Annealed Differential Evolution

Swagatam Das[1], Amit Konar[1] and Uday K. Chakraborty[2]

[1] Dept. of Electronics and Telecommunication Engg, Jadavpur University
Kolkata 700032, India
swagatamdas19@yahoo.co.in, konaramit@yahoo.co.in
[2]Dept. of Math & Comp. Sc., University of Missouri,
St. Louis, MO 63121, USA
uday@cs.umsl.edu

***Abstract*—Differential Evolution (DE) has recently emerged as a leading methodology for global search and optimization over continuous, high-dimensional spaces. It has been successfully applied to a wide variety of nearly intractable engineering problems. However, the DE and its variants usually employ a deterministic selection mechanism that always allows the better solution to survive to the next generation. This often prevents DE from escaping local optima at the early stages of search over a multi-modal fitness landscape and leads to a premature convergence. The present work proposes to improve the accuracy and convergence speed of DE by introducing a stochastic selection mechanism. The idea of a conditional acceptance function (that allows accepting inferior solutions with a gradually decaying probability) is borrowed from the realm of the Simulated Annealing (SA). In addition, the work proposes a *center of mass* based mutation operator and a decreasing crossover rate in DE. Performance of the resulting hybrid algorithm has been compared with three state-of-the-art adaptive DE schemes. The method is shown to be statistically significantly better on a six-function test-bed and one difficult engineering optimization problem with respect to the following performance measures: solution quality, time to find the solution, frequency of finding the solution, and scalability.**

***Keywords* – Differential Evolution, Simulated Annealing, Radar poly-phase code design, Hill-climbing.**

1. Introduction

Differential Evolution (DE), proposed by Storn and Price [1], is a simple yet powerful population-based stochastic search technique over the contiouous search space. DE has been sucessfully applied in diverse domains of science and engineering problems, such as mechanical engineering design [2, 3], signal processing [4], chemical engineering [5], machine intelligence and pattern recognition [6]. In many cases, it has reportedly outperformed the GA or the particle swarm optimization (PSO) [7].

Like other evolutionary algorithms, two fundamental processes drive the evolution of a DE population: the *variation* process, which enables exploring the different regions of the search space and the *selection* process, which ensures the exploitation of the previous knowledge about the fitness landscape.

Practical experiences, however, suggest that DE may occasionally stop proceeding towards the global optimum for no apparent reason, while the population has not converged to a local optimum or any other point [8]. Occasionally, even new individuals may enter the population, but the algorithm does not progress by finding any better solutions. This situation is usually referred to as *stagnation*. DE also suffers from the problem of premature convergence where the population converges to some local optima of a multi-modal objective function, loosing its diversity. The probability of stagnation depends on how many different potential trial solutions are available and also their capability to enter into the population of the subsequent generations [8].

There exists a good volume of works (a very brief review of which can be found in section 2), seeking to improve the convergence rate of DE by adapting the parameters like population size NP, the scale factor F and the crossover rate Cr. However, not much has been reported to improvise the basic selection mechanism of DE with a view to avoiding the above mentioned problems.

In this paper the selection mechanism of the classical DE has been modified by employing the concepts of the *Simulated Annealing* (SA) algorithm. SA belongs to a class of heuristic search algorithms called *probabilistic hill-climbing* [9], which dynamically alter the probability of accepting the inferior solutions. The hybrid algorithm presented here is referred to as the Annealed Differential Evolution (AnDE). Besides using an SA-based selection scheme, the AnDE introduces a center of mass based mutation strategy, in which the trial vectors are stochastically attracted towards the mean vector of the current population, instead of the best one as done in the case of DE/current-to-best/bin and DE/best/2 schemes [1, 10]. In addition, the use of a time varying cross-over rate is seen to improve the performance of AnDE. Empirical simulations with well-known benchmarks show the usefulness of these modifications.

The remainder of this paper is organized as follows. Section 2 provides a brief outline of the classical DE family of algorithms and also reviews previous work for the improvement of the performance of DE. Section 3 briefly discusses the spirit of the SA algorithm and then introduces the proposed hybrid algorithm. Experimental settings for the benchmarks and simulation strategies are explained in Section 4. Results are presented in Section 5, and finally the conclusions are drawn in Section 6.

1-4244-1340-0/07/$25.00 ©2007 IEEE

2. The DE and its Variants – An Overview

DE starts with a population of *NP* D-dimensional search variable vectors or *chromosomes* in the terminology of evolutionary computing. The i-th vector of the population at time step (iteration) t = t may be represented as:

