Journal of Global Optimization (2006) 35: 551–572
DOI 10.1007/s10898-005-3767-y

A Differential Free Point Generation Scheme in the Differential Evolution Algorithm

M. M. ALI[1], and L. P. FATTI[2]
[1]*School of Computational and Applied Mathematics, Witwatersrand University, 1 Jan Smuts Avenue, Johannesburg, South Africa (e-mail: mali@com.wits.ac.za)*
[2]*School of Statistics and Actuarial Science, Witwatersrand University, 1 Jan Smuts Avenue, Johannesburg, South Affrica*

(Received 9 May 2005; accepted in revised form 4 October 2005)

Abstract. In this paper we derive the probability distribution of trial points in the differential evolution (de) algorithm, in particular the probability distribution of points generated by mutation. We propose a point generation scheme that uses an approximation to this distribution. The scheme can dispense with the differential vector used in the mutation of de. We propose a de algorithm that replaces the differential based mutation scheme with a probability distribution based point generation scheme. We also propose a de algorithm that uses a probabilistic combination of the point generation by the probability distribution and the point generation by mutation. A numerical study is carried out using a set of 50 test problems, many of which are inspired by practical applications. Numerical results suggest that the new algorithms are superior to the original version both in terms of the number of function evaluations and cpu times.

Key words: Beta distribution, Continuous variable, Differential evolution, Global optimization, Population set based method, Probabilistic adaption

1. Introduction

The global optimization problem in this paper follows the form:

$$\text{minimize } f(x) \qquad \text{subject to} \quad x \in \Omega \subset \mathbb{R}^n, \tag{1}$$

where $f(x)$: $\Omega \mapsto \mathbb{R}$ is a continuous real-valued function. The domain Ω is defined by specifying upper (u^j) and lower (l^j) limits of each dimension j. Therefore, for any $x = (x^1, x^2, \dots, x^n) \in \Omega$, x^j is bounded, i.e. $l^j \leqslant x^j \leqslant u^j$. We denote the global optimal solution x^*, with its corresponding global optimal function value $f(x^*)$ or f^* for a short hand notation.

The differential evolution (de) algorithm [1] is a population set based algorithm [2] and is purely heuristic. All population set based algorithms use a population set S. The initial set

$$S = \{x_1, x_2, \dots, x_N\} \tag{2}$$

consists of N uniform random points in Ω. A contraction process is then used to drive these points to the vicinity of the global minimizer. The contraction process involves replacing bad point(s) in S with better point(s), per iteration. de attempts to replace all points in S by new points at each iteration. It progresses in an epoch or era base. During each epoch, N new function values are evaluated on N trial points. Trial points are generated using mutation and crossover.

In this paper, we identify a drawback of the mutant point (the point generated by the mutation of de) using the probability density of the mutant point. We demonstrate that the truncated probability density function of the mutant point can be closely approximated by the beta distribution. We then suggest two variants of the de algorithm. The first variant consists in replacing the 'differential' based mutation scheme with the beta distribution based point generation scheme. The second variant generates the trial points (by probabilistically) combining the mutation scheme with beta distribution.

This paper is divided into seven sections. In the next section, a brief description of de is given. Section 3 derives the probability density function (pdf) of the point generation. In Section 4, the motivation for trial point generation using the beta distribution is presented. In Section 5, the new algorithms are presented. Section 6 presents numerical results and finally, Section 7 contains the concluding remarks.

2. A Brief Description of de

The de algorithm attempts to replace each point in S with a new better point. Therefore, in each iteration, N competitions are held to determine the members of S for the next iteration. The ith ($i = 1, 2, \ldots, N$) competition is held to replace x_i in S. Considering x_i as the target point, a trial point y_i is found from two points (parents), the point x_i, i.e., the target point, and the primary trial point $\hat{x}_i$ (hereafter trial point) determined by the mutation operation. In its mutation phase, de randomly selects three distinct points $x_{p(1)}, x_{p(2)}$ and $x_{p(3)}$ from the current set S. None of these points should coincide with the current target point x_i. The weighted difference of any two points is then added to the third point which can be mathematically described as :

$$\hat{x}_i = x_{p(1)} + F(x_{p(2)} - x_{p(3)}), \tag{3}$$

where $F > 0$ is a scaling factor, and $x_{p(1)}$ is known as the base vector. If the point $\hat{x}_i \notin \Omega$ then the mutation operation is repeated. We denote the point generation using mutation by M_μ. The secondary trial point y_i is found from its parents x_i and $\hat{x}_i$ using the following crossover rule:

$$y_i^j = \begin{cases} \hat{x}_i^j & \text{if} \quad R^j \leqslant C_R \quad \text{or } j = I_i \\ x_i^j & \text{if} \quad R^j > C_R \quad \text{and } j \neq I_i, \end{cases} \tag{4}$$

where I_i is an integer randomly chosen with replacement from the set I, i.e., $I_i \in I = \{1, 2, \dots, n\}$; the superscript j represents the jth component of respective vectors; $R^j \in (0, 1)$, drawn uniformly for each j $(j = 1, 2, \dots, n)$. The ultimate aim of the crossover rule (4) is to obtain the secondary trial vector y_i with components coming from the components of the target vector x_i and mutated vector $\hat{x}_i$. This is ensured by introducing the parameter C_R and the set I. The targeting process continues until all members of S are considered. After all N secondary trial points y_i have been generated, acceptance is applied. In the acceptance phase, the function value at the secondary trial point, $f(y_i)$, is compared to $f(x_i)$, the value at the target point. If $f(y_i) < f(x_i)$ then y_i replaces x_i in S, otherwise, S retains the original x_i. Reproduction (mutation and crossover) and acceptance continue until some stopping conditions are met.

It can be seen from (3) that mutation (M_μ) is the main point generation mechanism of de. This operation calculates the coordinates of new points. The crossover operation (4) chooses the coordinates of a (secondary) trial point from the known coordinates of two points using a distribution controlled by C_R.

An important issue that needs to be addressed is the value of the scaling factor F in (3). To the best of our knowledge, no optimal choice of the scaling factor F has been suggested in the literature of de. For instance, in [3] F is a value in [0.4,0.8], in [4] a parameter-dependent anisotropic value, and in [5] dynamically calculated values are suggested. All of these are derived empirically, and in most cases choice of F varies from 0.4 to 1. However, in original de [1] F was chosen to lie in (0,2]. It appears that there has not been a coherent and systematic study using a large set of problems in seeking the optimal choice of F and the suggested values of the scaling factor have been largely dependent on small test problem sets used. We carried out numerical experiments with de using 50 test problems. Our numerical experiments found that M_μ often generates trial points outside the feasible region Ω and the number of points that fall outside Ω varies from problem to problem. However, we observed that the larger the F, the higher the number of such points are. On the other hand, the smaller the F, the higher the probability of de getting trapped in a local minimizer. The choice of F is therefore a delicate issue. We attempt to remedy this by generating trial points from a probability density. We derive the pdf of mutated points given by (3) and approximate the truncated pdf of the trial points with that of the a beta random variable (RV). We then generate trial points using a beta distribution.

We denote the pdf of a RV, say X by f_X and the joint pdf for RVs, say X and Y by f_{XY}.

3. Probability Density of Trial Points

In this section, we derive the pdf of mutated points generated by M_μ. We also look at the effect of F in this pdf. We then approximate a truncation of this pdf.

