

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/

Optimization Methods and Software
Vol. 00, No. 00, Month 200x, 1–18

Probabilistic adaptations of point generation schemes in some global optimization algorithms

P. KAELO and M. M. ALI*
School of Computational and Applied Mathematics,
Witwatersrand University, Private Bag 3,
Wits - 2050, Johannesburg, South Africa
(Received 00 Month 200x; In final form 00 Month 200x)

Modifications in the trial point generation scheme are suggested to the differential evolution (de) and the controlled random search (crs) algorithm to improve their performances. Each of the modified de and the crs algorithm uses two different point generation schemes. A probabilistic adaptation of point generation schemes in each of the algorithms is proposed. Localizations in acceptance rule and in trial point generation schemes are also suggested in the resulting de and the crs algorithm respectively. Numerical experiments indicate that the resulting algorithms are considerably better than their original counterparts. Therefore, they offer a reasonable alternative to many currently available stochastic algorithms, especially for problems requiring 'direct search type' methods.

Keywords: Global optimization, population set based method, modified differential evolution, modified controlled random search, probabilistic adaptation, continuous variable.

2000 Mathematics Subject Classifications: 65K05, 90C56, 78M50.

1 Introduction

The global optimization problem in this paper follows the form:

$$\text{minimize } f(x) \text{ subject to } x \in \Omega,$$

where x is a continuous variable with domain $\Omega \subset \mathbb{R}^n$, and $f(x) : \Omega \mapsto \mathbb{R}$ is a continuous real-valued function. The domain Ω is defined by specifying upper (u^j) and lower (l^j) limits of each component j. We denote the global optimal solution by x^*, with its corresponding global optimal function value $f(x^*)$ or f^* for a short hand notation. The function $f(x)$ may be non-linear or non-differentiable. Our only

*Corresponding author. Tel.:+27 11 7176139; Fax: +27 11 7176149 E-mail: mali@cam.wits.ac.za

requirement on the problem is that $f(x)$ can be computed for any $x \in \Omega$. A number of algorithms have been proposed for solving the above problem. They include genetic algorithms (ga) [1, 2], differential evolution (de) [3] and controlled random search (crs) [4, 5] amongst others. These methods use no properties of the function being optimized. They are also easy to implement. This paper is concerned with the crs2 [5] version of the crs algorithm and the de algorithm [3].

Controlled random search and differential evolution are population set based global optimization algorithms [2] and are purely heuristic. Like all population set based direct search methods, they use a population set S. The initial set

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

consists of N 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.

The objective of this paper is to investigate the efficiency and robustness of de and crs2 and to suggest some modifications to improve their performances. To achieve our objective, we first carried out an extensive numerical study of de and crs2 using a large set of 50 test problems. The numerical study suggested that each of the algorithms has some drawbacks. The main drawback of the crs2 algorithm is that it is not very robust in terms of finding the global minimum. On the other hand, while de has proved to be very efficient in terms of finding the global minimum, it has proved inefficient in terms of the number of function evaluations and cpu time needed to reach the global minimum. This shows that the point generation scheme of de is more search exploratory resulting in more success, but it is at a cost of a high number of function evaluations and cpu time. On the other hand, the point generation scheme of crs2 is less exploratory which results in less success but it is very efficient in terms of the number of function evaluations and cpu time. These features of the algorithms are also reported in [2] where the reasons for the features are also given.

In this paper we propose a new de and a new crs algorithm that use the complementary strengths of each other's point generation scheme. We embed the trial point generation scheme of crs2 in de and the trial point generation scheme of de in crs2. This is done so as to reduce the number of function evaluations in de and to improve the success in finding the global minimum in crs2. A scheme is then introduced in both the algorithms to probabilistically select a point generation scheme at each iteration. For example, in de, trial points are generated using its own scheme with some probability and using the point generation scheme of crs2 with the remaining probability. In addition, we suggest a local technique in de and in crs2 for exploration of searches at initial stages and faster convergence at later stages.

The organization of the paper is as follows. In section 2 we briefly describe de and crs2. 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 crs2

In this section we briefly present the de and the crs2 algorithm. We also introduce notations that will be used throughout this paper.

