

https://dspace.org/

Some Population Set-Based Methods for Unconstrained Global Optimization

Kaelo, Professor

2006-11-16T08:40:11Z
http://hdl.handle.net/10539/1771

Downloaded from a DSpace repository. https://dspace.org/

Differential evolution algorithms using hybrid mutation

P. Kaelo and M. M. Ali
School of Computational and Applied Mathematics,
Witwatersrand University, Wits - 2050, Johannesburg
South Africa

Abstract

Differential evolution [1] has gained a lot of attention from the global optimization research community. It has proved to be a very robust algorithm for solving non-differentiable and non-convex global optimization problems. In this paper, we propose some modifications to the original algorithm. Specifically, we use the attraction-repulsion concept of electromagnetism-like algorithm [2, 3] to boost the mutation operation of the original differential evolution. We carried out a numerical study using a set of 50 test problems, many of which are inspired by practical applications. Results presented show the potential of this new approach.

Keywords: Global optimization, mutation, differential evolution, electromagnetism-like algorithm, attraction-repulsion, standard deviation.

1 Introduction

Unconstrained global optimization problems (without loss of generality we consider only the minimization problems) can be represented by the following: Given a real valued objective function $f(x)$ defined on the set $x \in \Omega \subset \mathbb{R}^n$ (x a continuous variable), find the point x^* and the corresponding value f^* such that

$$f^* = f(x^*) = \min \{f(x) | x \in \Omega\}. \tag{1}$$

The domain Ω is assumed to be either a box or some other region easy to sample.

Global optimization problems are frequently encountered in many areas such as in engineering design, applied sciences, molecular biology and other scientific applications. In global optimization it is required to locate the global minimum among many local minima. In many cases, global optimization problems are non-differentiable, noisy and simulation based. Hence the gradient based methods cannot be used for finding the global minimum of such problems. As a result, many researchers have devoted themselves in finding some reliable stochastic

global optimization methods that do not require any computation of the gradients of the objective function. These methods evaluate the objective function in a random sample of points from the feasible search domain Ω and subsequently manipulate the sample. Thus, they are adaptable to a wide range of problems. They include, amongst others, differential evolution [1, 4], electromagnetism-like algorithm [2, 3], genetic algorithms [5, 6] and controlled random search [7, 8].

In this paper, we are mainly concerned with the differential evolution algorithm [1] (henceforth denoted by de). The de algorithm is a population set based direct search global optimization algorithm [6] and is purely heuristic. Like all population set based direct search algorithms, de uses a population set S. The initial set S consists of N random points, called vectors, in Ω. A contraction process is then used to drive these vectors to the vicinity of the global minimizer. The contraction process involves replacing bad vector(s) in S with better vector(s), per iteration.

The de algorithm has been applied to a number of test and practical problems by a number of researchers, e.g. Storn and Price [1], Lampinen and Zelinka [9], Storn [10] and Babu and Sastry [11], and it has proved to be a robust algorithm in terms of finding the global minimum. Notwithstanding its popularity, the de algorithm has a number of drawbacks. One of its drawbacks is the presence of problem dependent parameters. A further drawback of de is that it appears to be inefficient in terms of its convergence rate as the region of the global minimum is approached [4, 12]. Therefore, the strength of de is that it is robust in locating the global minimum and the weakness of de is that after reaching the vicinity of the global minimum, it takes many iterations to converge. This suggests that the point generation scheme of de is very exploratory at initial stages of the search of Ω but slows down the convergence when it approaches the region of the global minimum. The objective of this paper is to investigate the efficiency (in terms of the number of function evaluations and cpu time) and robustness (in terms of locating the global minimum) of the de algorithm using a large set of 50 test problems and to suggest some modifications to improve its performance. Central to our modifications is the use of the attraction-repulsion technique in the mutation of de. This technique forms part of the electromagnetism-like (em) algorithm [2]. We show that the incorporation of the attraction-repulsion technique in de helps in facilitating the convergence by creating more fit vectors especially when the region of the global minimum is approached.

The organization of the paper is as follows. In section 2 we briefly describe de and em algorithm. Section 3 contains the full description of our proposed modifications. In section 4 numerical results and comparisons are made. Section 5 contains the concluding remarks based on the results obtained.

2 A brief description of de and em

In this section, we briefly describe the de algorithm and the em algorithm.

2.1 Differential evolution (de) algorithm

The de algorithm utilizes N, n-dimensional parameter vectors $x_{i,k}$, $i = 1, 2, \cdots, N$, as a population to search the feasible region Ω. The index k denotes the iteration (or generation) number of the algorithm. The initial population,

$$S = \{x_{1,0}, x_{2,0}, \cdots, x_{N,0}\}, \tag{2}$$

where $k = 0$, is taken to be uniformly distributed in the search region. At each iteration, all vectors in S are targeted for replacement. Therefore, N competitions are held to determine the members of S for the next iterations. This is achieved by using the mutation, crossover and selection operators.