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

In each iteration of the algorithm, for each population member $\vec{X}_i(t)$, a *donor* or *trial* vector $\vec{V}_i(t+1)$ is created. It is the method of creating this donor vector that distiguishes various DE schemes from one another. In "scheme DE1" (DE/rand/1), to create $\vec{V}_i(t+1)$, three vectors (say r_1, r_2, and r_3) are randomly chosen from the current population. Next the difference of any two of these three vectors is scaled by a scalar F and the scaled difference is added to the third vector whence we obtain the donor vector:

$$v_{i,j}(t+1) = x_{r1,j}(t) + F.(x_{r2,j}(t) - x_{r3,j}(t)) \quad (2)$$

The DE family of algorithms use two kinds of crossover, namely 'exponential' and 'binary'. In 'exponential' crossover, we first choose an integer n randomly from [0, D-1]. We choose another integer L from the interval [1, D]. An offspring vector:

$$\vec{U}_i(t+1) = [u_{i,1}(t+1), u_{i,2}(t+1), u_{i,3}(t+1), ..., u_{i,D}(t+1)] \quad (3)$$

is then formed:

$$\left.\begin{array}{l} u_{i,j}(t+1) = V_{i,j}(t+1) \quad \text{for } j = <n>_D, <n+1>_D, ... <n-L+1>_D \\ \quad\quad = X_{i,j}(t+1) \quad \text{for all other } j \in [0, D-1] \end{array}\right\} \quad (4)$$

where the angular brackets $< >_D$ denote a modulo function with modulus D. The integer L is drawn from [1, D] according to the following policy:

L = 0;
Do L=L+1;
while (rand (0, 1) < Cr) AND (L<D));

Thus in effect, Probability (L>m) = $(Cr)^{m-1}$ for any m>0. Cr, the crossover constant, is a control parameter of DE. For each donor vector $\vec{V}_i$, a new set of n and L must be chosen randomly as shown above.

'Binary' crossover is implemented as follows:

$$u_{i,j}(t+1) = v_{i,j}(t+1) \quad \text{if rand}(0, 1) < \text{Cr}$$
$$= x_{i,j}(t) \quad \text{otherwise} \quad (5)$$

DE actually involves the Darwinian principle of "Survival of the fittest" in its selection process, which may be outlined as,

$$\vec{X}_i(t+1) = \vec{U}_i(t+1) \quad \text{if } f(\vec{U}_i(t+1)) < f(\vec{X}_i(t))$$
$$= \vec{X}_i(t) \quad \text{if } f(\vec{X}_i(t)) < f(\vec{U}_i(t+1)) \quad (6)$$

where f( ) is the function to be minimized. So if the new offspring yields a better value of the fitness function, it replaces its parent in the next generation; otherwise the parent is retained in the population. DE/current to best/1 follows the same procedure except that the donor vector is created using two randomly selected members of the population as well as the best vector of the current time step:

$$\vec{V}_i(t+1) = \vec{X}_i(t) + \lambda.(\vec{X}_{best}(t) - \vec{X}_i(t)) + \beta.(\vec{X}_{r2}(t) - \vec{X}_{r3}(t)) \quad (7)$$

where is another control parameter of DE in [0, 2]. To reduce the number of control parameters a usual choice is to put = F. Storn and Price [11] suggested a total of ten different working strategies for DE.

Researchers have suggested some guidelines for choosing the donor vector generation strategy and control parameter values for DE due to their high influences on the ultimate performance. Storn and Price [11] indicated that a reasonable value for *NP* could be chosen between 5D and 10D, and a good initial choice of *F* was 0.5. The effective value range of *F* is usually between 0.4 and 1. A good first choice for *Cr* is 0.1, but since a large *Cr* value can speed convergence, the value of 0.9 for *Cr* may also a good initial trial to see if a quick solution is possible. Moreover, if the population converges prematurely, either *F* or *NP* should be increased. Many interesting results on the parameter adaptation of both single and multi-objective DE can be traced in works reported in [12-19]. We cannot review each of them in details here due to space limitations.

Recently, Brest *et al*. [20] encoded control parameters *F* and *Cr* into the individual and adjusted by introducing two new parameters τ_1 and τ_2. In their algorithm (called Self Adaptive DE or SADE), a set of *F* values was assigned to each individual in the population, respectively. Then, a random number *rand* is uniformly generated in the range of [0, 1]. If rand < τ_1 the *F* was reinitialized to a new random value in the range of [0.1, 1.0], otherwise it kept unchanged. The *Cr* was adapted in the same manner but with a different re-initialization range of [0, 1].

Yan *et al*. introduced simulated annealing based DE scheme (SADE) [21] which outperformed one classical DE algorithm over a few test functions. The hybrid algorithm proposed here differs considerably from their work with respect to the selection mechanism, cooling schedule and the DE mutation strategy.

3. The Annealed DE Algorithm

Simulated Annealing (SA) exploits an analogy between the way in which a metal cools and freezes into a minimum-energy crystalline structure (the annealing process) and the search for a minimum in a more general system. The algorithm is based upon that of Metropolis *et al*. [22], which was originally proposed as a means of finding the equilibrium configuration of a collection of atoms at a given temperature. It was Kirkpatrick *et al*. [23] who proposed it as the basis of an optimization technique for combinatorial (and other) problems.

The major advantage of SA over the other heuristic search methods lies in the mechanism of avoidance of local minima. The algorithm employs a random search which not only accepts better solutions that decrease the objective function f (in case of a minimization problem), but also some inferior solutions that increase it. The latter are accepted with a probability:

$$P = \exp(-\frac{\Delta f}{T}) \quad (8)$$

where Δf is the change of the objective function over two successive iterations and T is a control parameter, which by analogy with the original application is known as the system `temperature'.