2.1 *Differential evolution (de)*

The contraction process in de involves replacing bad points with better points per iteration. In particular, de attempts to replace all points in S by new points at each iteration. Therefore, in each iteration, N competitions are held to determine the members of S for the next iteration. The i-th ($i = 1, 2, \cdots, N$) 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 point $\hat{x}_i$ determined by mutation. 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. A weighted difference of any two points is then added to the third point to give the mutated point $\hat{x_i}$, mathematically described as:

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

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

$$y_i^j = \begin{cases} \hat{x}_i^j \text{ if } R^j \leq C_R \text{ or } j = I_i \\ x_i^j \text{ if } R^j > C_R \text{ and } j \neq I_i \ , \end{cases} \tag{2}$$

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 (2) is to obtain the trial point y_i with components coming from the components of the target point x_i and the mutated point $\hat{x}_i$. And this is ensured by introducing C_R and the set I. Notice that for $C_R = 1$ the trial point y_i is the replica of the mutated point $\hat{x}_i$. The effect of C_R has been studied in [2, 6] and it was found that $C_R = 0.5$ is a good choice. The targeting process continues until all members of S are considered. After all N trial points y_i have been generated, acceptance is applied. In the acceptance phase, the function value at the 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 continues until a certain

stopping condition is met. The algorithm for de can be found in [3]. We refer to the global point generation scheme defined by (1) and (2) as the differential scheme and denote it by P_{gd}.

2.2 *Controlled random search (crs2)*

Unlike de, crs2 gradually contracts S by replacing the current worst point in S with a better trial point through repeated use of simplexes. At each iteration, therefore, the current worst point x_h in S is targeted. A trial point is generated by forming a simplex. The simplex is formed using $n+1$ distinct points, $x_{p(1)}, x_{p(2)}, \cdots, x_{p(n+1)}$, where $x_{p(2)}, x_{p(3)}, \cdots, x_{p(n+1)}$ are chosen at random (with replacement) from S and the point $x_{p(1)}$ is always the current best point x_l in S. The trial point is obtained by reflecting the point $x_{p(n+1)}$ in the centroid of the remaining n points of the simplex, as in the Nelder and Mead algorithm [7]. The trial point $\tilde{x}$ is thus given as

$$\tilde{x} = 2G - x_{p(n+1)}, \tag{3}$$

where the centroid G is given by

$$G = \frac{1}{n}\sum_{j=1}^{n} x_{p(j)}. \tag{4}$$

The trial point generation in the original crs1 [4] is similar to crs2 except that in crs1 the point $x_{p(1)}$ is also randomly chosen. The process of finding a trial point and replacing the current worst point in S, if the trial point is better than the current worst in S, is repeated until a certain stopping condition is met. It has been found that crs2 is much superior compared to crs1 [5]. The algorithm for crs2 can be found in [5]. We refer to the global point generation scheme defined by (3) and (4) as the simplex scheme and denote it by P_{gs}.

Although de and crs are both population set based methods, they associate a different meaning with iteration. For instance, there are N function evaluations needed for de per iteration as opposed to a single function evaluation needed for crs. A modified version of de is suggested in [2]. The modification is based on the scaling parameter F in mutation rule (1). There are a number of modifications suggested to crs2 [8–10]. They incorporate various local techniques in crs2. All previous modified versions of de and crs2 were suggested to make their convergence faster and also to improve their robustness in locating the global minimizer. However, these modifications were justified by numerical experiments using a small set of low dimensional test problems. We used a large set of test problems of dimensions ranging from 2 to 20.

In order to facilitate the understanding and to make the difference between the algorithms explicit, we append to each individual algorithm name the appropriate

parameter containing '(global scheme, local scheme)'. Using this notation, we can write de as de($P_{gd}, \cdot$) and crs2 as crs(P_{gs},·).

3 Proposed modifications to de and crs2

In this section, we propose a new version for each of crs and de. The modified versions generate global trial points using more than one point generation scheme. They generate trial points using some probability distribution over the set of schemes, say $\{s_1, s_2\}$, per iteration. Here, s_1 represents scheme 1 and s_2 represents scheme 2. The new algorithms use the concept of a learning system used in learning automata [11, 12] in deciding which scheme to use in generating trial point(s) per iterartion.