To see the effect of F in the pdf, it is enough to derive the pdf of a single coordinate of $\hat{x}_i$ in (3). Let the jth coordinate be our coordinate of interest, $j=1,2,\dots,n$. Our derivation of the pdf is based on the uniform distribution of points. Initially, points in S are generated uniformly from Ω and therefore points in all coordinate directions are independently uniform. However, as the contraction process continues, points in S will not be uniform over Ω. We consider three stages of the contraction process of S in de: the initial stage, the intermediate stage and the final stage. The initial stage is when S is uniform or near uniform in Ω. This may hold for only the initial few iterations of de. At this stage, the jth coordinate of points in S distributed uniformly in $[l^j, u^j]$, $j=1,2,\dots,n$. The intermediate stage is the stage when points in S will be non-uniform in Ω in that points will be distributed lower in the valley of the regions of attraction of different local minimizers. At the final stage the points in S will be distributed within the region of attraction of the global minimizer. If the function f is well behaved in that it can be approximated well by a quadratic near a minimizer, then the regions of attraction of the minimizer will likely be of an elliptical or spherical shape. Under this assumption, it is reasonable to assume that the points in a particular region of attraction are locally uniform within the region. At the final stage, the jth coordinate of points in S are distributed uniformly in $[x_l^j, x_u^j]$, where x_l^j $(\geqslant l^j)$ and x_u^j $(\leqslant u^j)$ are defined by $x_l^j=\min\{x_i^j\}$ and $x_u^j=\max\{x_i^j\}$ for all x_i in S. Points in S are unlikely to be uniform in Ω at the intermediate stage. However, we assume that subsets of points in S are locally uniform within their respective regions of attraction.

Under the above considerations, we derive the pdf of the jth coordinate of $\hat{x}_i$. Without loss of generality, let the jth coordinate be defined on [0,1]. Let the random variables be X_1 and $X_2 \sim U(0,1)$. We also define $Y_1=X_1$ and $Y_2=F(X_2-X_1)$, where F is the scaling factor in (3). By defining $y_1=u_1(x_1,x_2)=x_1$ and $y_2=u_2(x_1,x_2)=F(x_2-x_1)$, and by using the change of variable technique we can write $x_1=v_1(y_1,y_2)=y_1$ and $x_2=v_2(y_1,y_2)=y_1+\frac{1}{F}y_2$. Here, the values x_i and y_i are respectively the realizations of the RVs X_i and Y_i, $i=1,2$. The joint pdf $f_{Y_1Y_2}$ is given by

$$\begin{aligned} f_{Y_1Y_2}(y_1,y_2) &= |J|f_{X_1X_2}(v_1(y_1,y_2),v_2(y_1,y_2)) \\ &= |J|f_{X_1}(v_1(y_1,y_2))f_{X_2}(v_2(y_1,y_2)) = \frac{1}{F}, \end{aligned} \tag{5}$$

where J is the determinant of the Jacobian matrix

$$\begin{bmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & \frac{1}{F} \end{bmatrix}. \tag{6}$$

The supports (bounds) of y_1 and y_2 can be obtained from the supports of x_1 and x_2 (see, reference [6]). By using the bounds of x_1 and x_2, the bounds of y_1 and y_2 can be seen to be $0 \leqslant y_1 \leqslant 1$ and $-F \leqslant y_2 \leqslant F$. The other support in the y_1–y_2 plane can be found using the relationships between y_1 and y_2. These relationships are $y_1 + \frac{1}{F}y_2 = 0$ and $y_1 + \frac{1}{F}y_2 = 1$, for $x_2 = 0$ and $x_2 = 1$, respectively. The marginal density $f_{Y_2}(y_2)$ can be calculated from the integrals:

$$f_{Y_2}(y_2) = \int_0^{1-\frac{1}{F}y_2} \frac{1}{F}\mathrm{d}y_1 = \frac{1}{F}\left(1 - \frac{1}{F}y_2\right), \quad 0 \leqslant y_2 \leqslant F, \tag{7}$$

and

$$f_{Y_2}(y_2) = \int_{-\frac{1}{F}y_2}^{1} \frac{1}{F}\mathrm{d}y_1 = \frac{1}{F}\left(1 + \frac{1}{F}y_2\right), \quad -F \leqslant y_2 \leqslant 0. \tag{8}$$

Combining (7) and (8) we write

$$f_{Y_2}(y_2) = \frac{1}{F}\left(1 - \frac{1}{F}|y_2|\right), \quad -F \leqslant y_2 \leqslant F. \tag{9}$$

We now let $Y_3 \sim U(0,1)$ independently of Y_2. We consider the joint pdf of Y_3 and $Y_3 + Y_2$. We denote $Z_1 = Y_3$ and $Z_2 = Y_3 + Y_2$, where $Y_2 = F(X_2 - X_1)$. The joint pdf $f_{Z_1Z_2}$ is given by

$$\begin{aligned} f_{Z_1Z_2}(z_1,z_2) &= \left|J\right| f_{Y_3Y_2}(v_1(z_1,z_2),v_2(z_1,z_2)) = |J|f_{Y_2}(y_2)f_{Y_3}(y_3) \\ &= f_{Y_2}(z_2 - z_1), \end{aligned} \tag{10}$$

since $|J| = 1$, $v_1(z_1,z_2) = y_3 = z_1$ and $v_2(z_1,z_2) = y_2 = z_2 - z_1$. Therefore, the joint pdf $f_{Z_1Z_2}$ is given by

$$\begin{aligned} f_{Z_1Z_2}(z_1,z_2) = \frac{1}{F}\left(1 - \frac{1}{F}|z_2 - z_1|\right), \quad & 0 \leqslant z_1 \leqslant 1, \\ & z_1 - F \leqslant z_2 \leqslant z_1 + F. \end{aligned} \tag{11}$$

Clearly, the shape of the joint pdf $f_{Z_1Z_2}(z_1,z_2)$ is dependent on F. The marginal pdf of Z_2 will be our desired pdf. To approximately cover the ranges of F suggested in literature we consider two cases : (i) $F>1$ and (ii) $0.5\leqslant F\leqslant 1$.

Case (i): The marginal pdf is given by

$$f_{Z_2}(z_2)=\begin{cases}\frac{1}{F}\int_0^{z_2+F}\left(1-\frac{1}{F}(z_1-z_2)\right)\mathrm{d}z_1, & -F\leqslant z_2\leqslant 1-F,\\ \frac{1}{F}\int_0^{1}\left(1-\frac{1}{F}(z_1-z_2)\right)\mathrm{d}z_1, & 1-F\leqslant z_2\leqslant 0,\\ \frac{1}{F}\int_0^{z_2}\left(1-\frac{1}{F}(z_2-z_1)\right)\mathrm{d}z_1+\frac{1}{F}\int_{z_2}^{1}\left(1-\frac{1}{F}(z_1-z_2)\right)\mathrm{d}z_1, & 0\leqslant z_2\leqslant 1,\\ \frac{1}{F}\int_0^{1}\left(1-\frac{1}{F}(z_2-z_1)\right)\mathrm{d}z_1, & 1\leqslant z_2\leqslant F,\\ \frac{1}{F}\int_{z_2-F}^{1}\left(1-\frac{1}{F}(z_2-z_1)\right)\mathrm{d}z_1, & F\leqslant z_2\leqslant 1+F.\end{cases}\tag{12}$$

Case (ii) : The marginal PDF is given by

$$f_{Z_2}(z_2)=\begin{cases}\frac{1}{F}\int_0^{z_2+F}\left(1-\frac{1}{F}(z_1-z_2)\right)\mathrm{d}z_1, & -F\leqslant z_2\leqslant 0,\\ \frac{1}{F}\int_0^{z_2}\left(1-\frac{1}{F}(z_2-z_1)\right)\mathrm{d}z_1+\frac{1}{F}\int_{z_2}^{z_2+F}\left(1-\frac{1}{F}(z_1-z_2)\right)\mathrm{d}z_1 & 0\leqslant z_2\leqslant 1-F,\\ \frac{1}{F}\int_0^{z_2}\left(1-\frac{1}{F}(z_2-z_1)\right)\mathrm{d}z_1+\frac{1}{F}\int_{z_2}^{1}\left(1-\frac{1}{F}(z_1-z_2)\right)\mathrm{d}z_1, & 1-F\leqslant z_2\leqslant F,\\ \frac{1}{F}\int_{z_2-F}^{z_2}\left(1-\frac{1}{F}(z_2-z_1)\right)\mathrm{d}z_1+\frac{1}{F}\int_{z_2}^{1}\left(1-\frac{1}{F}(z_1-z_2)\right)\mathrm{d}z_1 & F\leqslant z_2\leqslant 1,\\ \frac{1}{F}\int_{z_2-F}^{1}\left(1-\frac{1}{F}(z_2-z_1)\right)\mathrm{d}z_1, & 1\leqslant z_2\leqslant 1+F.\end{cases}\tag{13}$$