In the mutation phase, for each target vector $x_{i,k}$, $i = 1, 2, \cdots, N$, a mutant vector $\hat{x}_{i,k}$ is obtained by

$$\hat{x}_{i,k} = x_{\alpha,k} + F(x_{\beta,k} - x_{\gamma,k}) , \tag{3}$$

where $\alpha, \beta, \gamma \in \{1, 2, \cdots, N\}$ are mutually distinct random indices and are also different from the current target index i. The vector $x_{\alpha,k}$ is known as the base vector and $F > 0$ is a scaling parameter. If the mutant vector $\hat{x}_{i,k} \notin \Omega$ then the mutation operation is repeated. We denote the mutation (3) by M_d. The crossover operator is then applied to obtain the trial vector $y_{i,k}$ from $\hat{x}_{i,k}$ and $x_{i,k}$. The crossover is defined by

$$y_{i,k}^{j} = \begin{cases} \hat{x}_{i,k}^{j} & \text{if } R^j \leq C_R \text{ or } j = I_i \\ x_{i,k}^{j} & \text{if } R^j > C_R \text{ and } j \neq I_i , \end{cases} \tag{4}$$

where I_i is a randomly chosen integer in the set I, i.e., $I_i \in I = \{1, 2, \cdots, n\}$; the superscript j represents the j-th component of respective vectors; $R^j \in (0, 1)$, drawn randomly for each j. The ultimate aim of the crossover rule (4) is to obtain the trial vector $y_{i,k}$ with components coming from the components of the target vector $x_{i,k}$ and the mutated vector $\hat{x}_{i,k}$. And this is ensured by introducing C_R and the set I. Notice that for $C_R = 1$ the trial vector $y_{i,k}$ is the replica of the mutated vector $\hat{x}_{i,k}$. The targeting process (mutation and crossover) continues until all members of S are considered. After all N trial vectors $y_{i,k}$ have been generated, acceptance is applied. In the acceptance phase, the function value at the trial vector, $f(y_{i,k})$, is compared to $f(x_{i,k})$, the value at the target vector and the target vector is updated using

$$x_{i,k+1} = \begin{cases} y_{i,k} & \text{if } f(y_{i,k}) < f(x_{i,k}) \\ x_{i,k} & \text{otherwise.} \end{cases} \tag{5}$$

Reproduction (mutation and crossover) and acceptance continues until some stopping conditions are met.

2.2 Electromagnetism-like (em) algorithm

Electromagnetism-like (em) algorithm [2, 3] is a recently proposed population set based algorithm. The algorithm applies an attraction-repulsion technique (Coulomb's Law) to move the

sample points to the vicinity of the global minimizer. Points with better objective function values attract others while those with inferior function values repel. The general electromagnetism-like algorithm is given as:

Algorithm 1: the pseudo em algorithm

Step 1 Initialize the population set S and the iteration counter k

Step 2 **while** not maximum iteration **do**

Step 3 Local search

Step 4 CalcF

Step 5 MoveF

The first step of the algorithm gives the N initial random sample points $x_{i,0}, i = 1, 2, \cdots, N$. In Step 3, a local search is applied to the sample points to explore the neighbourhood of the points in the sample. The procedure CalcF then finds the total force exerted on each point, at each iteration k, by all other points in the population. Here, a force exerted on a point $x_{i,k}$ by $x_{j,k}$ is calculated based on a charge, $q_{i,k}$, on $x_{i,k}$ and a charge, $q_{j,k}$, on $x_{j,k}$. The charge of each point $x_{i,k}$ is determined by its objective function value relative to the objective function value of the current best point $x_{b,k}$ in the population, i.e.

$$q_{i,k} = \exp\left[-n\frac{f(x_{i,k}) - f(x_{b,k})}{\sum_{l=1}^{N}[f(x_{l,k}) - f(x_{b,k})]}\right], i = 1, 2, \cdots, N, \tag{6}$$

where n is the dimension of the problem and N is the size of the population. The n-dimensional force vector $F_{ij,k}$ exerted on $x_{i,k}$ by $x_{j,k}$ is then determined by

$$F_{ij,k} = \begin{cases} (x_{j,k} - x_{i,k})\frac{q_{i,k}q_{j,k}}{||x_{j,k}-x_{i,k}||^2} & \text{if } f(x_{j,k}) < f(x_{i,k}) \\ (x_{i,k} - x_{j,k})\frac{q_{i,k}q_{j,k}}{||x_{j,k}-x_{i,k}||^2} & \text{if } f(x_{i,k}) \leq f(x_{j,k}). \end{cases} \tag{7}$$

It follows from (7) that the direction of the force $F_{ij,k}$ is decided by comparing the objective function values of $x_{i,k}$ and $x_{j,k}$. If the objective function value of $x_{j,k}$ is better than that of $x_{i,k}$ then $x_{j,k}$ attracts $x_{i,k}$, otherwise it repels $x_{i,k}$. Therefore, points with relatively good objective function values will attract the other points in the population while points with worse objective function values repel the others. Subsequently, the resultant force vector $F_{i,k}$ exerted on a point $x_{i,k}$ by all other points in the population is determined as a vector summation of individual component vector forces, i.e.

$$F_{i,k} = \sum_{j=1, j\neq i}^{N} F_{ij,k}, i = 1, 2, \cdots, N. \tag{8}$$

As an example to show the exertion of forces, we consider a force exerted on a point x_1 by two other points x_2 and x_3. We present the forces in Figure 1 where F_{12} is a repulsive force and F_{13} is an attractive force exerted on x_1 by x_2 and x_3 respectively. Thus, the objective

function value of x_2 is worse than that of x_1 and the objective function value of x_3 is better than that of x_1. The resultant force F_1 exerted on x_1 is given by

$$F_1 = F_{12} + F_{13}. \tag{9}$$

The resultant forces exerted on x_2 by x_1 and x_3, and on x_3 by x_1 and x_2 can be calculated in a similar way.