In learning automata, a learning system is composed of a stochastic automaton and a performance evaluation unit. The stochastic automaton consists of a finite set

$$U = \{s_1, s_2, \cdots, s_m\}$$

with m actions. A probability $\alpha_i(k)$ is associated with each action s_i ($i = 1, 2, \cdots, m$) at each iteration k and $\sum_{i=1}^{m} \alpha_i(k) = 1$. Initially, all the actions have equal probabilities. At each iteration k, the stochastic automaton selects an action s_i associated with an objective function $f(s_i)$ with probability $\alpha_i(k)$ and modify the overall probability distribution based on the response of the performance evaluation unit. The performance evaluation unit responds to the action s_i by giving an evaluation signal $a_k \in \{0, 1\}$ where $a_k = 0$ corresponds to a reward, if the selected action s_i meets the intended target, and $a_k = 1$ for penalty otherwise. A reinforcement-learning scheme for the automaton is then used to reward or penalize the action s_i. A number of reinforcement-learning schemes have been proposed, both linear and non-linear (see [11, 12] and references therein).

In this paper, we use the non-linear reinforcement-learning scheme that was used in [11]. Initially, equal probabilities (0.5) are assigned to each of the point generation schemes s_1 and s_2 and these probabilities are then updated, using the reinforcement scheme, at each iteration depending on whether the scheme used was successful or not. Thus, the probability $\alpha_1(k)$ is assigned to s_1 and $\alpha_2(k) = 1 - \alpha_1(k)$ to s_2 at iteration k. If the scheme s_1 proves successful then $\alpha_1(k)$ is increased using the rule

$$\alpha_1(k+1) = \alpha_1(k) + \beta_0 \alpha_1(k)(1 - \alpha_1(k)), \tag{5}$$

where $0 \leq \beta_0 \leq 1$ is a user supplied parameter that controls the increments. On the other hand, if s_1 does not prove successful then the probability $\alpha_2(k)$ of s_2 is increased. This means we decrease $\alpha_1(k)$ using the rule:

$$\alpha_1(k+1) = \alpha_1(k) - \beta_1 \alpha_1(k)(1 - \alpha_1(k)), \tag{6}$$

where $0 \leq \beta_1 \leq 1$ is also a user supplied parameter that controls the reductions. We can reward or penalize the probability $\alpha_2(k)$ in a similar way. However, rewarding $\alpha_2(k)$ means penalizing $\alpha_1(k)$ and penalizing $\alpha_2(k)$ means rewarding $\alpha_1(k)$. We can therefore work with only one probability, say $\alpha_1(k)$. The use of this adaptation in an algorithm is to guide the algorithm in deciding on which scheme to use most in generating trial points for any given problem. This allows an algorithm to bias the trial points generation to a scheme that solves a given problem most efficiently. In addition to the probabilistic combination of point generation schemes, we also propose localizations around the best point in each of the modified algorithms.

3.1 *A probabilistic crs2 with localizations*

In this section, we introduce a modified crs algorithm which incorporates the differential scheme P_{gd} to generate global trial points in crs2. In the modified crs2 algorithm, a trial point $\tilde{x}$ is generated using a probability distribution over $\{P_{gsl}, P_{gdl}\}$ per iteration. P_{gsl} is a trial point generation scheme that uses P_{gs} to generate a trial point $\tilde{x}$ but if $\tilde{x}$ is unsuccessful in replacing the current worst point x_h in S, a local technique is employed to generate a second trial point $\tilde{y}$. The point $\tilde{y}$ then competes with x_h. The local technique generates $\tilde{y}$ exploring the region around the current best point x_l in S by reflecting the unsuccessful trial point $\tilde{x}$. In particular, this is done by coordinate-wise reflection of $\tilde{x}$ through x_l as follows:

$$\tilde{y}^j = (1 + \rho_j){x_l}^j - \rho_j \tilde{x}^j, \tag{7}$$

where $\tilde{y}^j$, $\tilde{x}^j$ and x_l^j are the j-th coordinate of $\tilde{y}$, $\tilde{x}$ and x_l respectively and ρ_j is a random number in $[0, 1]$ for each j. If $\tilde{y}^j$ is not in the feasible region, then (7) can be repeated for a number of times. We refer to this local technique as the local mutation technique and denote it by P_{lm}. This feature of localization has a global exploration effect at earlier stages when the points in S are scattered and a local effect in terms of rapid convergence at later stages. P_{gdl} also associates a similar meaning. The scheme P_{gd} generates a trial point $\tilde{x}$ using (8) (see later) and (2). First, a point $\hat{y}$ is generated using (8) and then $\tilde{x}$ is obtained using (2). Components of $\tilde{x}$, therefore, come from the mutated point $\hat{y}$ and the targeted point x_h. If the trial point $\tilde{x}$ is unsuccessful then P_{gdl} employs P_{lm} as in P_{gsl}.