Integrations involved in the above pdfs (12) and (13) can easily be carried out. The pdfs $f_{Z_2}(z_2)$ given by (12) and (13) are both proper in the sense that their integrations result in unity for F in the respective ranges. The continuity of F is also preserved as the pdfs $f_{Z_2}(z_2)$ in (12) and (13) are equal for $F=1$ and this is given by

$$f_{Z_2}(z_2)=\begin{cases}\frac{1}{2}+z_2+\frac{z_2^2}{2}, & -1\leqslant z_2\leqslant 0,\\ \frac{1}{2}+z_2-z_2^2, & 0\leqslant z_2\leqslant 1,\\ 2-2z_2+\frac{z_2^2}{2}, & 1\leqslant z_2\leqslant 2.\end{cases}\tag{14}$$

For illustration, we consider five values of F, two from each range $(1,\infty)$ and [0.5,1], and $F=1$, and present the pdf for each of these values of F in Figure 1. It is evident from the figure that even for $F=0.5$, mutated points may fall outside the defined region [0, 1]. For $F=1$ integration of (14) shows that a third of the points fall outside [0, 1]. The phenomenon of points falling outside Ω is expected in de. In the early stages scattered points in S result in large differential vectors $(x_{p(2)}-x_{p(3)})$ in (3) which cause $\hat{x}_i$ to fall outside Ω. The scaling factor F also has an effect in it. At the final stage when the points in S form a cluster around the global minimizer, $\hat{x}_i$ will only fall outside Ω if the global minimizer lies close to the boundary of Ω.

Given a suitable value for F, we can draw trial points $\hat{x}_i$ from the pdf (12) or (13) instead of using M_μ in (3). However, trial points may still

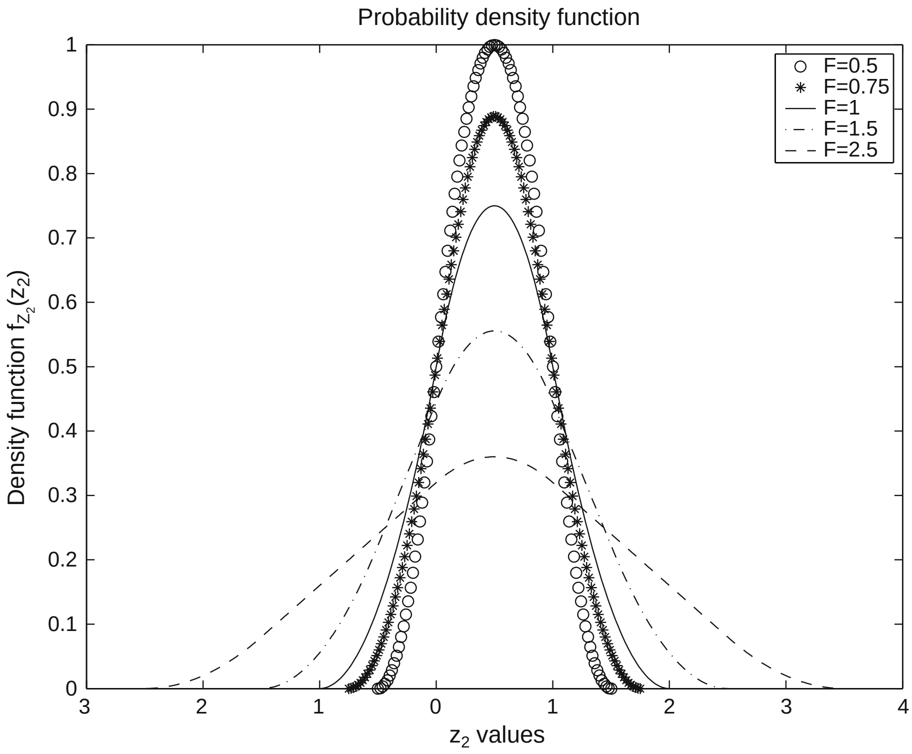

Figure 1. Probability density function.

fall outside Ω. We face two difficulties with this alternative approach. The first difficulty is in choosing a value of F that reduces the phenomenon of points falling outside, but at the same time maintains the exploratory feature of de. This is the same problem as encountered with M_μ. The second difficulty is in generating a point using either of the pdfs. This involves a nonlinear equation to be solved for each coordinate of the point. For example, to generate the jth component $\hat{x}_i^j$ of trial point $\hat{x}_i$ when $F=1$, one needs to integrate (14) to get $\hat{x}_i^j$ such that

$$r=\int_0^{\hat{x}_i^j} f_{z_2}(z_2)\mathrm{d}z_2, \tag{15}$$

where r is a uniform random variable in [0,1].

One approach of preventing points from falling outside Ω would be to truncate the density. The RV of such a truncated density can be conditionally defined as follows

$$Z_2'=\left\{Z_2\middle|Z_2\in[0,1]\right\}. \tag{16}$$

The pdf $f_{Z_2'}(z_2)$ of the observed points in [0,1] can be written as

$$f_{Z_2'}(z_2) = \begin{cases} \frac{f_{Z_2}(z_2)}{\Pr(0 \leqslant Z_2 \leqslant 1)} & \text{if } z_2 \in [0,1], \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

However, one cannot avoid the integration (15) for each coordinate of $\hat{x}_i$ using a truncated pdf. Moreover, as the contraction process continues the interval on which z_2 is defined will continue to shrink. Thus, the truncation may not even be necessary, unless, of course, the global minimizer lies close to the boundary of Ω.

To alleviate the above problems, we propose a β-density as an approximation to the truncated pdf. This approximation is justified by the shape of the pdfs in Figure 1. The useful property of the β-distribution is that the points will no longer fall outside the range [0,1]. However, in order to use this distribution, its standard deviation has to evolve in a similar way as the differential vector evolves. The mean of the β-distribution should also vary in the same way as the base vector varies for each targeted point. The β-density therefore must have the property of self-adjusting its shape as the contraction process, per iteration, proceeds. We can therefore iteratively generate trial points using the evolving β-distribution instead of using M_μ. We refer to this point generation as β-driven mutation and denote it by M_β.

4. The Self-Adjusting β-Density

The β-distribution on [0,1] has probability density given by

$$f_Z(z) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} z^{a-1}(1-z)^{b-1}, \quad 0 \leqslant z \leqslant 1, \qquad a, b > 0, \tag{18}$$

with mean $a/(a+b)$ and variance $ab/(a+b)^2(a+b+1)$. The symbol Γ represents the gamma function. Clearly, the values of a and b determine the shape of the density function. Since we are interested in the jth component $\hat{x}_i^j$ of a trial point $\hat{x}_i$, we let a^j and b^j represent the parameters of the β distribution in the jth coordinate, $j = 1, 2, \ldots, n$. We obtain the values of a^j and b^j from a given mean and a given standard deviation. We denote a randomly generated value of this coordinate by $\hat{x}^j$. Since $\hat{x}^j$ must lie between the upper and lower limits, u^j and l^j, respectively of the jth coordinate, $\hat{x}^j = k^j z^j + l^j$, where $k^j = u^j - l^j$ and z^j is a random realization from a $\beta(a^j, b^j)$ distribution. We define $\theta^j = (\underline{x}^j - l^j)/k^j$ and $A^j = \left[(k^j)^2 \theta^j (1-\theta^j)/(s^j)^2\right] - 1$, where $\underline{x}^j$ and s^j are the given mean and standard deviation of the β random variable Z^j with realization z^j, respectively. We use these to write $a^j = A^j \theta^j$ and $b^j = A^j(1-\theta^j)$. An estimate of the standard deviation s^j would be the sample standard deviation of the jth components of the N points in the current set S. However, we find that the following parameter based formula works in practice:

$$s^j = \tau \times \left|x^j_{\max} - x^j_{\min}\right|, \tag{19}$$

where τ is an empirically based parameter, and $x_{\max}, x_{\min}$ and their function values $f_{\max}, f_{\min}$ are such that

$$f(x_{\max}) = f_{\max} = \max_{x \in S} f(x) \quad \text{and} \quad f(x_{\min}) = f_{\min} = \min_{x \in S} f(x). \tag{20}$$

We choose the mean $\underline{x}$ as the best of the two uniform random points in S, each time we target to replace x_i in S. We also let the β-distribution be defined on $[l^j, u^j]$.

