024	0.0866648; 0.0988438	0.0854635; 0.0998667
f_{14}	30	0.0; 0.0	0.0; 0.0	0.0; 0.0
f_{15}	30	0.0441712; 0.0945989	0.0593778; 0.0991945	0.067818; 0.0992816
f_{16}	2	0.0277478; 0.0969767	0.0129757; 0.0984134	0.0297125; 0.0918671
f_{17}	30	0.0844465; 0.0975161	0.0593988; 0.0997203	0.0836182; 0.0996487
f_{18}	2	2.3063e-009; 9.91416e-008	4.64862e-009; 9.55725e-008	1.11418e-008; 8.51751e-008
f_{19}	5	0.00746793; 0.099394	0.0291998; 0.0995076	0.0483536; 0.0999149
f_{20}	5	−3.99626; −3.99019	−3.99737; −3.99035	−3.99644; −3.99027

DE, differential evolution; ODE, opposition-based learning DE; NSDE, non-linear simplex DE.

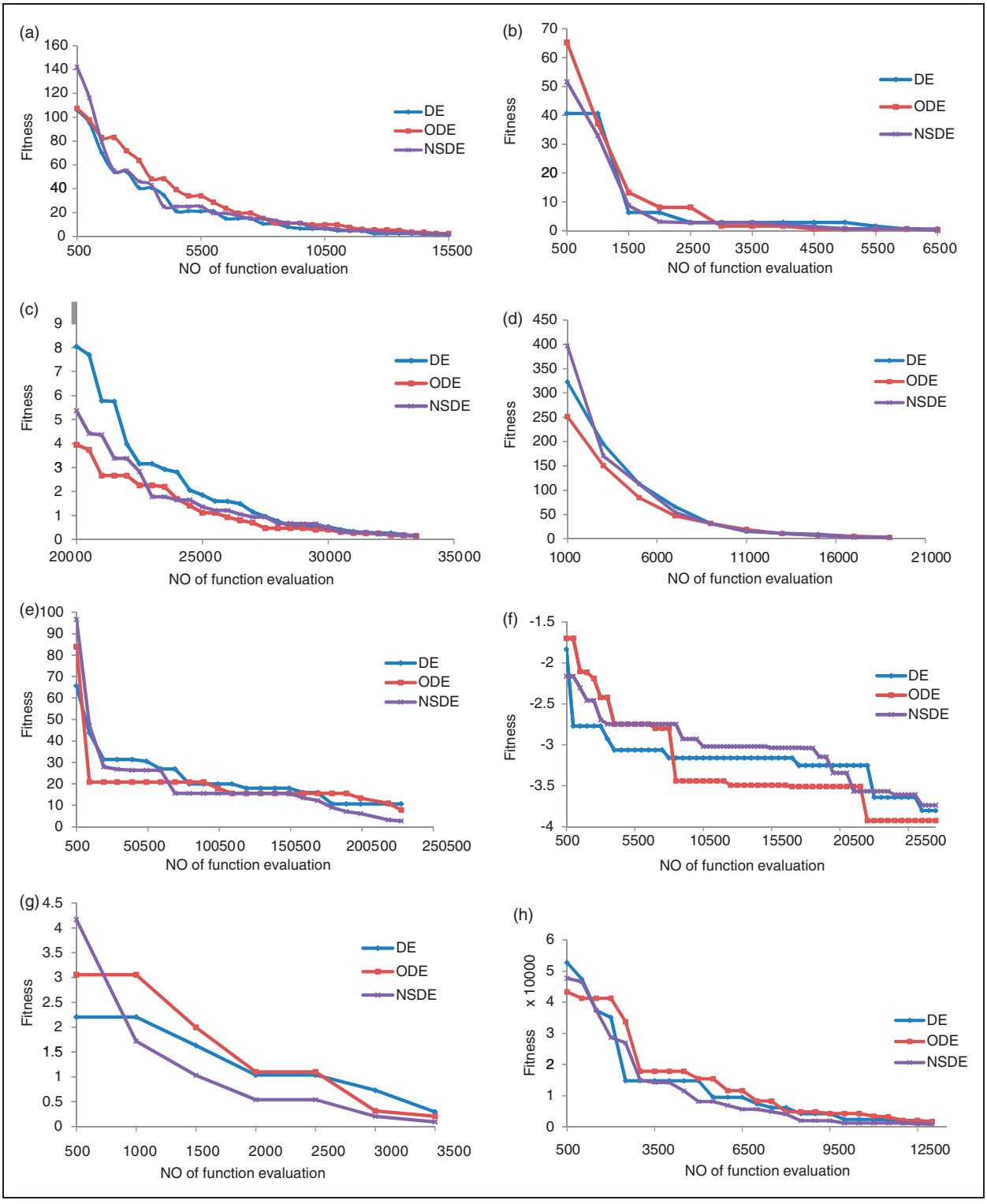

Figure 11 Performance curves of few elected benchmark problems: (a): Sphere (f_1) function; (b): Colville (f_8) function; (c): Axis parallel (f_2) function; (d): Griewenk (f_5) function; (e): Restrigin (f_4) function; (f): Inverted (f_{20}) cosine; (g): Tripod (f_{16}) function; (h): Step (f_{14}) function.