Unlike the *greedy* selection strategy employed in the classical DE, the AnDE algorithm proposed here incorporates a typical SA-type selection mechanism that conditionally accepts the inferior solutions to the next generation. Suppose at time step t, the i-th chromosome is $\vec{X}_i(t)$ and its offspring, created through the DE-type mutation and crossover operations at the next time step, is $U_i(t+1)$. Now if $f(\vec{U}_i(t+1)) < f(\vec{X}_i(t))$, i.e. if the offspring is better than the parent w.r.t the objective function, then $\vec{X}_i(t)$ is surely replaced in the next generation with $U_i(t+1)$. But even if $f(\vec{U}_i(t+1)) > f(\vec{X}_i(t))$, $U_i(t+1)$ may replace the parent chromosome $\vec{X}_i(t)$ with a probability of:

$$P_i = \exp[-(\frac{f(\vec{U}_i(t+1) - f(\vec{X}_i(t)}{T})] \quad (9)$$

The SA algorithm requires an annealing schedule for decreasing the control temperature T from an intial value T_0 to a final value T_f. This allows it to escape the local optima at the early stages of the search and to efficiently hill-climb as the temperature approaches zero. We should note that as T • 0, P_i also approaches zero. Several annealing schedules have been proposed and analysed in the SA literature [24]. The AnDE algorithm employs a very common temperature decrement rule known as the exponential cooling schedule (ECS), proposed by Kirkpatrick *et al.* [23]. According to this rule, the temperatures T(t) and T(t+1) over two successive time steps are related as:

$$T(t+1) = \alpha.T(t) \quad (10)$$

where is constant close to, but smaller than, 1.

The AnDE also modifies the basic mutation scheme of the DE/best/1 given by (7) by replacing the best vector $\vec{X}_{best}(t)$ with the mean of all the vectors belonging to the current generation. This mean vector can be regarded as the center of mass of the DE population (especially if we assume that the chromosomes are agents of equal mass in the multi-dimensional search space) and is given by:

$$\vec{X}_{CM} = \frac{1}{NP}\sum_{i=1}^{NP}\vec{X}_i(t) \quad (11)$$

where NP is the total number of search variable vectors in the population. With this modification, the mutation scheme for trial vector generation in AnDE becomes:

$$\vec{V}_i(t+1) = \vec{X}_i(t) + F.(\vec{X}_{CM}(t) - \vec{X}_i(t)) + F.(\vec{X}_{r2}(t) - \vec{X}_{r3}(t)) \quad (12)$$

The CM-based mutation scheme maintains cohesion in the population of vectors and facilitates the quick convergence of most of the vectors to some near-optimal regions of the fitness landscape. In addition, AnDE decreases the crossover rate Cr linearly with time from Cr_{max} = 1.0 to Cr_{min} = 0.5. If Cr = 1.0, it means that all components of the parent vector are replaced by the difference vector operator according to (12). But at the later stages of the optimizing process, if Cr is decreased, more components of the parent vector are inherited by the offspring. Such a tuning of Cr helps to explore the search space vigorously at the beginning, but adjust the movements of trial solutions finely during the later stages of search, so that they can explore the interior of a relatively small space in which the suspected global optimum lies. The time-variation of Cr may be expressed in the form of the following equation:

$$Cr = (Cr_{max} - Cr_{min}).\frac{(T_{max} - t)}{t} \quad (13)$$

where Cr_{max} and Cr_{min} are the maximum and minimum values of Crossover rate Cr, t is the current time step and T_{max} is the maximum time step (i.e. the maximum number of iterations allowed). The algorithm can be put forward in pseudo code as follows:

The AnDE Algorithm

Input: Randomly initialized search variable vectors $\vec{X}_i(0)$

Output: Position of the approximate global optima $\vec{X}*$

Begin

Initialize population of NP vectors uniformly in the search range;

While (current time step t < T_{max}) **do**

Begin

for i = 1 to *NP*,

Create Difference-Offspring $\vec{U}_i(t+1)$ from $\vec{X}_i(t)$ using mutation and crossover opeartors described in (12) and (5);

Evaluate objective functions $f(\vec{X}_i(t))$ and $f(\vec{U}_i(t+1))$;

if $f(\vec{U}_i(t+1)) \le f(\vec{X}_i(t))$

then $\vec{X}_i(t+1) = \vec{U}_i(t+1)$

else if $f(\vec{U}_i(t+1)) > f(\vec{X}_i(t))$

then if

rand(0, 1) < $\exp[-(\frac{f(\vec{U}_i(t+1) - f(\vec{X}_i(t)}{T(t)})]$

$\vec{X}_i(t+1) = \vec{U}_i(t+1)$;

else $\vec{X}_i(t+1) = \vec{X}_i(t)$;

```
        end if;
      end if;
    end for;
    Change the control temperature according to (10);
  end While;
end.
```

4. The Experimental Setup

4.1 Benchmark Functions Used

The performance of the AnDE algorithm has been evaluated on a test suite of seven well-known benchmarks (Table 1) [25]. In Table 1, *n* represents the number of dimensions (we used *n* = 25, 50, 75 and 100). The first two test functions are unimodal, having only one minimum. The others are multimodal, with a considerable number of local minima in the region of interest. All benchmark functions except f_6 have the global minimum at the origin or very near to the origin [25]. For Shekel's foxholes (f_6), the global minimum is at (-31.95, -31.95) and f_6 (-31.95, -31.95) • 0.998, and the function has only two dimensions. Table 2 summarizes the initialization and search ranges used for these functions. An asymmetrical initialization procedure has been used here following the work reported in [26].