The probabilities $\alpha_1(k)$ and $\alpha_2(k)$ are assigned to P_{gsl} and P_{gdl} respectively. These probabilities $\alpha_1(k)$ and $\alpha_2(k)$ are updated per iteration. If the trial point $\tilde{x}$ (or $\tilde{y}$ whenever $\tilde{x}$ fails) replaces x_h, then the probability of the associated scheme is increased. However, if both $\tilde{x}$ and $\tilde{y}$ fail to replace x_h, then the probability of the scheme is reduced. For example, if $\tilde{x}$ or $\tilde{y}$ generated by P_{gsl} succeeds in replacing x_h in S, then the scheme is rewarded by increasing $\alpha_1(k)$ (reducing $\alpha_2(k)$) using (5), otherwise $\alpha_1(k)$ is decreased ($\alpha_2(k)$ increased) using (6). The same rule applies to the scheme P_{gdl}. This version of crs is referred to as a probabilistic crs with

localizations and denoted by crs(P_{gsd},P_{lm}), where P_{gsd} means that trial points are generated probabilistically using P_{gs} or P_{gd}. Below we present the algorithm for crs(P_{gsd}, P_{lm}).

Algorithm 1 : the crs(P_{gsd}, P_{lm}) algorithm

Step 1. **Determine the initial set** S:

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

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

Step 2. **Stopping rule based on the best point and the worst point**. Determine the worst point x_h and best point x_l. If the stopping condition such as $|f(x_h) - f(x_l)| \leq \epsilon$ is satisfied, then stop.

Step 3. **Procedure selection**. If $\alpha_1(k) \geq \mu_k$ then go to Step 3a else go to step 3c.

Step 3a. **Generate a trial point** $\tilde{x}$ **using** P_{gs}.
Choose randomly n points $x_{p(2)}, x_{p(3)}, \cdots, x_{p(n+1)}$ from S, different from the current best, and let $x_{p(1)} = x_l$. Compute $\tilde{x}$ using (3) and (4). If $\tilde{x} \notin \Omega$ then repeat Step 3a. If $f(\tilde{x}) \geq f(x_h)$ then go to Step 3b else go to Step 4.

Step 3b. **Mutate** $\tilde{x}$ **to find** $\tilde{y}$ **using** P_{lm}. Generate another trial point $\tilde{y}$ using the trial point $\tilde{x}$ and x_l as follows:

```
for j = 1 to n do
    repeat
      ỹ^j = (1 + ρ_j) x_l^j − ρ_j x̃^j
    until ỹ^j ∈ Ω
  end.
```

If $f(\tilde{y}) \geq f(x_h)$ then go to Step 5 else go to Step 4.

Step 3c. **Generate a trial point** $\tilde{x}$ **using** P_{gd}.
Select randomly three distinct points $x_{p(1)}, x_{p(2)}$ and $x_{p(3)}$ from S, all different from x_h. Generate a trial point $\tilde{x}$ using

$$\hat{y} = x_{p(1)} + F\left(x_{p(2)} - x_{p(3)}\right), \tag{8}$$

where F is random in $[0.4, 1]$, and the crossover rule (2) . If $\tilde{x} \notin \Omega$ then repeat Step 3c. If $f(\tilde{x}) \geq f(x_h)$ then go to Step 3d else go to 4.

Step 3d. **Mutate** $\tilde{x}$ **to find** $\tilde{y}$ **using** P_{lm}. Same as in Step 3b. If $f(\tilde{y}) \geq f(x_h)$ then go to Step 5 else go to Step 4.

Step 4. **Update** S. If this step is reached from Step 3a or 3b then go to Step 4a. Else go to Step 4b.

Step 4a **Replace and reward**. Replace x_h and $f(x_h)$ by $\tilde{x}$ and $f(\tilde{x})$ respectively or

by $\tilde{y}$ and $f(\tilde{y})$ respectively. $\alpha_1(k+1) = \alpha_1(k) + \beta_0\alpha_1(k)(1-\alpha_1(k))$. Set $k = k+1$. Go to Step 2.

Step 4b **Replace and penalize**. Replace x_h and $f(x_h)$ respectively by $\tilde{x}$ and $f(\tilde{x})$ or $\tilde{y}$ and $f(\tilde{y})$. $\alpha_1(k+1) = \alpha_1(k) - \beta_1\alpha_1(k)(1-\alpha_1(k))$. Set $k = k+1$. Go to Step 2.