DE took 1,962,733 NFE. This implies that NSDE has an acceleration rate of around 35% in comparison with basic DE and an acceleration rate of 26% in comparison with ODE. ODE on the other hand reduces the NFE only up to 8.56%, in comparison with basic DE. A similar trend of performance can be observed from the average computational time. To solve 20 problems, NSDE took the least CPU time in comparison with the other two algorithms.

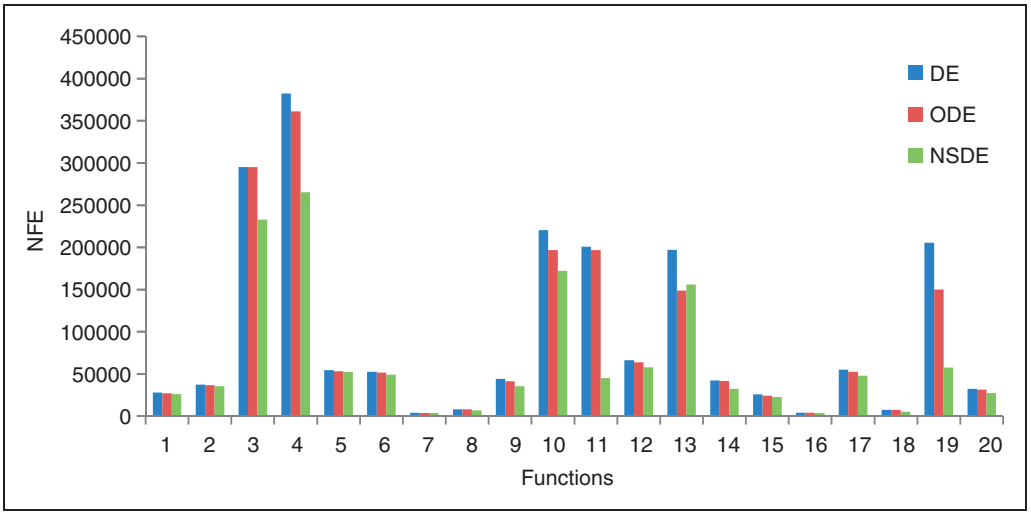

Figure 12 Average number of function evaluations taken by differential evolution (DE), opposition-based learning DE (ODE) and non-linear simplex DE (NSDE) for the 20 benchmark problems.

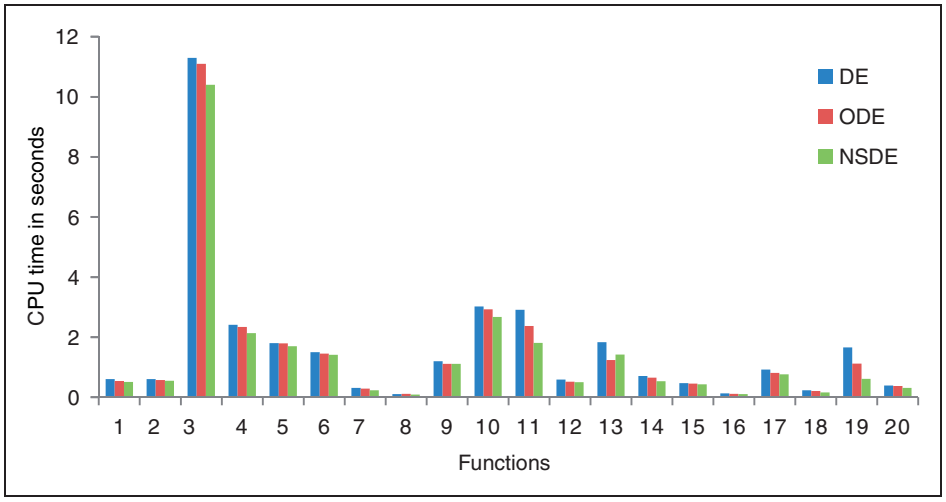

Figure 13 Average CPU time (in seconds) taken by differential evolution (DE), opposition-based learning DE (ODE) and non-linear simplex DE (NSDE) for the 20 benchmark problems.