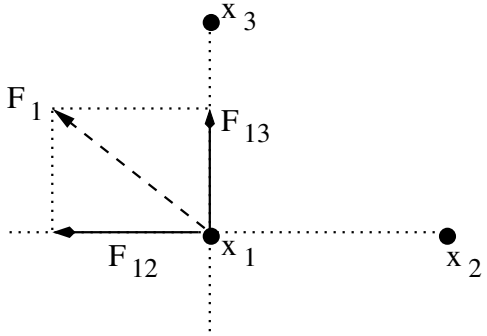

Figure 1: Exertion of forces on x_1 by x_2 and x_3

After the calculation of the force vector $F_{i,k}$ in (8) ($i = 1, 2, \cdots, N$), the point $x_{i,k}$ ($i = 1, 2, \cdots, N$) is moved to

$$x_{i,k+1} = x_{i,k} + \mu F_{i,k}, \tag{10}$$

using the procedure MoveF, in the direction of the force $F_{i,k}$, to give a new population of solutions; μ is a dimension dependent value. Here, the force vector $F_{i,k}$ only identifies the direction of movement but not the magnitude. The magnitude of the move of each dimension of $x_{i,k}$ is determined by a value randomly selected from $[0, \nu_j]$, where ν_j is the allowed feasible movement towards the upper bound or lower bound of the j-th dimension depending on whether the j-th component of the resultant force, $F^j_{i,k}$, is positive or negative [2, 3].

In the original em algorithm, all points in the population exert a force on all other points, i.e. in a population of N points, each point will have $N - 1$ points exerting force on it. In a recent hybrid scatter search / electromagnetism (hsse) algorithm by Debels et. al. [13], an approach is adopted whereby one point acts on any given point. In this approach, a charge on a point is calculated based on the point that exerts force on it. Thus, if $x_{j,k}$ exerts a force on $x_{i,k}$ then a charge $q_{ij,k}$ is computed based on the relative difference in the objective function values $f(x_{i,k})$ and $f(x_{j,k})$, and is given as

$$q_{ij,k} = \frac{f(x_{i,k}) - f(x_{j,k})}{f(x_{w,k}) - f(x_{b,k})}, \tag{11}$$

where $x_{w,k}$ and $x_{b,k}$ are the worst and the best solutions, respectively, in the population set. Better points $x_{j,k}$ give higher values of $q_{ij,k}$. Notice that unlike the positive point charge $q_{i,k}$ in em, the charge $q_{ij,k}$ in hsse is dependent on $x_{j,k}$ and $q_{ij,k} \in [-1, 1]$. The charge $q_{ij,k}$ is positive

if $f(x_{i,k}) > f(x_{j,k})$ and negative otherwise. Hence $q_{ij,k}$ is positive when $x_{j,k}$ attracts $x_{i,k}$ and negative when $x_{j,k}$ repels $x_{i,k}$. There is no charge when $f(x_{i,k}) = f(x_{j,k})$, i.e. $q_{ij,k} = 0$. The force exerted on $x_{i,k}$ by $x_{j,k}$ is then computed as:

$$F_{ij,k} = (x_{j,k} - x_{i,k})q_{ij,k}, \tag{12}$$

where $x_{j,k}$ is selected randomly from the remaining $N - 1$ points. After the computation of $F_{ij,k}$, the point $x_{i,k}$ is then moved to $x_{i,k+1}$ by

$$x_{i,k+1} = x_{i,k} + F_{ij,k}. \tag{13}$$

Thus, unlike em, which uses the resultant force $F_{i,k}$ of $N - 1$ points in moving $x_{i,k}$ to $x_{i,k+1}$, hsse uses only one force, $F_{ij,k}$, in moving $x_{i,k}$ to $x_{i,k+1}$. We incorporate the force $F_{ij,k}$, based on the charge $q_{ij,k}$, in the mutation of de, although our implementation is different.

3 The proposed algorithms

In this section, we describe two new versions of de. Our modifications to the de algorithm are based on (i) parameter value selection and (ii) changes to the mutation rule. We will now motivate why these changes have been made. We begin with the parameters of de. The original de algorithm calculates a trial point using (3) and (4). The mutation M_d defined by (3) has a parameter F and the crossover (4) has a parameter C_R. The values of these parameters have to be provided by the user at the beginning of the de algorithm. There are no optimal values of F and C_R as the optimal values are problem dependent. Various values for F and C_R have been suggested in the literature [1, 12, 14] by empirical means, and in most cases values of F and C_R varies in $[0.4, 1]$ and $[0.4, 0.9]$, respectively. These values differ due to the variations of the test problem set used in the empirical testing. Therefore, our selection of these values are random in their respective ranges. In particular, we make F a uniform random variable in $[0.4, 1]$ for each target point $x_{i,k}$ $(i = 1, 2, \cdots, N)$ in the k-th iteration and C_R a uniform random variable in $[0.5, 0.7]$ per iteration. Our numerical experiments have shown that these changes alone slightly improve the performance of de, especially in terms of the number of function evaluations [15].