4.2.The Spread Spectrum Radar Polyphase Code Design Problem

A famous problem of optimal design arises in the field of spread spectrum radar poly-phase codes [8]. Such a problem suits very well for the application of global optimization algorithms like DE. The problem can be formally stated as:

$$\text{global min } f(x) = \max_{x \in X}\{\varphi_1(x),....,\varphi_{2m}(x)\}$$

$$X = \{(x_1,....,x_n) \in R^n \mid 0 \le x_j \le 2\pi, j = 1,...,n\} \quad (17)$$

where m = 2n – 1 and

$$\varphi_{2i-1}(x) = \sum_{j=i}^{n} \cos\Big(\sum_{k=|2i-j-1|-1}^{j} x_k\Big), \quad i = 1, ..., n$$

$$\varphi_{2i}(x) = 0.5 + \sum_{j=i+1}^{n} \cos\Big(\sum_{k=|2i-j-1|-1}^{j} x_k\Big), \quad i=1,.....,n-1$$

$$\varphi_{m+i}(x) = -\varphi_i(x), \quad i=1,.....,m \quad (18)$$

The work in [27] shows that the above problem is NP-hard.

4.3 Simulation Strategy

Simulations were carried out to obtain a comparative performance analysis of the proposed method with respect to: (a) DE/rand/1/bin (b) DE/best/1/bin and (c) Self Adaptive DE (SADE). The first two classical DE schemes were chosen because of their wide popularity in solving numerical optimization or engineering problems.

Table 1. Benchmark Functions Used

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

Table 2. Search Ranges of Benchmark Functions

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

For all the four algorithms we used the same population size, which is 10 times the dimension of the problem. To make the comparison fair, the populations for all the competitor algorithms (for all problems tested) were initialized using the same random seeds.

We choose the number of fitness evaluations (FE) as a measure of computation time instead of 'generations' or 'iterations'.

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

For the DE/rand/1/bin and DE/best/1/bin, the cross-over rate *Cr* = 0.9 and the scale factor *F* = = 0.8 were used. The initial temperature T_0 in case of the AnDE

algorithm was set as 100 times the worst solution in the first generation [24]. For all DE versions the population size *NP* equals 10 times the number of dimensions in the function. All the algorithms discussed here have been developed from scratch in Visual C++ on a Pentium IV, 2.3 GHz PC, with 1024 KB cache and 2 GB of main memory in Windows XP environment.

5. Results

The following performance measures are used for our comparative study: (a) quality of the final solution, (b) speed of convergence towards the optimal solution, (c) success rate (frequency of hitting the optimum), and (d) scalability of the algorithms against the growth of problem dimensions. Table 3 compares the algorithms on the quality of the optimum solution. The mean and the standard deviation (within parentheses) of the best-of-run values for 30 independent runs of each of the four algorithms are presented. Each algorithm was run up to a predetermined maximum number of FEs (depending upon the complexity of the problem). Missing values of standard deviation indicate a zero standard deviation. The best solution in each case has been shown in bold.

Table 4 shows results of unpaired *t*-tests between AnDE and the best of the three competing algorithms in each case (standard error of difference of the two means, 95% confidence interval of this difference, the *t* value, and the two-tailed *P* value). For all cases in Table 4, sample size = 30 and degrees of freedom = 58. It is interesting to see from Tables 3 and 4 that the proposed method meets or beats the nearest competitor in a statistically significant way.

Table 5 shows, for all test functions and all algorithms, the number of runs (out of 30) that managed to find the optimum solution within a given tolerance or cutoff value. In Table 6 we report the mean number of function evaluations (FEs) and standard deviations (within parentheses) required by each competitor algorithm to converge within the prescribed cut-off value. In each case, the mean is calculated over the corresponding number of runs that managed to converge within the cut-off value. The entries marked NA (Not Applicable) in this table, imply that no run of the corresponding algorithm converged to the cut-off within the maximum number of FEs allowed. Missing values of standard deviation here also indicate a zero standard deviation.

In Figure 1 we have graphically presented the rate of convergence of all the methods for all the functions (in 100 dimensions). Figure 2 shows the scalability of the four methods on six test functions - how the average computational cost (measured in number of function evaluations) to find the solution varies with an increase in the dimensionality of the search space. These results show that the proposed method leads to significant improvements in most cases.

Table 3. Average and the standard deviation of the best-of-run solution for 30 runs tested on seven benchmark functions (for all cases except f_6, the cut-off value used is 1.00e-05, while for f_6 it is 0.998).