Performance curves of selected benchmark problems are illustrated in Figure 11(a)–(h). The average number of function evaluations and the CPU time taken by the algorithms considered in the present study are illustrated in Figures 12 and 13, respectively.

Numerical results of real life problems

In order to validate further the performance of the NSDE algorithm, we used it to solve two real life problems: frequency modulation sound parameter and transistor modelling. Both the problems are unconstrained in nature and are often used to test the efficiency of the algorithm. Experimental settings for real life problems are kept the same as that of benchmark problems.

The numerical results of the real life problems in terms of average fitness function value, standard deviation, NFE and time is shown in Table 4. Once again, from this table we can observe the superior performance of the proposed NSDE algorithm. For both the problems, NSDE not only gave a better performance in terms of fitness function value but also in terms of NFE and CPU time in comparison with the basic DE algorithm.

Discussion and conclusions

The objective of the present work is to encourage the research in the direction of efficient population-generation techniques. Here we have investigated the effect of integrating the NSM and pseudo-random numbers to generate the initial population of a DE algorithm. The resulting algorithm, named NSDE, is validated on a set of 20 benchmark problems and two real life problems. Computational results strongly suggest that the proposed NSDE effectively

Table 4 Numerical results of real life problem obtained by DE and NSDE algorithms

	DE	NSDE	NFE	Time
Solutions of frequency modulation sound parameters.				
x_1	1.00008	1.00015	DE=59178	DE=16.50
x_2	4.99993	4.99998	NSDE=29678	NSDE=8.09
x_3	1.49979	1.50006		
x_4	4.79993	4.79999		
x_5	−2.00031	−2.00004		
x_6	−4.90005	−4.90003		
$f(X)$	4.7008e-006	2.34107e-006		
Solutions of transistor modelling problem				
x_1	0.901341	0.901336	DE=780768	DE=17.3
x_2	0.891174	0.891053	NSDE=461465	NSDE=11.03
x_3	3.87757	3.87933		
x_4	3.94643	3.94662		
x_5	5.32623	5.32509		
x_6	10.6239	10.6162		
x_7	0.0	0.0		
x_8	1.08914	1.08881		
x_9	0.705575	0.706727		
$f(X)$	0.0543713	0.0543658		

DE, differential evolution; NSDE, non-linear simplex DE; NFE, number of function evaluations.

combines the advantages of local search (NSM) and mechanisms of meta-heuristic optimization (DE). The use of NSM in conjugation with pseudo-random numbers to generate the initial population helps in providing DE with the information of the search space and also maintains the diversity, which in turn helps in keeping a balance between the exploratory and exploitation capabilities of basic DE. From the numerical results and graphic illustrations, we can see that the proposed schemes improve the working of basic DE in terms of average CPU time and NFEs, without compromising the quality of solution. The crux of the present research is that even without any extra or new parameter, an improvement can be made in the working of an algorithm simply by having an efficient technique to generate the initial population to provide some additional information to the search algorithm.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

Ali MM (2007) Differential evolution with preferential crossover. *European Journal of Operational Research* 181: 1137–47.

Ali M, Pant M and Singh VP (2009) A modified differential evolution algorithm with Cauchy mutation for global optimization. In *Proceedings of International Conference on Contemporary Computing (IC3, 2009)*. Springer, 127–37.

Bergey PK and Ragsdale C (2005) Modified differential evolution: a greedy random strategy for genetic recombination. *Omega* 33: 255–65.

Das S, Abraham A and Konar A (2008) Adaptive clustering using improved differential evolution algorithm. *IEEE Transactions on Systems, Man, and Cybernetics* 38(1): 218–37.