In the early stages, while the spread of points in S is still large, the calculated parameters a^j and b^j could take on negative values, which are not permitted, or values between 0 and 1, for which the β-distribution is U-shaped. We therefore restrict them to values in the range $[1, \infty)$, replacing them by 1 if they fall outside it. Notice that in the limiting case $a^j = b^j = 1$, the β-distribution is a uniform distribution. Hence, if our standard deviation is high we will be generating a realization from a uniform distribution. We use the algorithm of Cheng [7] for generating the β-variates.

5. Differential Free Point Generation in de

In this section, we propose two new versions of de. The first version replaces M_μ with M_β. This version of de is referred to as the differential free de algorithm (fde). The main feature of fde is that trial points are always generated inside the search region Ω. The second version generates trial points using some probability distribution on $\{M_\mu, M_\beta\}$. We refer to this version of de as the probabilistic differential free de algorithm (pfde). The main feature of pfde is that it probabilistically adapts a trial points generation scheme that solves a given problem in a robust manner. Full details of the new algorithms are given below.

5.1. THE DIFFERENTIAL FREE DE ALGORITHM

The fde differs from de in that it replaces M_μ with M_β. Like de, fde also targets all N points in S at each iteration in order to replace them with better points. When $x_i, i = 1, 2, \dots, N$, is targeted, fde randomly selects two points in S (as opposed to three points in M_μ of de) and uses the best of the two, say $\underline{x}$ as the mean of the β-distribution. Notice that unlike de, fde does not exclude the targeted point x_i from being selected. Using $\underline{x}$ as the mean and the standard deviation s as defined by (19), fde generates a trial point, say, $\hat{x}_i$, using the β-distribution. Since the targeted point

x_i is not excluded from being selected it may emerge as the mean of the β-distribution. If it does then a local exploration is carried out around targeted point x_i using the β-distribution, a feature that de lacks. As in the case of de, fde performs crossover (4) in obtaining the secondary trial point y_i. We now present the algorithm for fde.

ALGORITHM 1. The fde Algorithm

Step 1. *Determine the initial set S*

$$S = \{x_1, x_2, \dots, x_N\},$$

where the points x_i, $i = 1, 2, \dots, N$, are sampled randomly in Ω; evaluate $f(x)$ at each x_i. Take $N \gg n$, n being the dimension of the function $f(x)$. Set iteration counter $k = 0$.

Step 2. *Determine the best and the worst point in S*. Determine the points $x_{\max}$ and $x_{\min}$. If the stopping condition is satisfied, then stop.

Step 3. *Generate points to replace points in S*. For each $x_i \in S$ ($i = 1, 2, \dots, N$), determine the secondary trial point y_i by the following two operations:

- Mutation using M_β : Randomly select two points from S (which may include x_i, the targeted point) and find the best point, say $\underline{x}$ of the two. Obtain $\hat{x}_i^j$ using β-distribution with mean $\underline{x}^j$ and standard deviation $\tau \times |x_{\max}^j - x_{\min}^j|$, $j = 1, 2, \dots, n$. Calculate each component $\hat{x}_i^j$ of $\hat{x}_i$.
- Crossover : Calculate the secondary trial vector y_i corresponding to the target x_i from x_i and $\hat{x}_i$ using the crossover rule (4).

Step 4. *Replace points in S*. Select each trial vector y_i for the $(k+1)$th iteration using the acceptance criterion : replace $x_i \in S$ with y_i if $f(y_i) < f(x_i)$ otherwise retain x_i. Set $k := k + 1$ and go to Step 2.

5.2. THE PROBABILISTIC DIFFERENTIAL FREE DE ALGORITHM

The pfde algorithm generates trial points $\hat{x}_i$ using some probability distribution over the set $\{M_\mu, M_\beta\}$. That is, trial points are either generated using M_μ or M_β at each iteration. Initially equal probabilities (0.5) are assigned to both scheme M_μ and M_β and these probabilities are updated according to some rules based on reward (for being successful) and penalty (for being unsuccessful). This probabilistic adaptation in the algorithm guides pfde in deciding on which mutation scheme to use most in generating points for any given problem. This allows the algorithm to bias the the rule that solves a given problem in a most efficient and robust manner.

Our motivation for this modification is purely based on numerical experiments with de and fde using 50 test problems. We observed that fde, that uses only M_β, performed well in terms of the number of function evaluations for most of the problems, while for few others it performed poorly. The same was true for de which uses M_μ. de was superior to fde in terms of locating the global minimum value but much inferior in terms of the number of function evaluations. This motivated us to introduce a scheme that combines M_μ and M_β probabilistically. This probabilistic scheme penalizes (rewards) a mutation scheme for not making (making) good progress. We combine the scheme M_μ with M_β so that mutated points are either generated with some probability α_k using M_μ or with probability $\gamma_k = 1 - \alpha_k$ using M_β. A scheme is selected with some probability (e.g. M_μ is first selected with probability 0.5) to create mutated points, and this probability is updated after each iteration. If $\hat{x}_i$ are generated, say using M_μ, and a higher number of these points are found to be producing much better secondary trial points compared to the current N points in S then the probability for using the scheme M_μ is increased (reward). We use the following scheme for increasing the probability for M_μ:

$$\alpha_k = \alpha_{k-1} + \tfrac{1}{2}\alpha_{k-1}(1 - \alpha_{k-1}), \tag{21}$$

and γ_k is obtained using $\gamma_k = 1 - \alpha_k$. If, however, the secondary trial points are not better in comparison to the current N points in S then the probability of M_μ is decreased (penalty) using

$$\alpha_k = \alpha_{k-1} - \tfrac{1}{2}\alpha_{k-1}(1 - \alpha_{k-1}). \tag{22}$$

The motivation for the use formulae (21) and (22) in probabilistic adaptation can be found in [8] where they have been used in the context of combinatorial optimization using learning automata. The value $\frac{1}{2}$ used in (21) and (22) controls the increment on α_k in (21) and the reduction on α_k in (22). Other values can also be used but our numerical studies have suggested that it is a good value to choose.

To determine whether to reward or to penalize a mutation scheme we use the following empirical scheme. We count the number of replacements (nr) in S in an iteration. If nr is greater than or equal to the nearest integer to $0.6N$ then we reward the mutation scheme used. If it is less than or equal to $0.3N$ than we penalize the scheme. If, however, nr falls between $0.3N$ and $0.6N$ then we neither reward or penalize the scheme. This adaptive process tends to let the algorithm decide for itself which scheme to use most in generating trial points for any given problem so that it solves the problem in a much more efficient way. This is done by increasing the value of α_k whenever M_μ gives more favourable points, thus reducing the probability of using M_β. On the other hand, if M_β produces a high nr then α_k

is reduced. We force $\alpha_k, k \geqslant 1$, to lie in $[0.05, 0.95]$. For example, if α_k goes below 0.05, we set $\alpha_k = 0.05$ by clipping. This is done in order to avoid the algorithm switching entirely to one scheme. Thus α_k and γ_k lie in $(0, 1)$ to allow the algorithm to be able to switch from one scheme to the other. The algorithm for pfde is described below.

ALGORITHM 2. The pfde Algorithm

Step 1. Determine the initial set S. Same as in Algorithm 1. Set $\alpha_k = 0.5$.

Step 2. Determine the best and the worst point in S; check the stopping condition. Same as in Algorithm 1.

Step 3. Generate points to replace points in S. If $\omega \leqslant \alpha_k$, where ω is a random number in [0, 1], then go to Step 3a else go to Step 3b.