Fun	D	Max^m FE	Mean Best Value (Standard Deviation)			
			DE/rand/1/bin	DE/best/1/bin	SADE	AnDE
f_1	25	50,000	**1.00e-05**	**1.00e-05**	**1.00e-05**	**1.00e-05**
	50	1×10^5	**1.00e-05**	**1.00e-05**	**1.00e-05**	**1.00e-05**
	75	5×10^5	**1.00e-05**	**1.00e-05**	**1.00e-05**	**1.00e-05**
	100	1×10^6	**1.00e-05**	**1.00e-05**	**1.00e-05**	**1.00e-05**
f_2	25	50,000	1.6342e-01 (4.253e-05)	3.7691e-01 (5.209e-04)	2.7672e-01 (1.465e-02)	**3.2113e-05 (4.028e-07)**
	50	1×10^5	8.8639e-01 (3.237e-05)	2.7862e-01 (9.282e-05)	8.2364e+00 (4.003e-01)	**2.6752e-03 (5.746e-05)**
	75	5×10^5	5.2318e+00 (3.911e-02)	1.8232e+01 (5.243e-03)	1.8562e+01 (3.461e-01	**4.4653e-02 (3.318e-05)**
	100	1×10^6	1.8704e+01 (6.282e-02)	3.5624e+00 (3.366e-03)	9.3534e+01 (4.568e-03)	**6.1219e-01 (4.346e-04)**
f_3	25	50,000	**1.00e-05**	**1.00e-05**	**1.00e-05**	**1.00e-05**
	50	1×10^5	**1.00e-05**	**1.00e-05**	6.7827e-05 (1.364e-08)	**1.00e-05**
	75	5×10^5	6.8734e-04 (5.329e-08)	1.2038e-04 (4.917e-08)	3.4039e-03 (2.112e-08)	**6.503e-05**
	100	1×10^6	2.1935e-03 (1.186e-08)	8.7325e-03 (5.521e-09)	1.6321e-03 (6.271e-04)	**8.3621e-04**
f_4	25	50,000	**1.00e-05**	**1.00e-05**	**1.00e-05**	**1.00e-05**
	50	1×10^5	**1.00e-05**	9.5043e-03 (4.895e-09)	2.1214e-04 (6.232e-05)	**1.00e-05**
	75	5×10^5	6.4543e-03 (3.434e-06)	2.1309e-02 (7.281e-06)	1.2651e-02 (3.862e-04)	**3.0071e-04 (1.832e-07)**
	100	1×10^6	8.3247e-02 (1.613e-06)	6.3298e-01 (1.217e-05)	1.3483e-01 (4.012e-02)	**3.2876e-03 (1.536e-07)**
f_5	25	50,000	**1.00e-05**	**1.00e-05**	**1.00e-05**	**1.00e-05**
	50	1×10^5	**1.00e-05**	**1.00e-05**	**1.00e-05**	**1.00e-05**
	75	5×10^5	1.0084e-03 (2.923e-06)	4.8271e-03 (4.294e-06)	9.7362e-03 (7.113e-04)	1.817e-04 (1.958e-07)
	100	1×10^6	3.6298e-02 (9.127e-04)	5.7309e-02 (1.682e-06)	3.2175e-03 (5.636e-05)	**2.2627e-04 (6.347e-05)**
f_6	2	50,000	**9.9980e-01**	9.99832e-01 (5.426-10)	9.99805e-01 (4.264e-08)	**9.9980e-01**

Table 4. Results of unpaired t-tests on the data of Table 3

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

Table 5. Number of runs converging to the cut off fitness value for six benchmark functions.

Function	Dim	No. of runs converging to the cut off			
		DE/rand/1/bin	DE/best/1/bin	SADE	AnDE
f_1	25	**30**	**30**	**30**	**30**
	50	**30**	**30**	**30**	**30**
	75	**30**	**30**	**30**	**30**
	100	**30**	**30**	**30**	**30**
	25	8	3	10	**17**
	50	0	0	0	**12**
	75	0	0	0	**6**

f_2	100	0	0	0	**3**
f_3	25	**30**	**30**	**30**	**30**
	50	**30**	**30**	9	**30**
	75	14	11	15	**30**
	100	6	5	1	**30**
f_4	25	**30**	**30**	**30**	**30**
	50	15	12	13	**30**
	75	6	8	11	**15**
	100	8	4	9	**17**
f_5	25	13	15	15	**30**
	50	12	17	11	**30**
	75	5	8	8	**19**
	100	2	4	4	**11**
f_6	2	**30**	12	17	**30**

Table 6. Mean no. of FEs and standard deviation required to converge to the cut-off fitness over the successful runs.