Step 5. **Update** $\alpha_1(k)$. If this step is reached from Step 3a or Step 3b then go to Step 5a else go to Step 5b.

Step 5a. **Penalty** $\alpha_1(k+1) = \alpha_1(k) - \beta_1\alpha_1(k)(1-\alpha_1(k))$. Set $k = k+1$. Go to Step 3.

Step 5b. **Reward** $\alpha_1(k+1) = \alpha_1(k) + \beta_0\alpha_1(k)(1-\alpha_1(k))$. Set $k = k+1$. Go to Step 3.

Remarks

1. The crs(P_{gsd},P_{lm}) is a generalized crs algorithm from which other crs algorithms can be obtained. For example, crs(P_{gsd},$\cdot$) is a probabilistic crs algorithm without localizations, crs(P_{gsd},$\cdot$)$|_{\alpha_1(k)=1}$ is the original crs2=crs(P_{gs}, $\cdot$) and crs(P_{gsd},P_{lm})$|_{\alpha_1(k)=1}$ is the original crs2 with localizations, i.e. crs(P_{gs}, P_{lm}).
2. μ_k in Step 3 is a random number in $[0,1]$ drawn for each k.
3. In Step 3b, the repetition was done for atmost 4 times. If after 4 times $\tilde{y}^j$ is still not feasible then the scheme P_{gs} is penalized in Step 5. This ensures that a lot of time is not spent on trying to find a feasible $\tilde{y}^j$.

3.2 *A probabilistic de with localizations*

In this section, we introduce the modified de algorithm. This new version of de incorporates the simplex scheme P_{gs} in de. That is, trial points are generated using a probability distribution over $\{P_{gs}, P_{gd}\}$. An iteration in de attempts to replace N points in S by repeatedly targeting x_i ($i = 1, \cdots, N$). For every iteration therefore, a scheme from the set $\{P_{gs}, P_{gd}\}$ is selected with some probability and N trial points y_i are generated. If P_{gd} is selected, then y_i are generated using (8) and (2). If P_{gs} is selected, then y_i are generated using (3) and (4). For each y_i, if $f(y_i) < f(x_i)$ then a local technique may be employed. This local technique explores the region around y_i and x_l, the current best point in S, using some reflection and contraction rules to give two points r_i and c_i respectively. In particular, each time a trial point y_i is found which is better than the targeted point x_i but worse than x_l, then r_i and c_i are found, with some probability, using the following rules:

$$r_i = x_l - (y_i - x_l), \tag{9}$$

and

$$c_i = x_l + 0.5(y_i - x_l). \tag{10}$$

The best of y_i, r_i and c_i then competes with the target point x_i. If $f(y_i) < f(x_l)$ then y_i replaces x_i in S. We refer to this local technique as the localization around the best point and denote it by P_{lb}. Notice that P_{lb} is invoked in the acceptance phase of de, i.e. if the trial point y_i is found to be successful.

Rewarding or penalizing a scheme comes after an iteration has been completed. The probabilities $\alpha_1(k)$ and $\alpha_2(k)$ are assigned to P_{gs} and P_{gd} respectively. If a scheme, say P_{gs}, replaces a high number of points in S, then $\alpha_1(k)$ is increased. If, however, a moderate number of points are replaced, then $\alpha_1(k)$ is left unchanged. Otherwise, if P_{gs} replaces a small number of points then $\alpha_1(k)$ is reduced. The same is true for the scheme P_{gd}. We refer to this modified de as the probabilistic de with localizations and denote it by de(P_{gsd},P_{lb}). Below we present the algorithm for de(P_{gsd},P_{lb}).

Algorithm 2: the de(P_{gsd},P_{lb}) algorithm

Step 1. **Determine the initial set** S : Same as in Algorithm 1.

Step 2. **Determine the best point and the worst point in** S : Same as in Algorithm 1.

Step 3. **Procedure selection**. If $\alpha_1(k) \geq \mu_k$ then go to Step 3a else go to step 3b.

Step 3a **Generate points to replace points in** S for the next iteration using P_{gs}. For each $x_i \in S$ $(i = 1, 2, \cdots, N)$, determine y_i by the following operation:

- Compute y_i using (3) and (4) as done in Algorithm 1. If $y_i \notin \Omega$ then repeat the above process.

Go to step 4.