- *Step 3a.* Mutation rule M_μ: For each $x_i \in S(i = 1, 2, \ldots, N)$ determine $\hat{x}_i$ using (3). Go to Step 3c.
- *Step 3b.* Mutation using M_β: Randomly select two points from S (which may include x_i, the targeted point) and find the best point, say $\underline{x}$ of the two. Obtain $\hat{x}_i^j$ using β-distribution with mean $\underline{x}^j$ and standard deviation $\tau \times |x_{\max}^j - x_{\min}^j|$. Calculate each component $\hat{x}_i^j$ of $\hat{x}_i$. Go to Step 3c.
- *Step 3c.* Crossover: Calculate the secondary trial vector y_i corresponding to the target x_i from x_i and $\hat{x}_i$ using the crossover rule (4).

Step 4. Replace points in S. Set $\mathrm{nr} = 0$. Select each trial vector y_i for the $(k+1)$th iteration using the acceptance criterion : If $f(y_i) < f(x_i)$ then set $\mathrm{nr} = \mathrm{nr}+1$ and replace $x_i \in S$ with y_i otherwise retain x_i. If this Step is reached from Step 3a then go to Step 4a else go to Step 4b.

- *Step 4a.* If $\mathrm{nr} \geqslant 0.6N$ then $\alpha_k = \alpha_{k-1} + \frac{1}{2}\alpha_{k-1}(1 - \alpha_{k-1})$. If $\mathrm{nr} \leqslant 0.3N$ then $\alpha_k = \alpha_{k-1} - \frac{1}{2}\alpha_{k-1}(1 - \alpha_{k-1})$. Set $k = k+1$ and go to Step 2.
- *Step 4b.* If $\mathrm{nr} \geqslant 0.6N$ then $\alpha_k = \alpha_{k-1} - \frac{1}{2}\alpha_{k-1}(1 - \alpha_{k-1})$. If $\mathrm{nr} \leqslant 0.3N$ then $\alpha_k = \alpha_{k-1} + \frac{1}{2}\alpha_{k-1}(1 - \alpha_{k-1})$. Set $k = k+1$ and go to Step 2.

REMARKS

1. It can be seen from Algorithm 2 above that when $\alpha_k = 0$ then we have the fde algorithm and when $\alpha_k = 1$ then we have the de algorithm.

6. Numerical Results

In this section we judge the performance of the new algorithms using a collection of 50 test problems. These problems range from 2 to 20 in dimension and have a variety of inherent difficulty. All the problems have continuous variables. A detailed description of each test problem (P) in the collection can be found in [9]. We compare the results obtained by the new algorithms with those of the de algorithm. The algorithms were run 100 times on each of the 50 test problems to determine the success rate (sr) (or percentages of success) of each algorithm. There were 5000 runs in total. We calculated the average number of function evaluations (fe) and cpu times (cpu) for those runs for which the global minima were found. We used sr, fe and cpu as the criteria for comparison. A solution to the problem need not be the global minimum f^* exactly, but may be any value less than f^*_ε,

$$f^*_\varepsilon = f^* + \varepsilon, \tag{23}$$

where $\varepsilon = 9 \times 10^{-4}$. Therefore, a run was terminated when either the best function value in S was identical to the optimal solution to at least three decimal digits, or the maximum number of iteration (T) was reached. We also counted a run if it failed to satisfy (23) but produced the global minimum within either one or two decimal digits of accuracy. We refer such a run as a near success (ns) run. The number of function evaluations and cpu times for these runs are not reflected in our comparison.

All the algorithms have some parameter values that are to be provided by the user. We first discuss the parameters that are common to all algorithms. For example, maximal iteration parameter T and the size N of the population set S. We took the value of T to be 10,000 and the value of N to be $10n$, where n is the dimension of the problem. These are heuristic choices. For example, the value of N can always be increased for obtaining the global minimum with higher probability. However, the higher the value of N, the higher the number of fe is. Other parameters of de are the scaling parameter F in its mutation scheme (3) and controlling parameter C_R in its crossover scheme (4). The effect of C_R has been studied in [2,10] and it was found that $C_R = 0.5$ is a good choice. We have also conducted a series of runs of de using values (F varying from 0.3 to 1.25; C_R from 0.25 to 0.9) for each of these parameters. The best results obtained using the 50 problems were for $F = 0.75$ and $C_R = 0.5$. The parameter values for all algorithms were found empirically using the 50 test problems. We do not claim these values to be the optimal for any given problem in general but they are good values to choose.

A parameter associated with fde is τ in (19). Our numerical studies found that the results of fde were sensitive to τ. We therefore present the results of fde for several values of the parameter τ.

To see the effect of τ in fde we present the results of fde with varying τ in Table I. We also present the results of de in the same table to make a comparison between fde and de. We have excluded the Odd Square function (OSP) and Storn's Tchebychev function (ST) from Table I as they were not solved by either de or fde. In Table I, the first column contains the problem name [9] with its dimension given in brackets; the column fo represents the average number of points per successful run that fall outside the feasible region in de. Trial points do not fall outside Ω in fde. The last row in Table I presents the total results. We now look at the effect of τ in fde. We use $\tau = 0.25, 0.5$ and 1 for this. Total results in the last row indicate a substantial decrease in fe with a decrease in τ. There are a number of problem for which the sensitivity to τ is higher than others. For example, the Ackley problem (ACK), the Epistatic Michalewicz (EM), the Neumaier's problems (NF2 and NF3), the Salomon problem (SAL), the Shekel family (S5, S7 and S10) and the Shekel foxhole (FX) dominate fe. A change in τ makes a considerable impact in fe for these problems. In terms of fe $\tau = 0.25$ outperforms the other two. In terms of sr $\tau = 0.25$ also remains superior. However, when we reduced τ to 0.15 then sr deteriorated. Therefore, we use $\tau = 0.25$ for fde for our next comparisons.

We now compare fde with de using total results in Table I. In terms of fe fde with $\tau = 0.25$ outperforms de by about 15%. However, in terms of sr de outperforms fde by about 6%. By looking at ns we see that there were more runs in fde than in de that were unable to produce solutions to three decimal digits of accuracy, but were able to obtain solutions with either one or two decimal digits of accuracy. If we compare de with fde for $\tau = 0.25$ by assuming the number of runs in brackets (ns) as successes, we see that in terms of sr+ns de remains superior to fde by 3% runs. This shows that fde is less robust than de in obtaining accurate global minima. Both the algorithms, however, were able to obtain minimum values for 46 problems. There are however two functions, namely the Rosenbrock function (RB) for which de succeeded but fde failed, and the Salomon function (SAL) for which the fde was successful in 21% of the runs but de failed. Both de and fde failed with the PTM function. If we compare de with fde by excluding the results of these two functions, we see that the comparison is even more favourable to fde in terms of fe.

During our numerical experiments, we observed that both algorithms had their strengths and weaknesses. One of the weaknesses of de is that the trial points fall outside Ω. Specially, for functions with global minimum near to the boundary. For instance, for the function SWF about 56% of all trial points fell outside. On average, per function, 21% of the trial points