Function	Dim	Mean and Std Dev of FEs required to reach the Cut-off value			
		DE/rand/1/bin	DE/best/1/bin	SADE	AnDE
f_1	25	8682.05 (11.725)	9306.65 (15.873)	8836.90 (120.287)	**5232.85 (43.232)**
	50	22193. 00 (3.961)	24019. 65 (45.911)	26373. 50 (90.873)	**19282. 30 (29.732)**
	75	45092.45 (291.49)	44915.05 (20.982)	38274.75 (176.342)	**34789.73 (27.632)**
	100	989173.65 (23.88)	900024.60 (32.814)	938633.65 (183.983)	**581921.50 (73.818)**
f_2	25	7621.125 (24.821)	8915.35 (4.122)	8372.30 (51.876)	**6832.29 (12.342)**
	50	NA	NA	NA	**15628.83 (43.932)**
	75	NA	NA	NA	**33515.33 (113.432)**
	100	NA	NA	NA	**641234.67 (37.69)**
f_3	25	34884.45 (4.841)	46828.05 (5.583)	31522.30 (26.145)	**15775.75 (14.858)**
	50	67124.05 (34.991)	95614.95 (45. 526)	42313.67 (112.622)	**34322.15 (53.232)**
	75	416454.67 (7.806)	349772.57 (56.92)	263222.00 (154.34)	**195144.72 (34.95)**
	100	892452.50 (15.00)	752754.40 (4.975)	749991	**702321.43 (56.53)**
f_4	25	8725.55 (4.806)	12365.75 (7.039)	7973.25 (26.207)	**4875.15 (15.232)**
	50	17934.85 (18.910)	27568.33 (12.876)	12564.30 (62.863)	**9382.36 (3.982)**
	75	409281.33 (30.663)	447328.15 (35.723)	381305.67 (83.587)	**302615.13 (22.637)**
	100	726121.375 (12.655)	837261.25 (36.786)	562311.35 (31.747)	**509182.00 (18.753)**
f_5	25	37261.67 (9.57)	45930.57 (30.56)	21905.00 (28.418)	**10720.50 (19.547)**
	50	62918.57 (22.546)	90382 (32.822)	27483.46 (56.465)	**20483.05 (22.849)**
	75	267211.25 (7.558)	389222.40 (29.804)	169023.66 (137.632)	**56472.45 (32.118)**
	100	890332.50 (13.540)	990321.25 (21.525)	680491.25 (230.327)	**102317.25 (42.238)**
f_6	2	16472.85 (24.242)	27502.40 (14.803)	18291.86 (23.238)	**12237.60 (9.851)**

Table 7. Average and the standard deviation of the best-of-run solution for 30 runs for spread spectrum radar poly-phase code design problem (number of dimensions n = 19 and n = 20). For all cases each algorithm was run for 50,000 FEs.

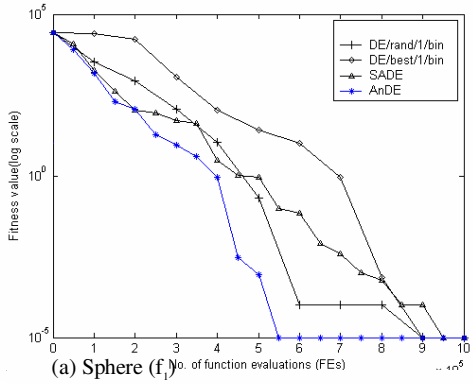
n	Mean best-of-run solution (Std Dev)			
	DE/rand/1/bin	DE/best/1/bin	SADE	AnDE
19	7.4849e-01 (8.93e-05)	7.5245e-01 (6.83e-03)	7.5172e-01 (3.72e-02)	**7.4441e-01 (6.89e-03)**
20	9.5746e-01 (4.83e-03)	9.0932e-01 (4.63e-03)	8.4532e-01 (5.62e-02)	**8.0814e-01 (7.93e-03)**

Table 8. Results of unpaired t-tests on the data of Table 7

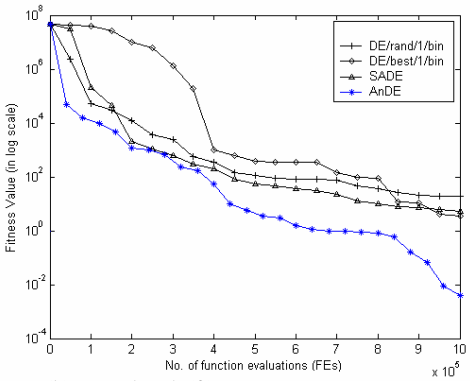
n	Std. Err	*t*	95% Conf. Intvl	Two-tailed *P*	Significance
19	0.002	2.5024	-0.00734 to -0.00081	0.0152	Significant
20	0.010	3.5880	-0.05792 to -0.01644	0.0007	**Extremely significant**

Table 9. Average no of FEs (for 30 runs) and standard deviations for spread spectrum radar polyphase code design problem to converge to a given cut-off value of the objective function

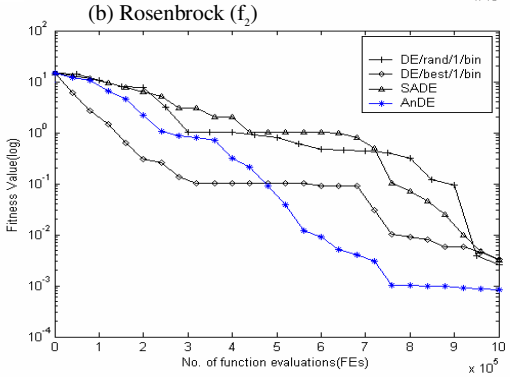
n	Cut off Value	Mean No. of FE required to reach the cut-off value			
		DE/rand/1/bin	DE/best/1/bin	SADE	AnDE
19	7.60e-01	48378.50	49129.15	45361.30	**41876.60**
20	9.10e-01	47973.60	48932.85	47654.50	**43131.30**