Our second modification to de lies in calculating the mutant vector $\hat{x}_{i,k}$. In particular, we use a mixture of (3) and a variant of electromagnetism - like scheme in creating $\hat{x}_{i,k}$. We refer to this variant scheme as M_e. M_e uses three distinct random vectors, $x_{\alpha,k}$, $x_{\beta,k}$ and $x_{\gamma,k}$ from S, just as in M_d, in creating $\hat{x}_{i,k}$. However, unlike M_d, it does not exclude the target vector $x_{i,k}$ in selecting these three vectors. Operations involved in M_e in creating $\hat{x}_{i,k}$ are different from those in em and hsse. In particular, em uses the resultant force $F_{i,k}$ of $N - 1$ individual forces exerted on the target vector $x_{i,k}$ in moving it to another location (i.e. creating $x_{i,k+1}$); hsse uses the single force $F_{ij,k}$ exerted on $x_{i,k}$ in creating $x_{i,k+1}$ for the next iteration. Operations involved in M_e are intermediate of the above two, i.e. between (8) and (12). M_e uses the resultant force $F_{\alpha,k}$ of two individual forces $F_{\alpha\beta,k}$ and $F_{\alpha\gamma,k}$ exerted on $x_{\alpha,k}$ by $x_{\beta,k}$ and $x_{\gamma,k}$,

respectively. The forces $F_{\alpha\beta,k}$ and $F_{\alpha\gamma,k}$ are calculated as in (11) - (12). The mutant vector $\hat{x}_{i,k}$ is then given by

$$\hat{x}_{i,k} = x_{\alpha,k} + F_{\alpha,k}. \tag{14}$$

M_e, given by (14), differs from M_d in calculating $\hat{x}_{i,k}$ in that:

- the goodness of the points $x_{\alpha,k}$, $x_{\beta,k}$ and $x_{\gamma,k}$ is considered in M_e when $x_{\alpha,k}$ is moved to create $\hat{x}_{i,k}$. M_d simply adds the scaled differential vector $F(x_{\beta,k} - x_{\gamma,k})$ to $x_{\alpha,k}$ in creating $\hat{x}_{i,k}$.
- M_e is parameter free while M_d is not.
- when the points in S clusters around the global minimizer, M_e induces increasingly more local effects than M_d, resulting in faster convergence.

Under these circumstances, the use of (14) is fully justified.

The modifications proposed in this section result in two new versions of de. They differ only on how the M_e operation is invoked. In the first modification both M_d and M_e are used per iteration to generate the mutant vectors. We refer to this new de algorithm as the de algorithm with integrated mutation per iteration and denote it by deim. In the second modification either M_d or M_e is used per iteration to generate the mutant vectors. This version of de is referred to as the de algorithm with adapted mutation per iteration and denoted by deam.

3.1 A de algorithm with integrated mutation per iteration

The deim algorithm is similar to the original de except that it invokes both M_d and M_e per iteration. It uses the M_d operation more frequently than M_e at earlier stages and vice-versa at later stages. At later stages, frequent use of M_e gives rapid convergence. When the vector $x_{i,k}$ ($i = 1, 2, \cdots, N$) is targeted at the k-th iteration of deim, either M_d or M_e is used to create $\hat{x}_{i,k}$ followed by crossover rule (4) in generating the trial vector $y_{i,k}$. The criterion which determines whether to use M_d or M_e is based on component-wise standard deviation of vectors in S. We calculate the standard deviation σ_k^j of the j-th components of points in S at the k-th iteration. Intuitively, as the iterations proceed, $||\sigma_k|| \to 0$, where $\sigma_k = (\sigma_k^1, \sigma_k^2, \cdots, \sigma_k^n)$. For each target vector $x_{i,k}$ ($i = 1, 2, \cdots, N$), we randomly select a component j from the set $\{1, 2, \cdots, n\}$ and compare σ_k^j with σ_0^j. If σ_k^j is less than a certain percentage of σ_0^j then we invoke M_e in calculating $\hat{x}_{i,k}$, otherwise we invoke M_d. In particular, if $\sigma_k^j < \epsilon_1 \sigma_0^j$ then we use M_e, where ϵ_1 is a parameter which controls the frequency of invoking M_e. Notice that σ_k^j ($j = 1, 2, \cdots, n$) are calculated at the beginning of each iteration. Below we present the algorithm for deim.

Algorithm 2: the deim algorithm

Step 1 Determine the initial set: Set iteration counter $k = 0$.

$$S = \{x_{1,k}, x_{2,k}, \cdots, x_{N,k}\}$$

where the vectors $x_{i,k}$, $i = 1, 2, \cdots, N$, are sampled randomly in Ω; evaluate $f(x_{i,k})$ at each $x_{i,k}$, $i = 1, 2, \cdots, N$. Take $N \gg n$, n being the dimension of the function $f(x_{i,k})$.