Das S, Abraham A, Chakraborty UK and Konar A (2009) Differential evolution using a neighborhood based mutation operator. *IEEE Transactions on Evolutionary Computation* 13(2): 526–53.

Das S and Konar A (2006) Design of two dimensional IIR filters with modern search heuristics: a comparative study. *International Journal of Computational Intelligence and Applications* 6(3): 329–355.

Joshi R and Sanderson AC (1999) Minimal representation multisensor fusion using differential evolution. *IEEE Transactions on Systems, Man, and Cybernetics* 29(1): 63–76.

Kaleo P and Ali MM (2006) A numerical study of some modified differential evolution algorithms. *European Journal of Operational Research* 169: 1176–84.

Li Z, Feng Y, Tan J and Wei Z (2008) A methodology to support product platform optimization using multi-objective evolutionary algorithms. *Transactions of the Institute of Measurement and Control* 30(3–4): 295–312.

Lu JC and Wang FS (2001) Optimization of low pressure chemical vapor deposition reactors using hybrid differential evolution. *Canadian Journal of Chemical Engineering* 79: 246–54.

Nelder JA and Mead R (1965) A simplex method for function minimization. *Computer Journal* 7: 308–13.

Olsson DM and Nelson LS (1975) The Nelder–Mead simplex procedure for function minimization. *Technometrics* 17: 45–51.

Omran M, Engelbrecht AP and Salman A (2005) Differential evolution methods for unsupervised image classification. *Proceedings of the 7th Congress on Evolutionary Computation (CEC-2005)* 966–73.

Pant M, Ali M and Singh VP (2008) Differential evolution with parent centric crossover. *Second UKSIM European Symposium on Computer Modelling and Simulation Proceeding in IEEE* 141–6.

Pant M, Ali M and Abraham A (2009a) Mixed strategy embedded differential evolution. *IEEE Congress on Evolutionary Computation*. Norway, 1240–6.

Pant M, Ali M and Singh VP (2009b) Parent centric differential evolution algorithm for global optimization. *Opsearch* 46(2): 153–68.

Pant M, Thangaraj R, Abraham A and Grosan C (2009c) Differential evolution with Laplace mutation operator. *IEEE Congress on Evolutionary Computation*. Norway, 2841–9.

Parsopoulos KE and Vrahatis MN (2002) initializing the particle swarm optimization using nonlinear simplex method. *Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*. WSEAS Press, 216–21.

Price WL (1983) global optimization by controlled random search. *Journal of Optimization Theory and Applications* 40(3): 333–48.

Rahnamayan S, Tizhoosh HR and Salama MMA (2007) A novel population initialization method for accelerating evolutionary algorithms. *Computer and Applied Mathematics with Application* 53: 1605–14.

Rahnamayan S, Tizhoosh HR and Salama MMA (2008) Opposition based-differential evolution. *IEEE Transactions on Evolutionary Computation* 12(1): 64–79.

Rogalsky T, Derksen RW and Kocabiyik S (1999) Differential evolution in aerodynamic optimization. *Proceedings of the 46th Annual Conference of the Canadian Aeronautics and Space Institute* 29–36.

Storn R and Price K (1995) DE-a simple and efficient adaptive scheme for global optimization over continuous space. Technical Report TR-95-012, ICSI, March 1995. Available at: ftp.icsi.berkeley.edu/pub/techreports/1995/tr-95-012ps.Z.

Storn R and Price K (1997) DE-a simple and efficient heuristic for global optimization over continuous space. *Journal of Global Optimization* 11(4): 41–359.

Tang WJ and Wu QH (2009) Biologically inspired optimization: a review. *Transactions of the Institute of Measurement and Control* 31(6): 495–515.

Wang F-S and Jang H-J (2000) Parameter estimation of a bio-reaction model by hybrid differential evolution. *Proceedings IEEE Congress on Evolutionary Computation* 410–17.

Wang CX, Cui DW, Wan DS and Wang L (2006) A novel genetic algorithm based on gene therapy theory. *Transactions of the Institute of Measurement and Control* 28(3): 253–62.

Zhu Y, He X, Hu K and Niu B (2009) Information entropy based interaction model and optimization method for swarm intelligence. *Transactions of the Institute of Measurement and Control* 31(6): 461–74.