Table I. Comparison of De and Fde using 48 test problems

	fde						de		
	$\tau=1$		$\tau=0.5$		$\tau=0.25$		$C_R=0.5$, $F=0.75$		
$P(n)$	fe	sr(ns)	fe	sr(ns)	fe	sr(ns)	fe	sr(ns)	fo
ACK(10)	243,556	100	60,048	100	21,556	100	203,684	100	1,789
AP(2)	530	100	372	100	1,490	100	450	100	19
BL(2)	827	100	824	100	607	98(2)	587	100	271
B1(2)	1,004	98	656	96	1,086	75(25)	746	100	22
B2(2)	1,526	100	803	97	727	69(31)	870	100	23
BR(2)	14,059	100	6,540	100	1,911	100	1,002	100	330
CB3(2)	882	100	414	100	247	96(4)	498	100	24
CB6(2)	1,714	100	614	100	714	100	656	100	20
CM(2)	358	100	247	100	622	100	338	100	21
DA(2)	15,787	98(2)	4,402	100	5,221	100	1,427	100	201
EP(2)	974	100	598	100	532	98(2)	668	100	67
EM(10)	901,232	5(41)	860,714	19(38)	530,190	26(45)	538,619	84(10)	333,565
EXP(10)	11,626	100	4,386	100	2,878	100	9,478	100	1,254
GP(2)	2,173	100	772	100	612	94	714	100	41
GW(10)	311,854	100	73,450	100	24,066	98(2)	251,408	100	1,334
GRP(3)	4,376	100	2,028	100	1,475	100	1,408	100	407
H3(3)	1,274	100	655	100	439	100	712	100	208
H6(6)	298,894	62(38)	91,141	100	25,060	100	6,644	100	3,220
HV(3)	70,881	100	14,152	100	5,327	90(5)	3,372	100	95
HSK(2)	338	100	254	100	544	100	295	100	37
KL(4)	132	100	133	100	116	100	565	100	88
LM1(3)	1,414	100	802	100	627	100	1162	100	102
LM2(10)	17,598	100	6,500	100	4,188	100	14,164	100	1,887
MC(2)	1,010	100	542	100	605	100	490	100	71
MR(3)	318	100	267	100	234	100	292	100	83
MCP(4)	794	100	563	100	463	100	455	100	264
ML(10)	246,180	96(4)	130,351	93(7)	29,673	80(20)	21,630	100	13,711
MRP(2)	13,716	94(6)	5,415	82(18)	1,284	80(20)	1,256	40(20)	22
MGP(2)	4,781	94(6)	1,357	76	163	64	939	80	25
NF2(4)	277,458	26(74)	214,858	69(31)	127,486	98(2)	106,205	96(4)	27,616
NF3(10)	0	(9)	632,900	25(75)	225,792	100	116,898	100	5,815
PP(10)	24,872	100	8,680	100	5,274	100	18,562	100	7,194
PRD(2)	7,829	96	4,359	87(3)	2,276	86	1,691	98(2)	523
PQ(4)	16,690	100	4,316	100	3,244	100	4,112	100	400
PTM(9)	0	0(2)	0	0(5)	0	0(58)	0	0(91)	0
RG(10)	192,804	100	63,446	100	21,138	100	152,222	100	5,635
RB(10)	0	0(4)	0	0(8)	0	0(14)	243,028	100	2,813
SAL(5)	390,239	7(93)	317,483	14(86)	303,414	21(79)	0	0(100)	0
SF1(2)	19,225	98(2)	13,447	16(14)	17,883	51(49)	5,009	46	50
SF2(2)	2,735	100	1,598	100	3,422	100	1,932	100	17
SBT(2)	76,548	52(47)	40,684	84(16)	11,834	100	3,574	100	1,041
SWF(10)	53,240	100	27,610	100	15,260	100	43,948	100	52,796
S5(4)	314,880	20(80)	85,513	81(5)	21,992	66(9)	6,140	98	813
S7(4)	288,532	22(78)	42,941	96	4,688	81	5,028	100	510
S10(4)	266,445	48(52)	34,033	95	6,454	93	5,032	100	421
FX(5)	423,900	7(9)	274,350	20	95,625	18	16,560	22	6,217
SIN(20)	583,736	97	51,536	100	21,116	100	120,744	100	45,186
WP(4)	321,560	5(95)	226,198	58(42)	78,298	99(1)	13,822	100	242
Total	**53,77,261**	**3,725 (642)**	**33,12,952**	**3,932 (690)**	**16,27,853**	**4,081 (368)**	**19,29,036**	**4,364 (227)**	**5,16,490**

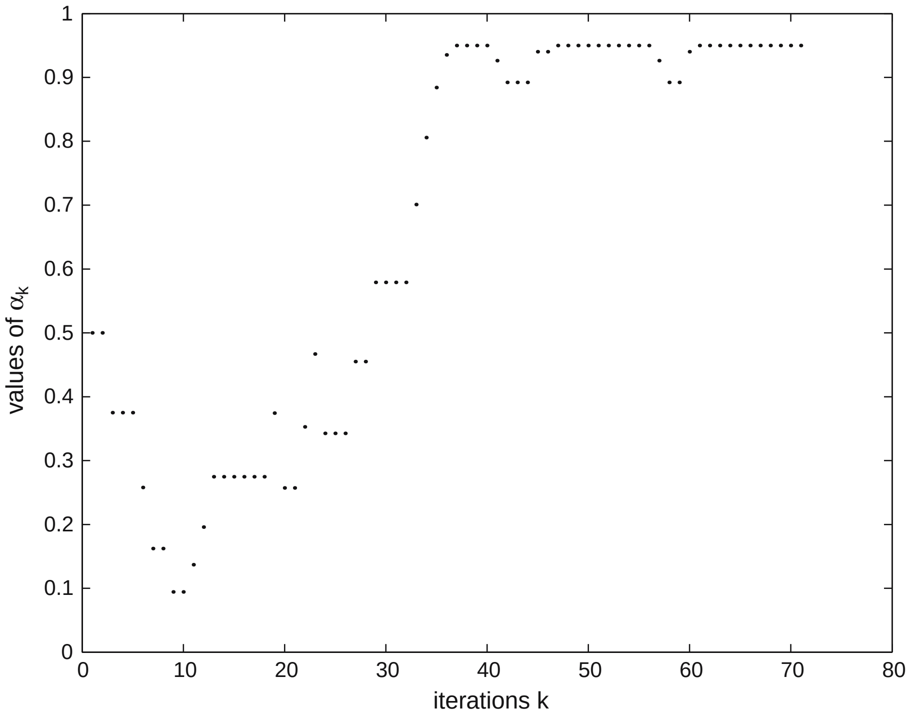

Figure 2. Adaptation of pfde on H6.

of de fell outside. Another weakness of de is that it requires a higher number of fe on average than fde. A strength of de is that it produces a higher number of successes. A strength of fde is that it can reach the vicinity of the global minimum quickly, requiring less fe.

We observed that fde reached the vicinity of the global minimum faster than de, but that after reaching the vicinity of the global minimum it found difficulty in satisfying the stopping condition (23). It spent a considerable number of fe after reaching the vicinity of the global minimum for problems such as the Shekel family, Epistatic Michalewicz (EM), Hartman 6 (H6) and the Neumaier's problems (NF2 and NF3). de also exhibited a similar behaviour for some functions but to a much lesser extent. We aimed to remedy this behaviour of fde by the pfde algorithm that can use the complementary strengths of de and fde. Our next comparison shows that we have achieved this to a great extent.

We now compare de with pfde in Table II. We ran de and pfde (with $\tau = 0.25$) using a different stopping condition, namely the condition

$$\left|f_{\max} - f_{\min}\right| < 10^{-4}, \qquad (24)$$

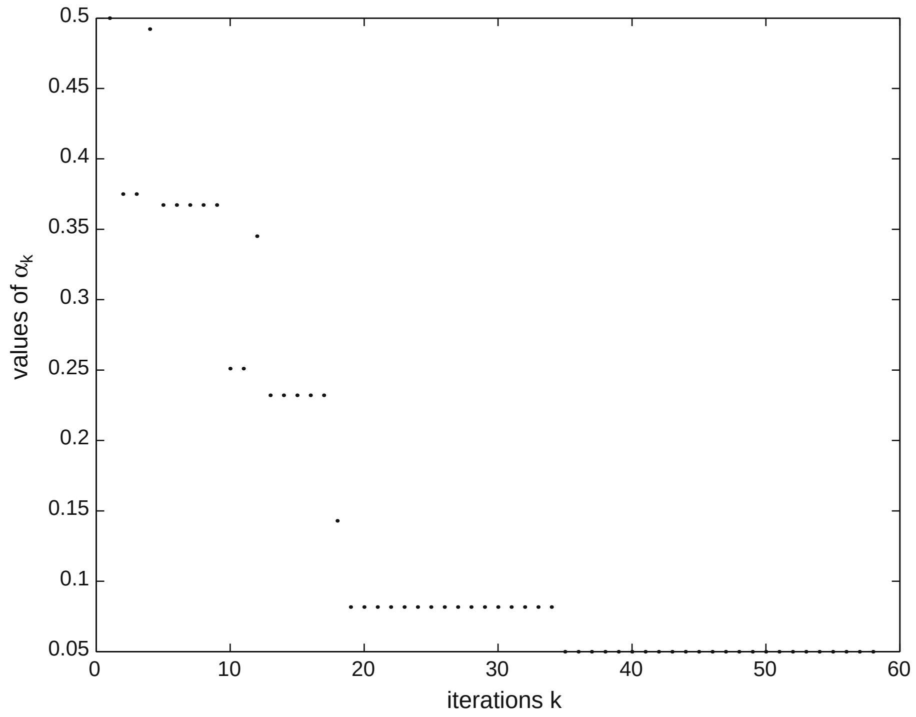

Figure 3. Adaptation of pfde on PP.

where $f_{\max}$ and $f_{\min}$ are defined in (20). The above stopping condition is often used for the population based global optimization methods [2]. We studied the effectiveness of (24) in obtaining a solution with a desired accuracy, e.g. at least three decimal digits of accuracy. Therefore, a success was counted if the solution obtained using (24) also satisfied (23). Results of these studies are presented in Table II, where the last row contains the total results. We have excluded the Price Transistor modeling function (PTM) along with OSP and ST from this table as they were not solved by any of the algorithms presented in Table II. The algorithms with superscript in Table II use (24) as the stopping condition. They are the same algorithm but they implement different stopping conditions. For example, pfde uses (23) as its stopping condition while pfde^1 uses (24).