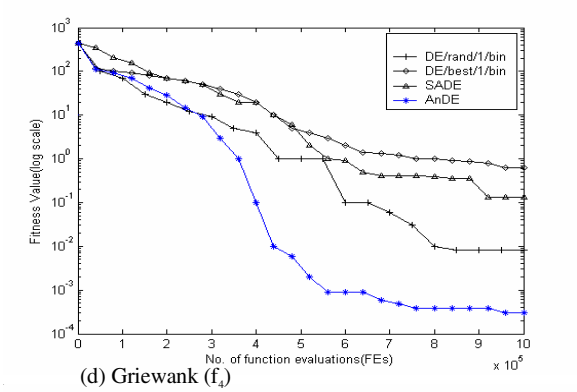
(a) Sphere (f_1)



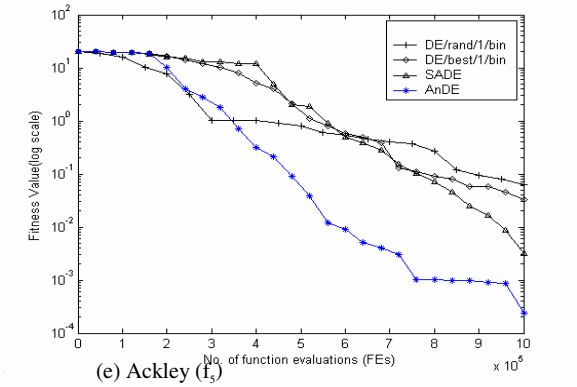
(b) Rosenbrock (f_2)



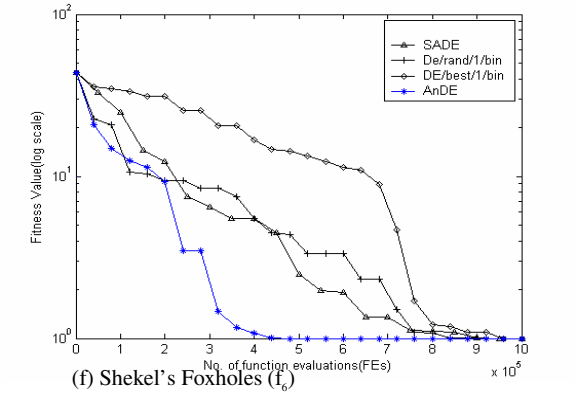
(c) Rastrigin (f_3)



(d) Griewank (f_4)



(e) Ackley (f_5)



(f) Shekel's Foxholes (f_6)

Fig.1 Progress to the optimum solution (all plots are for dimension = 100, except Shekel's foxhole which is 2-dimensional)

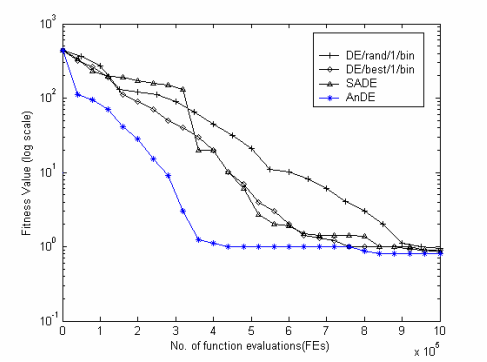
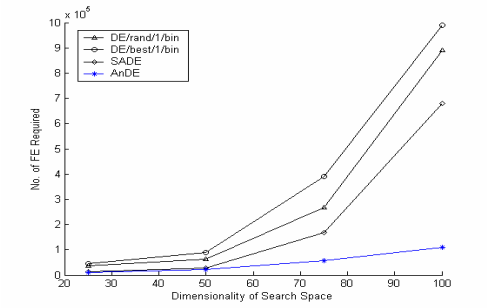
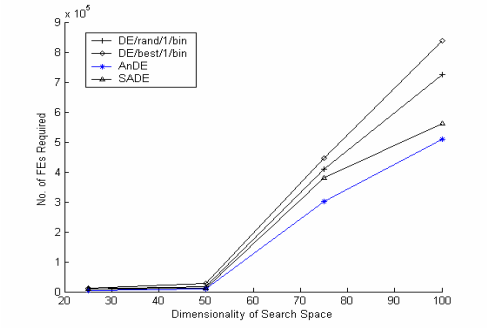


Fig.2 Progress to the optimum solution for Spread Spectrum Radar Polyphase Code Design Problem (with n = 20)



(a) Ackley (f_5)



(b) Griewank (f_4)

Fig 3. Variation of mean number of FE required for convergence with increase in dimensionality of the search space.

6. Conclusions

An improved DE has been presented, based on the use of the Simulated Annealing-based selection mechanism and a new center of mass based mutation strategy. The present scheme attempts to make a judicious use of the exploration and exploitation abilities of the search and is therefore more likely to avoid false or premature convergence in many cases. It has also been shown, empirically, to improve upon the convergence speed and scalability of the classical DE. The hybrid DE-variant has been compared against two classical (DE/rand/1/bin and DE/best/1/bin) and one state of the art (SADE) variant of DE using a six-function test suite and one additional real-world problem. The following performance metrics have been used: (a) solution quality, (b) speed of convergence, (c) frequency of hitting the optimum, and (d) scalability. The proposed algorithm has been shown to meet or beat the nearest competitor in a statistically meaningful way for most of the tested problems.