Step 2 Determine best and worst vectors in S: Determine the vectors $x_{h,k}$ and $x_{l,k}$, where $x_{h,k}$ corresponds to the worst vector and $x_{l,k}$ corresponds to the best vector in S. If the stopping condition such as

$$|f(x_{h,k}) - f(x_{l,k})| \leq \epsilon \tag{15}$$

is satisfied, then stop.

Step 3 Generate vectors to replace vectors in S: Randomly choose C_R from $[0.5, 0.7]$. Evaluate σ_k^j, $j = 1, 2, \cdots, n$. For each $x_{i,k} \in S$ ($i = 1, 2, \cdots, N$), choose j randomly from $\{1, 2, \cdots, n\}$. If $\sigma_k^j \geq \epsilon_1 \sigma_0^j$ then go to Step 3a else go to Step 3b.

Step 3a Mutation operation M_d: Randomly choose three distinct vectors $x_{\alpha,k}$, $x_{\beta,k}$ and $x_{\gamma,k}$ from S, all different from the target vector $x_{i,k}$. Find the mutant vector $\hat{x}_{i,k}$ using

$$\hat{x}_{i,k} = x_{\alpha,k} + F\left(x_{\beta,k} - x_{\gamma,k}\right), \tag{16}$$

where $F \in [0.4, 1]$ is chosen randomly. If $\hat{x}_{i,k} \notin \Omega$ repeat Step 3a. Go to Step 3c.

Step 3b Mutation operation M_e: Randomly choose three vectors $x_{\alpha,k}$, $x_{\beta,k}$ and $x_{\gamma,k}$ from S. Find the mutant vector $\hat{x}_{i,k}$ using

$$\hat{x}_{i,k} = x_{\alpha,k} + F_{\alpha,k}, \tag{17}$$

where $F_{\alpha,k}$ is the resultant force of $F_{\alpha\beta,k}$ and $F_{\alpha\gamma,k}$ on $x_{\alpha,k}$. $F_{\alpha\beta,k}$ and $F_{\alpha\gamma,k}$ are calculated using (12). If $\hat{x}_{i,k} \notin \Omega$ repeat Step 3b. Go to Step 3c.

Step 3c Crossover: Calculate the trial vector $y_{i,k}$ corresponding to the target vector $x_{i,k}$ from $x_{i,k}$ and $\hat{x}_{i,k}$ using the crossover rule (4).

Step 4 Acceptance rule to replace vectors in S: Update the vectors in S for the next iteration using the acceptance criterion :

$$x_{i,k+1} = \begin{cases} y_{i,k} & \text{if } f(y_{i,k}) < f(x_{i,k}) \\ x_{i,k} & \text{otherwise.} \end{cases} \tag{18}$$

Set $k := k + 1$ and go to Step 2.

Remarks

1. A suggested value for N is $10n$, where n is the dimension (see [15], for a full discussion on the choice of N).
2. The vectors x_h and x_l and their function values $f(x_h)$ and $f(x_l)$, respectively, are such that
$$f(x_h) = \max_{x \in S} f(x) \quad \text{and} \quad f(x_l) = \min_{x \in S} f(x).$$
3. The mutated vector $\hat{x}_{i,k}$ obtained by M_e is more local to the base vector when the points form a cluster in the vicinity of a minimizer. However, during the early stages when the vectors in S are scattered around the search region Ω, M_e gives vectors that are far from the base vector (exploratory). Therefore, at the early stages the search will be more global and at the later stages it is more local.
4. ϵ_1 in Step 3 is a user provided value in $(0, 1)$. ϵ_1 controls the frequency of M_e.

3.2 A de algorithm with adapted mutation per iteration

Unlike deim, deam uses only one mutation operation per iteration. That is, either M_d or M_e is invoked at the beginning of each iteration and used throughout the iteration. The invoking process is based on the standard deviation as in deim, but since we invoke one mutation operation per iteration, we compare $||\sigma_k||$ with $||\sigma_0||$. If $||\sigma_k|| < \epsilon_2||\sigma_0||$ then M_e is invoked, otherwise M_d is invoked in creating $\hat{x}_{i,k}$ corresponding to the target vector $x_{i,k}$ ($i = 1, 2, \cdots, N$). The parameter ϵ_2 is a switching parameter. Crossover follows the mutation in creating the trial vector $y_{i,k}$ corresponding to the target vector $x_{i,k}$. The deam algorithm is presented below.

Algorithm 3: the deam algorithm

Step 1 Determine the initial set S : same as in Algorithm 2.

Step 2 Determine best and worst vectors in S : same as in Algorithm 2.

Step 3 Generate points to replace points in S: Randomly choose C_R from $[0.5, 0.7]$. Evaluate $\sigma_k = (\sigma_k^1, \sigma_k^2, \cdots, \sigma_k^n)$. If $||\sigma_k|| \geq \epsilon_2||\sigma_0||$ then go to Step 3a else go to Step 3b.