Step 3b **Generate points to replace points in** S for the next iteration using P_{gd}. For each $x_i \in S$ $(i = 1, 2, \cdots, N)$, determine y_i by the following two operations:

- **Mutation**: Use mutation rule (8).
- **Crossover**: Use crossover rule (2).

Go to step 4.

Step 4. **Localization and acceptance rule to replace points in** S. Set $R_c = 0$.
For $i = 1, 2, \cdots, N$, do the following Steps 4a-4d

Step 4a **Controlled localization**: If $f(y_i) \geq f(x_i)$ then retain x_i, otherwise if $\tau^i < \omega$ and $f(y_i) > f(x_l)$ then go to Step 4b else go to Step 4d.

Step 4b **Reflection** : Find r_i using (9). If $r_i \notin \Omega$ then go to Step 4c. If $f(r_i) < f(y_i)$ then replace x_i by r_i else go to Step 4c. Set $R_c = R_c + 1$ and go to Step 4a for the next index.

Step 4c **Contraction** : Find c_i using (10). If $f(c_i) < f(y_i)$ then replace x_i by c_i else go to Step 4d. Set $R_c = R_c + 1$ and go to Step 4a for the next index.

Step 4d **Replace**: Set $R_c = R_c + 1$. Replace x_i by y_i and go to Step 4a for the next index.

If Step 4 was reached from Step 3a then go to Step 5 else go to Step 6.

Step 5. **Update** $\alpha_1(k)$:

Step 5a **Reward**. If $R_c \leq 0.67N$ (nearest integer to $0.67N$) then go to Step 5b. $\alpha_1(k+1) = \alpha_1(k) + \beta_0\alpha_1(k)(1-\alpha_1(k))$. Set $k = k+1$. Go to Step 2.

Step 5b **Unchanged**. If $R_c \leq 0.33N$ then go to Step 5c. $\alpha_{k+1} = \alpha_1(k)$. Set $k = k+1$. Go to Step 2.

Step 5c **Penalize**. $\alpha_1(k+1) = \alpha_1(k) - \beta_1\alpha_1(k)(1-\alpha_1(k))$. Set $k = k+1$. Go to Step 2.

Step 6. **Update** $\alpha_1(k)$:

Step 6a **Penalize**. If $R_c \leq 0.67N$ then go to Step 6b. $\alpha_1(k+1) = \alpha_1(k) - \beta_1\alpha_1(k)(1-\alpha_1(k))$. Set $k = k+1$. Go to Step 2.

Step 6b **Unchanged**. If $R_c \leq 0.33N$ then go to Step 6c. Set $k = k+1$. $\alpha_{k+1} = \alpha_1(k)$. Set $k = k+1$. Go to Step 2.

Step 6c **Reward**. $\alpha_1(k+1) = \alpha_1(k) + \beta_0\alpha_1(k)(1-\alpha_1(k))$. Set $k = k+1$. Go to Step 2.

Remarks

4. The de(P_{gsd},P_{lb}) algorithm is a generalized de algorithm from which other de algorithms can be obtained. For example, de(P_{gsd},$\cdot$) is a probabilistic de algorithm without localizations, de(P_{gsd},$\cdot$)$|_{\alpha_1(k)=0}$ is the original de=de(P_{gd}, $\cdot$) and de(P_{gsd},P_{lb})$|_{\alpha_1(k)=0}$ is the original de with localizations, i.e. de(P_{gd}, P_{lb}).
5. For the probabilistic de algorithms we used a variable scaling factor F as opposed to a fixed F in the original de. That is, F in (8) is variable.
6. τ^i in Step 4a is a random number in $[0, 1]$ for each i, and ω is a user provided value within $[0, 1]$. ω controls the frequency of local explorations around y_i and x_l, i.e. ω is the probability of invoking P_{lb} in de(P_{gsd},P_{lb}).
7. R_c in Step 4 counts the number of target points replaced. More details of R_c are given in section 4.
8. Unlike in the original de, where the trial point y_i competes with the target point x_i, in de(P_{gsd},P_{lb}) either the trial point y_i competes with the target point x_i or the best of y_i, r_i and c_i competes with x_i.

4 Numerical results and discussion

In this section, we compare all algorithms using 50 test problems. These problems have a variety of inherent difficulty [13]. All the problems have continuous variables and a detailed description of each problem can be found in [6, 14]. We compare the crs type algorithms and the de type algorithms separately. Thus, we compare crs2 with crs(P_{gsd}, P_{lm}) and de with de(P_{gsd}, P_{lb}). We answer the following research questions:

- Does the de algorithm benefit from P_{gs} and localization P_{lb}?
- Does the crs2 algorithm benefit from P_{gd} and localization P_{lm}?