We compare first the results of an algorithm obtained by implementing the stopping conditions (23) and (24). A comparison between de in Table I and de^1 in Table II shows that de^1 incurred higher fe and fo and obtained a lower sr. In particular, de is superior to de^1 by about 20%, 1% and 1% successful runs in terms of fe, fo and sr, respectively. A similar comparison between pfde and pfde^1 shows that pfde is superior to pfde^1 by about 19%, about 2% and 56 successful runs in terms of fe, fo and sr, respectively. It is

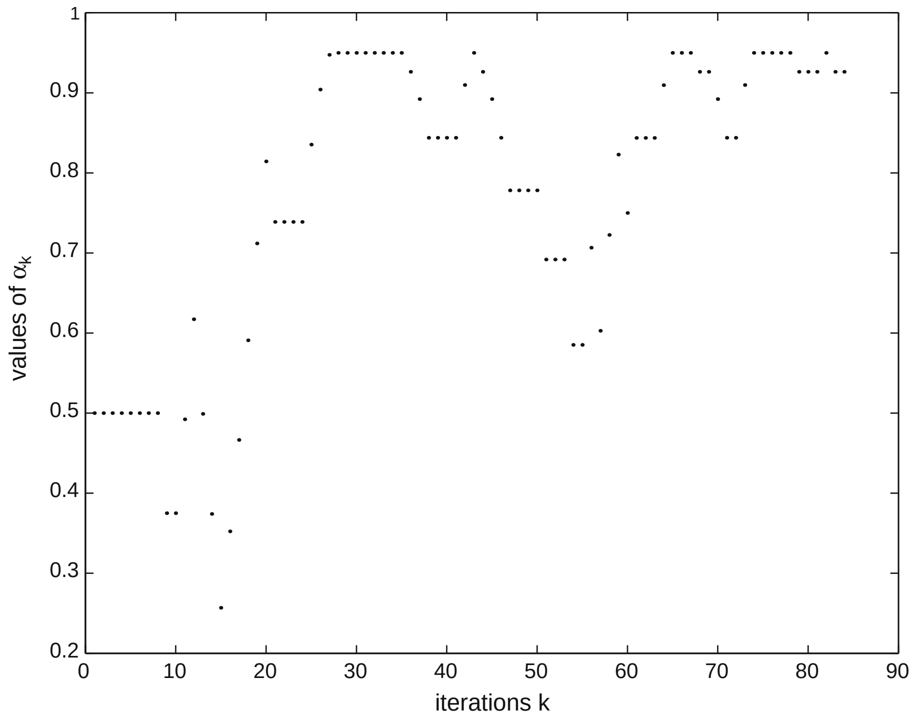

Figure 4. Adaptation of pfde S5.

therefore clear that the condition (24) worsens fe, fo and sr. A low sr indicates that the imposition of a stopping condition like (24) does not necessarily improve the quality of the final solution.

A comparison between de and pfde (or de[1] and pfde[1]) shows that the de algorithms are much inferior to the pfde algorithms in terms of fe and fo. For example, pfde is superior to de by about 44% and about 83% in terms of fe and fo, respectively. The total cpu required by de[1] and pfde[1] are 61.53 and 33.91, (excluding the cpu for SAL) respectively. This also shows that pfde is much superior to de in terms of cpu. Besides pfde was able to obtain the global minimum value for a difficult problem, namely the Salomon function (SAL), where de failed. In terms of sr however de outperforms pfde by 1.2% and de[1] outperforms pfde[1] by 1.8%.

We now show how the probabilistic adaptation takes place within pfde. We present four figures to illustrate these adaptations. The figures have been plotted using the number of iterations k on the horizontal axis and values of α_k on the vertical axis. For a particular problem, we observed slight variations in plots from run to run. However, the general trend is the same for all successful runs. Therefore, we present each figure from a single run. Figures 2–5 refer respectively to the functions Hartman 6 (H6),

Table II. Comparison of pfde and de using 47 test problems

	pfde			pfde[1]				de[1]			
$P(n)$	fe	sr	fo	fe	sr	fo	cpu	fe	sr	fo	cpu
ACK(10)	33,197	100	650	28,912	100	536	0.70	206,978	100	1,799	6.72
AP(2)	440	100	9	786	96	8	0.01	871	100	23	0.01
BL(2)	420	100	86	2,966	100	173	0.02	1,694	100	288	0.01
B1(2)	596	91	8	1,046	89	9	0.01	1,241	100	26	0.01
B2(2)	618	92	9	1,184	92	11	0.01	1,367	100	36	0.01
BR(2)	692	100	82	1,132	100	89	0.01	1,612	100	339	0.01
CB3(2)	327	100	8	744	99	10	0.01	915	100	24	0.01
CB6(2)	465	100	9	938	100	8	0.01	1,259	100	23	0.01
CM(2)	233	100	10	625	100	9	0.01	794	100	26	0.01
DA(2)	1,860	100	166	1,433	94	190	0.01	1,880	100	202	0.02
EP(2)	545	100	24	714	76	22	0.01	1,055	64	67	0.01
EM(10)	250,795	68	54,618	240,752	68	53,102	7.00	538,741	96	333,571	14.86
EXP(10)	4,052	100	503	11,764	100	759	0.02	16,366	100	1,352	0.03
GP(2)	546	100	16	912	100	16	0.01	1,207	100	44	0.01
GW(10)	38,223	98	515	51,857	98	599	0.98	264,494	100	1,376	4.77
GRP(3)	1,096	100	163	4,626	100	168	0.27	4,004	100	407	0.24
H3(3)	582	100	89	1,162	100	100	0.01	1,474	100	209	0.01
H6(6)	5,457	100	1,060	8,236	100	1,129	0.10	9,180	100	3,294	0.13
HV(3)	2,932	100	28	4,413	100	30	0.02	4,602	100	95	0.03
HSK(2)	218	100	14	610	100	14	0.01	744	100	37	0.01
KL(4)	123	100	38	2,058	100	314	0.02	2702	100	112	0.02
LM1(3)	716	100	38	1,518	100	46	0.01	1,860	100	102	0.02
LM2(10)	5,484	100	540	8354	100	896	0.07	21,258	100	1,902	0.31
MC(2)	376	100	24	664	100	29	0.01	834	100	74	0.01
MR(3)	210	100	38	2,692	100	129	0.02	3,857	100	134	0.02
MCP(4)	387	100	90	2,356	100	316	0.02	2,654	100	283	0.03
ML(5)	14,686	95	3,870	16,056	95	3,922	0.25	24,883	100	15,283	0.36
MRP(2)	868	69	9	1,457	67	9	0.01	1,883	52	17	0.01
MGP(2)	744	70	8	1,132	69	17	0.01	1,545	74	29	0.01
NF2(4)	58,618	100	4,001	94,437	100	4,167	0.82	149,692	96	29,364	1.17
NF3(10)	89,050	100	824	122,446	98	9,737	1.37	157,534	100	5,877	1.79
PP(10)	6,194	100	1,186	14,242	100	3,290	0.23	25,382	100	7,380	0.49
PRD(2)	1,260	88	241	2,183	88	280	0.01	2,241	94	757	0.01
PQ(4)	2,968	100	187	6,567	100	119	0.06	7,430	100	433	0.07
RG(10)	32,420	100	1,370	30,924	100	112	0.61	156,654	100	5,772	3.15
RB(10)	325,308	100	885	309,522	100	1,220	18.77	304,630	100	2,747	19.18
SAL(5)	27,214	16	68	140,150	11	136	1.33	0	0	0	0.00
SF1(2)	4,578	41	30	5,220	41	29	0.01	5,097	48	56	0.01
SF2(2)	2,178	100	8	2,453	97	9	0.02	2,518	100	18	0.02
SBT(2)	2,252	100	327	2,411	100	367	0.02	3,884	100	1,093	0.03
SWF(10)	17,786	100	5,323	18,048	100	5,607	0.82	50,090	100	53,847	1.66
S5(4)	4,512	89	224	6,362	89	274	0.05	7,431	100	850	0.06
S7(4)	3,844	90	159	5,404	90	158	0.04	6,423	100	438	0.05
S10(4)	3,424	96	121	5,041	96	155	0.06	6,373	100	431	0.06
FX(5)	13,880	13	2,254	15,520	13	2,312	0.20	17,906	9	6,279	0.26
SIN(20)	26,444	100	4,857	27,332	100	4,504	1.12	155,916	100	44,927	5.88
WP(4)	11,973	100	93	22,742	100	96	0.14	18,464	100	248	0.13
Total	**10,00,791**	**4,311**	**84,880**	**12,32,103**	**4,255**	**86,468**	**35.24**	**21,99,619**	**4,333**	**5,21,691**	**61.53**