Step 3a Mutation operation M_d: For each $x_{i,k} \in S$ ($i = 1, 2, \cdots, N$), find the mutant vector $\hat{x}_{i,k}$ using

$$\hat{x}_{i,k} = x_{\alpha,k} + F\left(x_{\beta,k} - x_{\gamma,k}\right), \tag{19}$$

where $F \in [0.4, 1]$ is chosen randomly. The vectors $x_{\alpha,k}$, $x_{\beta,k}$ and $x_{\gamma,k}$ are randomly chosen from S, all distinct and different from the target vector $x_{i,k}$. If $\hat{x}_{i,k} \notin \Omega$ repeat Step 3a for the i-th mutant vector. Go to Step 3c.

Step 3b Mutation operation M_e: For each $x_{i,k} \in S$ ($i = 1, 2, \cdots, N$), find the mutant vector $\hat{x}_{i,k}$ using

$$\hat{x}_{i,k} = x_{\alpha,k} + F_{\alpha,k}, \tag{20}$$

where $F_{\alpha,k}$ is the resultant force of $F_{\alpha\beta,k}$ and $F_{\alpha\gamma,k}$ on $x_{\alpha,k}$. $F_{\alpha\beta,k}$ and $F_{\alpha\gamma,k}$ are calculated using (12). The three vectors $x_{\alpha,k}$, $x_{\beta,k}$ and $x_{\gamma,k}$ are randomly chosen from S. If $\hat{x}_{i,k} \notin \Omega$ repeat Step 3b for the i-th mutant vector. Go to Step 3c.

Step 3c Crossover: For each i, $i = 1, 2, \cdots, N$, calculate the trial vector $y_{i,k}$, corresponding to the target vector $x_{i,k}$ from $x_{i,k}$ and $\hat{x}_{i,k}$ using the crossover rule (4).

Step 4 Acceptance rule to replace vectors in S. Same as in Algorithm 2.

Remark

5. The switching parameter ϵ_2 in Step 3 is a user provided value in $(0, 1)$.

4 Numerical results

Performance of the new algorithms described thus far was judged using a collection of 50 test problems. These problems were taken from the literature. They range from 2 to 20 in dimension and have a variety of inherent difficulty [16]. All the problems have continuous variables. A detailed description of each test problem (P) in the collection can be found in

[17, 18]. We use 45 problems from [17] and one (see Zak) problem from [18]. The number of problems, therefore, is 46, with several variations of some problem dimensions (see ACK, EXP,SBT3 and SIN) as listed in Table 1, bringing the total number up to 50. The purpose of this section is to determine if the changes made to de were able to answer our research questions stated at the end of the introduction section. In particular, if M_e was able to hasten the convergence rate of de without compromising its robustness.

The algorithms were run a hundred times on each of the test problems. There were 5000 runs in total. We use average results for comparison. Averages were taken on the number of successful runs for each problem. We use success rate, sr, average number of function evaluations, fe, and average cpu times, cpu, as the criteria for comparison. A success was counted when the value $f(x_{l,k})$ of a run was such that $|f_{opt} - f(x_{l,k})| \leq 0.01$, where f_{opt} is the known global minimum of the problem being solved.

4.1 Parameter selection

The optimal parameter values for the algorithms were found empirically using 50 test problems. We do not claim these values to be the optimal for any problem in general but they will be good values to choose. The common parameters of the algorithms are the scaling parameter F in their mutation rule (3) and controlling parameter C_R in their crossover rule (4). For de, the best results obtained using 50 problems were for $F = 0.5$ and $C_R = 0.5$ [4, 15]. As for the other two algorithms, F was randomly chosen from $[0.4, 1]$ per target vector $x_{i,k}$, $i = 1, 2, \cdots, N$, and C_R was randomly chosen from $[0.5, 0.7]$ per iteration [15]. Other common parameters to all algorithms are the population size N, the maximum iteration number T and the stopping rule parameter ϵ. We took the value of N to be $10n$ [6], where n is the dimension of the problem. In every case, a run was terminated when either the maximum number of iterations $T = 10000$ was reached or the objective function values of all vectors in S were identical to four decimal places, i.e. $\epsilon = 10^{-4}$ (see (15) in Algorithm 2).

Other parameters used in the new versions of de are the parameters ϵ_1 and ϵ_2 which control the use of M_e in deim and deam, respectively. We carried out numerical experiments using various values for these parameters in both algorithms. We observed, from these numerical experiments, that values of $\epsilon_1, \epsilon_2 \in (0, 0.4]$, have a positive effect of reducing both fe and cpu of de without compromising its robustness in terms of sr. For values of $\epsilon_1, \epsilon_2 > 0.4$, fe and cpu of de reduce greatly. However, the success rate, sr, of de reduces. Notice that the use of M_e in both algorithms depends on the rate at which the population set shrinks. Hence the adaptation of M_e depends from problem to problem. In the next subsection we present the best results obtained using $\epsilon_1 = 0.35$ for deim and $\epsilon_2 = 0.4$ for deam.