Each algorithm was run 100 times on each of the 50 test problems to determine the success rate (or percentage success), sr, of each algorithm on each problem. There were 5000 runs in total for each algorithm. We calculated the average number of function evaluations (fe) and cpu times (cpu) for those runs for which the global minima were found. We note that except for the 9 dimensional Price's Transistor Modeling (PTM) and 10 dimensional Odd Square (OSP), Salomon (SAL) and Shekel Foxholes (FX) functions [14], at least one of the algorithms have positive success rate on the remaining 46 problems. Therefore, results for these four functions are not reflected in our presentation. We used sr, fe and cpu as the criteria for comparison. A success was counted when the value $f(x_l)$ of a run was such that $|f_{opt} - f(x_l)| \leq 0.01$ where f_{opt} is the known global minimum of the problem being solved.

4.1 *Parameter selection*

Each of the crs type algorithms has some parameter values to be provided by the user that are common to all algorithms. These common parameters are the number of elements N of S, the maximum number of iterations T and the parameter ϵ for the stopping rule. We took the value of N to be $10(n+1)$, where n is the dimension of the problem. This value of N is suggested in [5]. N is a heuristic choice and can therefore always be increased for obtaining the global minimum with higher probability. In every case, a run was terminated when either a maximum number of iterations $T = 1000n^2$ was reached or the function values of all points in S were identical to four decimal places, i.e,

$$|f(x_h) - f(x_l)| \leq \epsilon = 10^{-4}. \tag{11}$$

Other parameters used in crs(P_{gsd}, P_{lm}) are β_0 in (5), β_1 in (6), C_R in (2) and F in (8). Different combinations of β_0 and β_1 values can be used. We carried out numerical experiments using various combinations of values for these parameters. However, the results presented here are the best results obtained for $\beta_0 = \beta_1 = 0.15$. We used an impirical value $C_R = 0.3$ and a variable $F \in [0.4, 1]$ whenever crs(P_{gsd}, P_{lm}) used P_{gd}. For more details of these parameter values for crs type algorithms, see [6].

The parameters of the de type algorithms are the scaling factor F in the mutation rules (1), the controlling parameter C_R in the crossover rule (2), the stopping rule parameter ϵ, the maximum iteration number T and N, the number of points in S. The results presented here were obtained for $F = 0.5$ and $C_R = 0.5$ in the case of the original de. The value $C_R = 0.5$ was also used in de(P_{gsd}, P_{lb}). These values were found by impirical means. The value of N was taken as $10n$ [2] and a run was terminated when either $T = 10000$ or the condition (11) was satisfied. The parameters β_0 and β_1 used here were set to $\beta_0 = 0.30$ and $\beta_1 = 0.45$ for de(P_{gsd}, P_{lb}). For results using other combinations of β_0 and β_1, see Kaelo [6]. In the probabilistic de type algorithms, $\alpha_1(k)$ was increased when the number of points replaced (R_c) per iteration

was greater than or equal to $0.67N$ (nearest integer to $0.67N$). $\alpha_1(k)$ was reduced when the number of points (R_c) replaced was less than or equal to $0.33N$ and $\alpha_1(k)$ was kept unchanged if the total number of points replaced (R_c) was greater than $0.33N$ but less than $0.67N$. These values were found by numerical experiments [6].

It is clear from the previous two paragraphs that there are some parameters that belong to the de type algorithms that were used in the crs type algorithms, and vice versa. This is because of the use of the differential scheme P_{gd} in crs2 and the simplex scheme P_{gs} in de. Notice also that these parameters have different values in crs and de. This is the case because although crs and de use a population based set S that is repeatedly contracted, they are essentially different algorithms.

In both types of the probabilistic algorithms, we forced $\alpha_1(k), k \geq 1$, to lie in $[0.05, 0.95]$. That is, if $\alpha_1(k)$ goes below 0.05, we set $\alpha_1(k) = 0.05$ by clipping. This was done in order to avoid the algorithms switching entirely to one scheme. Thus $\alpha_1(k)$ and $\alpha_2(k)$ lie in $(0, 1)$ to allow the algorithms to be able to switch from one scheme to the other.