Bibliography

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

[2] Rogalsky, T., Derksen, R.W., and Kocabiyik, S.: Differential Evolution in Aerodynamic Optimization, In:

Proc. of 46th Annual Conf of Canadian Aeronautics and Space Institute (1999) 29-36.

[3] R.Joshi and A.C. Sanderson, "Minimal representation multisensor fusion using differential evolution", *IEEE Trans. Systems, Man, and Cybernetics,* Part A, vol 29, no. 1, pp. 63-76, 1999.

[4] Das, S., Konar, A.: Design of Two dimensional IIR Filters with Modern Search Heuristics: A Comparative Study, International Journal of Computational Intelligence and Applications, 2007 (Accepted).

[5] Lampinen, J.: A Bibliography of Differential Evolution Algorithm. Technical Report. Lappeenranta University of Technology, Department of Information Technology, Laboratory of Information Processing, 1999. Available via Internet http://www.lut.fi/~jlampine/debiblio.htm.

[6] Omran, M., Engelbrecht, A, P., Salman, A.: Differential Evolution Methods for Unsupervised Image Classification, Proc. Seventh Congress on Evolutionary Computation (CEC-2005). IEEE Press.

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

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

[9] Romeo, F. and Sangiovanni-Vincentelli, A.: Probabilistic Hill-Climbing Algorithms: Properties and Applications, Computer Science Press, Chapel Hill. NC, 1985.

[10] Price, Kenneth V. : An Introduction to Differential Evolution. In: David Corne, Marco Dorigo and Fred Glover (eds.) New Ideas in Optimization. McGraw-Hill, London (UK), pp. 79–108, 1999.

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

[12] Gamperle, R., Muller, S. D. and Koumoutsakos, A.: Parameter Study for Differential Evolution," in WSEAS NNA-FSFS-EC 2002, Interlaken, Switzerland, Feb. 11-15 2002. WSEAS.

[13] J. Ronkkonen, S. Kukkonen, and K. V. Price. Real Parameter Optimization with Differential Evolution, in The 2005 IEEE Congress on Evolutionary Computation (CEC2005), vol. 1, pages 506 – 513. IEEE Press, 2005.

[14] Liu, J. and Lampinen, J.: A Fuzzy Adaptive Differential Evolution Algorithm, In Soft computing- A Fusion ofFoundations, Methodologies and Applications, v.9 n.6, p.448-462, April 2005.

[15] Zaharie, D.: Control of Population Diversity and Adaptation in Differential Evolution Algorithms, In Matousek, D., Osmera P. (eds.), Proc. of MENDEL 2003, 9th International Conference on Soft Computing, Brno, Czech Republic, pp. 41-46, June 2003.

[16] Zaharie D. and Petcu D.: Adaptive Pareto Differential Evolution and its Parallelization, Proc. of 5th International Conference on Parallel Processing and Applied Mathematics, Czestochowa, Poland, Sept. 2003

[17] Abbass, H.: The Self-Adaptive Pareto Differential Evolution Algorithm, In: Proceedings of the 2002 Congress on Evolutionary Computation (2002) 831-836.

[18] Omran, M., Salman, A., Engelbrecht, A, P.: Self-adaptive Differential Evolution, Computational Intelligence And Security, PT 1, Proceedings Lecture Notes In Artificial Intelligence 3801: 192-199 2005.

[19] Das, S., Konar, A., Chakraborty, U.K., Two improved differential evolution schemes for faster global search, ACM-SIGEVO Proceedings of GECCO, Washington D.C., June 2005, pp. 991-998.

[20] Brest, J., Greiner, S., Boskovic, B., Mernik, M. and Zumer, V., Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems, IEEE Transactions on Evolutionary Computation, Vol. 10, and Issue: 6, Dec. 2006, pp. 646 – 657.

[21] Yan, J., Ling, Q., and Sun, D.: A Differential Evolution with Simulated Annealing Updating Method, Proceedings of the Fifth International Conference on Machine Learning and Cybernetics, Dalian, 13-16 August 2006.

[22] Metropolis, N., Rosenbluth, A. W.: Rosenbluth, M., Teller, A. H. and Teller, E.: Equation of State Calculations by Fast Computing Machines. J. Chem. Phys. 21, 1087-1092, 1953.

[23] Kirkpatrik, S., Gelatt, C. and Vecchi, M.: Optimization by Simulated Annealing. *Science*, 220: 671–680, 1983.

[24] van Laarhoven, P. and Aarts, E.: Simulated Annealing: Theory and Application, Kluwer Academic, 1987.

[25] Yao, X., Liu, Y., Lin, G. Evolutionary programming made faster, IEEE Transactions on Evolutionary Computation, vol 3, No 2, (1999) 82-102.

[26] Angeline, P. J.:Evolutionary optimization versus particle swarm optimization: Philosophy and the performance difference, Lecture Notes in Computer Science (vol. 1447), Proceedings of 7th International Conference on. Evolutionary Programming – Evolutionary Programming VII (1998) 84-89.

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