Appendix A: Test functions

1. Sphere function:

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

with $-5.12 \le x_i \le 5.12$, $\min f_1(0, \ldots, 0) = 0$

2. Axis parallel hyper-ellipsoid:

$$f_2(x) = \sum_{i=1}^{n} i x_i^2$$

with $-5.12 \le x_i \le 5.12$, $\min f_2(0, \ldots, 0) = 0$

3. Rosenbrock's valley:

$$f_3(x) = \sum_{i=1}^{n-1} [100(x_{i+1}^2 - x_i^2) + (1 - x_i^2)]$$

with $-2 \le x_i \le 2$, $\min f_3(1, \ldots, 1) = 0$

4. Restrigin's function:

$$f_4(x) = 10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i))$$

with $-5.12 \le x_i \le 5.12$, $\min f_4(0, \ldots, 0) = 0$

5. Griewenk function:

$$f_5(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$$

with $-600 \le x_i \le 600$, $\min f_5(0, \ldots, 0) = 0$

6. Ackley's function:

$$f_6(X) = -20^{*}\exp\left(-.2\sqrt{1/\mathrm{n}\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{x}_\mathrm{i}^2}\right) - \exp\left(1/\mathrm{n}\sum_{\mathrm{i}=1}^{\mathrm{n}} \cos(2\pi \mathrm{x}_\mathrm{i})\right) + 20 + e\,,$$

with $-32 \le x_i \le 32$, $\min f_6(0, \ldots, 0) = 0$

7. Beale function:

$$f_7(x) = [1.5 - x_1(1 - x_2)]^2 + [2.25 - x_1(1 - x_2^2)]^2 + [2.625 - x_1(1 - x_2^3)]^2$$

with $-4.5 \le x_i \le 4.5$, $\min f_7(3, 0.5) = 0$

8. Colville function:

$$f_8(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) 19.8(x_2 - 1)(x_4 - 1)$$

with $-10 \le x_i \le 10$, $\min f_8(1, 1, 1, 1) = 0$

9. Levy function:

$$f_9(x) = \sin^2(3\pi x_1) + \sum_{i=1}^{n-1} (x_i - 1)(1 + \sin^2(3\pi x_{i+1})) + (x_n - 1)(1 + \sin^2(2\pi x_n))$$

with $-10 \le x_i \le 10$, $\min f_9(1, \ldots, 1) = 0$

10. Michalewicz function:

$$f_{10}(x) = -\sum_{i=1}^{n} \sin(x_i)(\sin(ix_i^2/\pi))^{2m}$$

with $0 \le x_i \le \pi$, m=10, $\min f_{10(n=10)} = -9.66015$

11. Zakharov function:

$$f_{11}(x) = \sum_{i=1}^{n} x_i^2 + (\sum_{i=1}^{n} 0.5ix_i)^2 + (\sum_{i=1}^{n} 0.5ix_i)^4$$

with $-5 \le x_i \le 10$, $\min f_{11}(0, \ldots, 0) = 0$

12. Schawefel's problem 2.22:

$$f_{12}(x) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|$$

with $-10 \le x_i \le 10$, $\min f_{12}(0, \ldots, 0) = 0$

13. Schwefel's problem 2.21:

$$f_{13}(x) = \max_i \{|x_i|, 1 \le i \le n\}$$

with $-100 \le x_i \le 100$, $\min f_{13}(0, \ldots, 0) = 0$

14. Step function:

$$f_{14}(x) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2$$

with $-100 \le x_i \le 100$, $\min f_{14}(-0.5 \le x_i \le 0.5) = 0$

15. Quartic function:

$$f_{15}(x) = \sum_{i=1}^{n} i x_i^4 + random[0, 1)$$

with $-1.28 \le x_i \le 1.28$, $\min f_{15}(0, \ldots, 0) = 0$

16. Tripod function:

$$f_{16}(x) = p(x_2)(1 + p(x_1)) + |(x_1 + 50p(x_2)(1 - 2p(x_1)))| + |(x_2 + 50(1 - 2p(x_2)))|$$

with $-100 \le x_i \le 100$, $\min f_{16}(0, -50) = 0$
where $p(x) = 1$ for $x > 0$ otherwise $p(x) = 0$