4.2 Comparisons and discussions

In this subsection, we compare the numerical results of all the algorithms and try to answer our research question. In particular, we answer the following question: How does the em mutation operation, M_e, affect the performance of de. Full results obtained by all algorithms are presented in Table 1, where tr stands for total results (or total of averages). The first and

Table 1: Comparison of de, deim and deam using 50 test problems

		de			deim			deam		
P	n	fe	sr	cpu	fe	sr	cpu	fe	sr	cpu
ACK	10	25810	100	0.56	13654	100	0.43	13747	100	0.42
–	20	104476	100	4.10	47618	100	5.48	48222	100	2.97
AP	2	683	100	0.01	579	100	0.01	567	100	0.01
BL	2	991	100	0.01	1019	100	0.02	1000	100	0.01
B1	2	977	100	0.01	776	98	0.01	789	100	0.01
B2	2	1039	100	0.01	811	100	0.01	803	100	0.01
BR	2	1030	100	0.01	1041	100	0.02	992	100	0.01
CB3	2	717	100	0.01	574	100	0.01	587	100	0.01
CB6	2	976	100	0.01	643	100	0.01	650	100	0.01
CM	4	2300	100	0.03	1731	100	0.03	1737	100	0.03
DA	2	1277	100	0.01	1040	100	0.01	1038	100	0.01
EP	2	650	95	0.01	651	100	0.01	687	100	0.01
EXP	10	9903	100	0.20	6316	100	0.24	6310	100	0.23
–	20	41698	100	1.58	22986	100	4.15	23278	100	3.96
GP	2	937	100	0.01	784	100	0.01	778	100	0.01
GW	10	14705	100	0.32	8857	100	0.31	8804	100	0.30
GRP	3	3024	98	0.25	2129	100	0.18	2189	100	0.20
H3	3	1220	100	0.01	998	100	0.02	994	100	0.01
H6	6	6836	99	0.13	4818	97	0.14	4753	100	0.13
HV	3	3782	99	0.04	2730	100	0.03	2737	100	0.03
HSK	2	559	100	0.01	479	100	0.01	455	100	0.01
KL	4	1839	100	0.02	1574	100	0.03	1575	100	0.03
LM1	3	1465	100	0.02	1185	100	0.02	1210	100	0.02
LM2	10	11583	100	0.26	7584	100	0.29	7619	100	0.28
MC	2	639	100	0.01	546	100	0.01	517	100	0.01
MR	3	3270	74	0.03	1744	41	0.03	1957	83	0.03
MCP	4	3672	100	0.05	1528	100	0.03	1522	100	0.03
ML	10	200673	70	4.95	48681	63	2.13	48358	55	2.17
MRP	2	1485	64	0.01	895	56	0.01	896	62	0.01
MGP	2	1032	68	0.01	982	75	0.01	974	65	0.01
NF2	4	80475	100	0.93	70547	100	1.66	61890	100	1.39
NF3	10	84620	100	1.67	97112	100	2.34	94553	100	2.13
PP	10	14785	100	0.33	9898	100	0.44	9890	100	0.40
PRD	2	1862	90	0.01	1564	98	0.03	1570	97	0.02
PQ	4	5346	100	0.06	3620	100	0.05	3630	100	0.05
RG	10	98303	100	2.05	50834	100	1.71	49486	100	1.54
RB	10	265700	2	18.22	298376	92	6.66	307132	90	6.20
SF2	2	1990	100	0.02	2773	100	0.04	2941	100	0.04
SBT	2	2581	100	0.03	2650	100	0.04	2747	100	0.04
SBT3	3	1463	100	0.02	1204	100	0.02	1190	100	0.02
–	5	3488	100	0.05	2463	100	0.05	2474	100	0.04
SWF	10	28157	100	0.99	24023	100	4.59	24519	100	4.26
S5	4	5734	95	0.07	4560	96	0.08	4527	99	0.07
S7	4	4707	100	0.05	3453	98	0.06	3294	99	0.05
S10	4	4788	100	0.06	3748	99	0.07	3532	100	0.06
SIN	10	14653	100	0.38	8633	100	0.42	8703	100	0.39
–	20	61242	100	2.99	30446	100	5.06	30814	100	2.47
ST	9	900090	100	28.84	327803	97	11.46	314691	100	10.81
WP	4	14598	96	0.17	10484	100	0.15	9851	100	0.12
ZAK	10	62530	100	1.33	31944	100	1.38	31891	100	1.27
tr		2106360	4750	70.94	1171088	4810	49.98	1155070	4850	42.32

second columns of Table 1, respectively, represent the acronym of the problems and their corresponding dimensions. All the algorithms have a positive sr for all the 50 test problems.

We first compare de with deim to see the combined effect of M_d and M_e per iteration. The combined effect of M_d and M_e over M_d can clearly be seen in tr from Table 1. For example, deim is superior to de by 44%, 30% and 1.3% with respect to fe, cpu and sr, respectively. These significant reductions clearly demonstrate the usefulness of M_e as a 'co-mutation' operator per iteration in de.

We then compare de with deam to see the effect of adapting either M_d or M_e per iteration in de. A comparison of de and deam using tr shows that deam is superior to de by 45%, 40% and 2.1% with respect to fe, cpu and sr, respectively. These results clearly show that a significant improvement of de can be achieved by implementing M_e on iteration basis in a controlled manner. A comparison of de, deim and deam shows that deam is the best and deim is the runner-up.

It can be seen from Table 1 that fe and cpu are dominated by the problems with higher dimensions, with a few exceptions, e.g. NF2 (Neumaier 2 Problem) and WP (Wood Problem). Next, we study the effects of the changes made to de on high and low dimensional problems by separating them into two sets: A and B. Set A consists of problems of dimensions 9 to 20 and set B consists of problems of dimensions 2 to 8. The total results incurred by the algorithms on A and B are summarized in Tables 2 and 3, respectively.