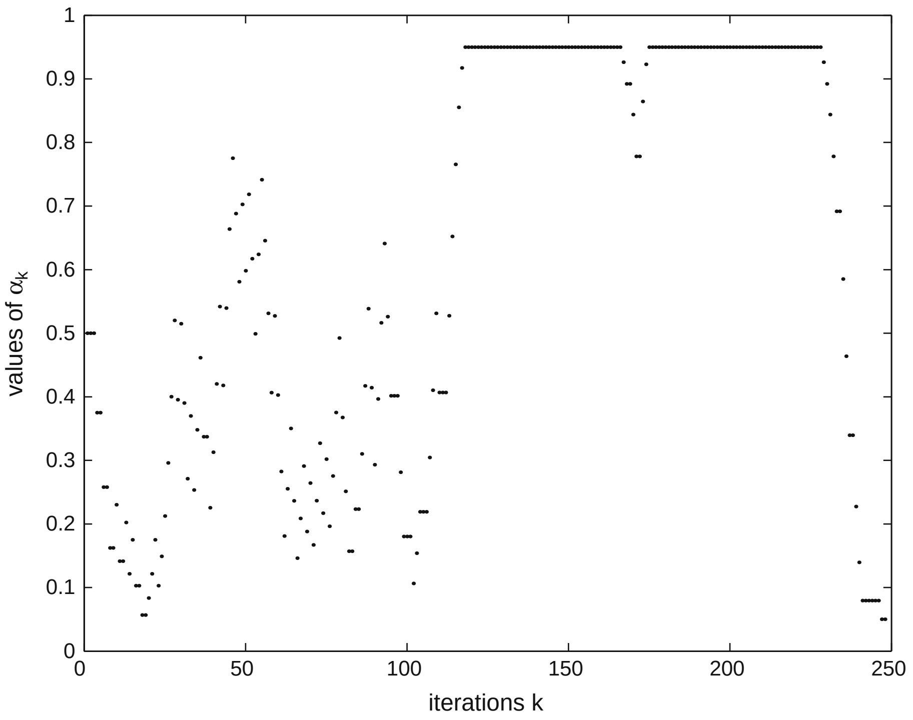

Figure 5. Adaptation of pfde on SWF.

Paviani (PP), Shekel 5 (S5) and Schwefel (SWF). The pfde algorithm uses M_μ for $\alpha_k = 1$ and M_β for $\alpha_k = 0$ and a mixture of the two for any $\alpha_k \in (0, 1)$. For the Hartman 6 problem (Figure 2), pfde tends to use more of M_β than M_μ for the first 30 iterations before almost switching completely to M_μ after about 35 iterations to solve the problem. For the Paviani problem (Figure 3), pfde quickly switches to M_β and consistently uses the scheme to solve this problem. As can be seen from Table II, this problem was very expensive for de in terms of fe. It needed about 4 times more fe than that pde needed. This clearly shows the dominance of the scheme M_β in pfde in solving this problem. For the Shekel 5 problem (Figure 4), for the first 10 iterations pfde uses both the schemes evenly. After about 15 iterations pfde switches to M_β and after about another 10 iterations it uses about an even mixture of the two schemes before switching to a majority of M_μ. For the Schwefel function (Figure 5), pfde favours M_β for the first 100 iterations before switching completely to M_μ and stays there until 230 iterations. At the end it switches back to M_β for a few iteration to solve the problem. The probabilistic adaptation of pfde can be shown using other problems as well but we have taken the above problems as representative.

7. Conclusion

We have developed and tested two versions of the de algorithm on a large set of problems. Numerical results have shown that the new versions are considerably better than their original counterpart in terms of the number of function evaluations. The new version pfde has more flexibility than the original de. In effect, we have generalized the de in that it is special cases of pfde. We have also shown how the probabilistic adaptation in pfde guides an algorithm to improve its robustness and efficiency in terms of fe and cpu times. The new methods are slightly less efficient in terms sr. However, this is compensated for by the large decrease in number of function evaluations and cpu times. The advantage of fde is that trial points no longer fall outside the search region while the number of points that fall outside the search region in pfde is much smaller than for de.

The direct search type methods such as the de methods have been designed to solve optimization problems that are non-differentiable and noisy, or they have no exactly known mathematical expressions. These types of problems arise naturally in many practical applications where the function values are dependent on simulation. These functions are very expensive to evaluate. Therefore the use of the new algorithms is totally justified as they require much less function evaluations than the original de.

Further research is underway in developing an efficient hybrid de global optimization method for large dimensional problems.

Acknowledgement

This paper was written when the first author participated in the year long optimization program at IMA, during the academic year 2002–2003. Financial support from IMA is acknowledged.

References

1. Storn, R. and Price, K. (1997) Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization* 11, 341–359.
2. Ali, M.M. and Törn, A. (2004) Population set based global optimization algorithms: some modifications and numerical studies. *Computers and Operations Research* 31, 1703–1725.
3. Price, K. (1999) An introduction to differential evolution. In: Corne, D. Dorigo, M. and Glover, F. (eds.), *New Ideas in Optimization*. pp. 79–108, McGraw-Hill, London.
4. Zaharie, D. (2002) Critical values for the control parameters of differential evolution algorithms. In: Matousek, R. and Osmera, P. (eds.). *Proceedings of MENDEL 2002, 8th International Mendel Conference on Soft Computing, Bruno, Czech Republic*. pp. 62–67, Bruno University of Technology, Faculty of Mechanical Engineering, Bruno.

5. Ali, M.M. and Törn, A. (2002) Topographical differential evolution using pre-calculated differential. In: Dzemyda, G. Saltenis, V. and Zilinskas, A. (eds.). *Stochastic and Global Optimization*, pp. 1–17, Kluwer Academic Publisher.
6. Hogg, R.V. and Tanis, E.A. (1997) *Probability and Statistical Inference*, 5th edition.
7. Cheng, R.C.H. (1978) Generating Beta Variates with Non-Integral Shape Parameters, Vol. 21, pp. 317–322, Communications of the ACM.
8. Najim, K. Pibouleau, L. and Le Lann, M.V. (1990) Optimization technique based on learning automata. *Journal of Optimization Theory and Applications* 64, 331–347.
9. Ali, M.M. Khompatraporn, C. and Zabinsky, Z.B. (2005) A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *Journal of Global Optimization* 31, 635–672.
10. Kaelo, P. (2005) Some Population Set Based Methods for Unconstrained and Constrained Global Optimization. PhD thesis, Witwatersrand University.