17. Alpine function:

$$f_{17}(x) = \sum_{i=1}^{n} |x_i \sin(x_i) + 0.1x_i|$$

with $-10 \le x_i \le 10$, $\min f_{17}(0, \ldots, 0) = 0$

18. Cshaffer's function 6:

$$f_{18}(x) = 0.5 + \frac{\sin^2 \sqrt{(x_1^2 + x_2^2)} - 0.5}{1 + 0.01(x_1^2 + x_2^2)^2}$$

with $-10 \le x_i \le 10$, $\min f_{18}(0, 0) = 0$

19. Pathological function:

$$f_{19}(x) = \sum_{i=1}^{n-1} \left(0.5 + \frac{\sin^2 \sqrt{(100x_i^2 + x_{i+1}^2)} - 0.5}{1 + 0.001(x_i^2 + x_{i+1}^2 - 2x_i x_{i+1})^2} \right)$$

with $-100 \le x_i \le 100$, $\min f_{19}(0, \ldots, 0) = 0$

20. Inverted cosine wave function:

$$f_{20}(x) = -\sum_{i=1}^{n-1} \left(\exp\left(\frac{-(x_i^2 + x_{i+1}^2 + 0.5x_i x_{i+1})}{8} \right) \times \cos\left(4\sqrt{x_i^2 + x_{i+1}^2 + 0.5x_i x_{i+1}} \right) \right)$$

with $-5 \le x_i \le 5$, $\min f_{20}(0, \ldots, 0) = -n + 1$

Appendix B: Real life problems

B1 Frequency modulation sound parameter identification (Das et al., 2009)

An interesting application of global optimization algorithms is to optimize the parameters of the frequency modulated (FM) synthesizer. FM sound synthesis plays an important role in several modern music systems. The DE algorithm initializes a set of parameters and the FM synthesizer generates the corresponding sounds. In the feature extraction step, the dissimilarities of features between the target sound and synthesized sound are used to compute the fitness value. The process continues until synthesized sounds become very similar to the target.

The problem is to satisfy six parameters a_1, w_1, a_2, w_2, a_3, w_3 of the frequency modulation sound model is given below.

$$\begin{aligned} y(t) = a_1 \times \sin(w_1 \times t \times \theta + a_2 \\ \times \sin(w_2 \times t \times \theta + a_3 \\ \times \sin(w_3 \times t \times \theta))) \end{aligned}$$

with $\theta = (2.\pi/100)$. The fitness function is defined as the sum of square error between the evolved data and the model data as follows:

$$f(a_1, w_1, a_2, w_2, a_3, w_3) = \sum_{t=0}^{100} (y(t) - y_0(t))^2$$

where the model data are given by the following equation.

$$\begin{aligned} y_0(t) = 1.0 \times \sin(5.0 \times t \times \theta + 1.5 \\ \times \sin(4.8 \times t \times \theta + 2.0 \\ \times \sin(4.9 \times t \times \theta))) \end{aligned}$$

Each parameter is in the range $(-6.4, 6.35)$. This is highly complex multimodal problem with minimum value zero.

B2 Transistor modelling (Price 1983)

The mathematical model of the transistor design is given by,

$$\text{Minimize } f(x) = \gamma^2 + \sum_{k=1}^{4} (\alpha_k^2 + \beta_k^2)$$

where

$$\begin{aligned} \alpha_k = (1 - x_1 x_2) x_3 \{ \exp[x_5 (g_{1k} - g_{3k} x_7 \times 10^{-3} - g_{5k} x_8 \\ \times 10^{-3})] - 1 \} g_{5k} + g_{4k} x_2 \end{aligned}$$

$$\beta_k = (1 - x_1 x_2) x_4 \{\exp[x_6(g_{1k} - g_{2k} - g_{3k} x_7 \times 10^{-3} + g_{4k} x_9 \times 10^{-3})] - 1\} g_{5k} x_1 + g_{4k}.$$

$$\gamma = x_1 x_3 - x_2 x_4$$

subject to:

$$x_i \geqslant 0$$

and the numerical constants g_{ik} are given by the matrix

$$\begin{bmatrix} 0.485 & 0.752 & 0.869 & 0.982 \\ 0.369 & 1.254 & 0.703 & 1.455 \\ 5.2095 & 10.0677 & 22.9274 & 20.2153 \\ 23.3037 & 101.779 & 111.461 & 191.267 \\ 28.5132 & 111.8467 & 134.3884 & 211.4823 \end{bmatrix}$$

This objective function provides a least-sum-of-squares approach to the solution of a set of nine simultaneous non-linear equations, which arise in the context of transistor modelling.