Table 2: Comparison of algorithms on A

	de	deim	deam
fe	1938928	1034765	1028017
sr	1472	1552	1545
cpu	68.77	47.09	39.50

Table 2 shows that both the new algorithms, deim and deam, are much superior to de in all respects. On set A, deim is superior to de by about 47%, 32% and 5.4% with respect to fe, cpu and sr, respectively. Likewise, deam is superior to de by about 47%, 43% and 5.0% with respect to fe, cpu and sr, respectively. Notice that deam is the winner in terms of cpu. We also ran the algorithms on 20 dimensional Rastrigin (RG) problem. The de algorithm failed to solve this problem within the maximum number of iterations while the other two algorithms successfully solved it. However, the results of this problem are not reflected in our comparison.

A similar comparison of the algorithms on the problem set B draws a different picture. For the low dimensional problems, set B, de is the winner and deam is the runner-up with respect to cpu and the converse is true with respect to sr. The new algorithms, deim and deam, are however always superior to de with respect to fe – attaining about 18% and 24% superiority over de, respectively. The overall comparison clearly shows that the new algorithms are much superior to de. On average they achieved superiority over de with respect to efficiency (fe and

Table 3: Comparison of algorithms on B

	de	deim	deam
fe	167432	136323	127053
sr	3278	3258	3305
cpu	2.17	2.89	2.82

cpu) without compromising the robustness. This shows that our research objective has been achieved.

5 Conclusion

In this paper, we identified a number of drawbacks of the differential evolution (de) algorithm, associated with the parameters of the algorithm and the convergence rate. We then proposed a scheme for choosing the parameter values and modified the mutation rule of de in an attempt to remedy these drawbacks. Consequently, we proposed two new versions of the de algorithm. Each of the new version incorporates a new mutation rule based on a variant of an attraction - repulsion scheme of electromagnetism - like algorithm. The algorithms also select parameter values using a uniform distribution over some given ranges. Numerical results obtained using a large set of problems shows that the new algorithms are much superior to the original algorithm. In particular, the new algorithms significantly reduced the number of function evaluations and cpu time of de without compromising the robustness of de.

References

[1] Storn R, Price K. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 1997; 11: 341–59.

[2] Birbil SI, Fang S-C. An electromagnetism-like mechanism for global optimization. Journal of Global Optimization 2003; 25(3): 263–82.

[3] Birbil SI, Fang S-C, Sheu R-L. On the convergence of a population-based global optimization. Journal of Global Optimization 2004; 30: 301–18.

[4] Kaelo P, Ali MM. A numerical study of some modified differential evolution algorithms. European Journal of Operational Research 2005, accepted for publication.

[5] Michalewicz Z. Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, 1996.

[6] Ali MM, Törn A. Population set based global optimization algorithms : some modifications and numerical studies. Computers and Operations Research 2004; 31(10): 1703–25.

[7] Price, WL. Global optimization by controlled random search. Journal of Optimization Theory and Applications 1983; 40: 333–48.

[8] Kaelo P, Ali MM. Probabilistic adaptations of point generation schemes in some global optimization algorithms. Optimization Methods and Software 2005, accepted for publication.

[9] Lampinen J, Zelinka, I. Mechanical engineering design optimization by differential evolution. In: Corne D, Dorigo M, Glover F,(eds). New Ideas in Optimization. London, McGraw-Hill, 1999. pp. 127–46.

[10] Storn R. System design by constraint adaptation and differential evolution. IEEE Transactions on Evolutionary Computation 1999; 3(1): 22–34.

[11] Babu BV, Sastry KKN. Estimation of heat transfer parameters in a trickle-bed reactor using differential evolution and orthogonal collocation. Computers and Chemical Engineering 1999; 23(3): 327–39.

[12] Ali MM, Törn A. Topographical differential evolution using pre-calculated differentials. In : G. Dzemyda, V. Saltenis and A. Zilinskas (editors). Stochastic and Global Optimization, Kluwer Academic Publisher, 2002. pp.1–17.

[13] Debels D, De Reyck B, Leus R, Vanhoucke M. A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. European Journal of operational research 2005, In press.

[14] Price K. An introduction to differential evolution. In: Corne D, Dorigo M, Glover F,(eds). New ideas in optimization. London, McGraw-Hill, 1999. pp. 79–108.

[15] Kaelo P. Some population set based methods for unconstrained global optimization. *PhD thesis*, to be submitted in 2005.

[16] Törn A, Ali MM, Viitanen S. Stochastic global optimization: problem classes and solution techniques. Journal of Global Optimization 1999; 14: 437–47.

[17] Ali MM, Khompatraporn C, Zabinsky ZB. A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. Journal of Global Optimization 2005; 31(4): 635–672.

[18] Chelouah R, Siarry P. Genetic and Nelder-Mead algorithms for a more accurate global optimization of continuous multiminima functions. European Journal of Operational Research 2003; 16(2): 335–48.

[19] Fan, H.-Y. and Lampinen, J., 2003, A trigonometric mutation operation to differential evolution, Journal of Global Optimization, 27(1), 105-129.