4.2 *Comparisons and discussions*

The results of de and crs2 along with the results of the new algorithms are presented in Table 1, where tr in the last row represents the total fe and sr.

To address our first research question, we compare de with de(P_{gsd},P_{lb}). The effect of the parameter ω which controls the frequency of localizations around y_i and x_l has been studied in Kaelo [6]. It was observed that as ω increases, fe decreases. However, this trend is not true for sr. Overall best results were obtained for $\omega = 0.1$. Therefore, we present the results of de(P_{gsd},P_{lb}) for $\omega = 0.1$. Full results and discussion on ω can be found in [6]. All de type algorithms were able to solve 46 problems. From Table 1, we see that although de(P_{gsd},P_{lb}) secured only 46 more sr than de, total fe for de(P_{gsd},P_{lb}) is considerably less than that of de. This shows that the point generation scheme P_{gs} together with P_{lb} have a positive effect both in fe and sr. We note that there are problems for which de(P_{gsd},P_{lb}) needed more fe than de, especially for the difficult problems Epistatic Michalewicz (EM) and Neumaier 2 (NF2). However, while de and de(P_{gsd},P_{lb}) have the same sr for NF2, de(P_{gsd},P_{lb}) has more sr than de for EM. On the other hand, de(P_{gsd},P_{lb}) is much superior to de in terms of fe and sr on other difficult problems such as modified Langerman (ML), Rosenbrock (RB) and Storn's Tchebychev (ST). If we compare the total results then de(P_{gsd},P_{lb}) is superior to de with respect to fe and sr.

We now show the effects of the new features P_{gs} and P_{lb} separately in de. We first compare de with de($P_{gsd}, \cdot$) to see the effect of the combined scheme P_{gsd} in de=de($P_{gsd}, \cdot$)$|_{\alpha_1(k)=0}$. We then compare de($P_{gsd}, \cdot$) with de(P_{gsd}, P_{lb}) and de($P_{gsd}, \cdot$)$|_{\alpha_1(k)=0}$ with de(P_{gsd}, P_{lb})$|_{\alpha_1(k)=0}$ to see the effect of P_{lb} on de and de($P_{gsd}, \cdot$) respectively. The total results of all these algorithms are presented in Table 2, where we also present the total cpu. These results are for all the 46 problems

Table 1. Comparison of de, de(P_{gsd},P_{lb}), crs2 and crs(P_{gsd}, P_{lm}) using 46 problems

		de		de(P_{gsd},P_{lb})		crs2		crs(P_{gsd}, P_{lm})	
P	n	fe	sr	fe	sr	fe	sr	fe	sr
ACK	10	25810	100	20615	100	17586	100	8146	100
AP	2	683	100	766	100	406	83	533	99
BL	2	991	100	829	100	578	98	553	100
B1	2	977	100	1102	100	819	74	780	100
B2	2	1039	100	1134	100	825	76	782	100
BR	2	1030	100	850	100	465	85	559	100
CB3	2	717	100	766	100	554	97	521	100
CB6	2	976	100	947	100	615	97	583	100
CM	4	2300	100	2610	100	2266	100	1271	100
DA	2	1277	100	1335	100	789	63	945	100
EP	2	650	95	857	100	487	98	562	99
EM	10	249958	85	532909	95	0	0	0	0
EXP	10	9903	100	7750	100	6425	100	3139	100
GP	2	937	100	950	100	671	88	649	99
GW	10	14705	100	11793	100	10037	100	4788	100
GRP	3	3024	98	2100	98	977	100	1071	100
H3	3	1220	100	1235	100	908	100	733	100
H6	6	6836	99	5594	99	3993	90	2143	76
HV	3	3782	99	3226	100	1876	100	1702	100
HSK	2	559	100	631	100	467	100	443	100
KL	4	1839	100	1180	97	519	100	530	40
LM1	3	1465	100	1654	100	1377	100	959	100
LM2	10	11583	100	10526	100	8945	100	4046	100
MC	2	639	100	661	100	462	100	459	100
MR	3	3270	74	2632	100	1266	41	1087	43
MCP	4	3672	100	1159	100	577	100	511	100
ML	10	200673	70	121283	69	75780	17	15441	31
MRP	2	1485	64	1194	55	694	39	733	50
MGP	2	1032	68	1050	59	623	25	669	36
NF2	4	80475	100	208111	100	75214	93	6945	100
NF3	10	84620	100	21413	100	12686	100	7460	100
PP	10	14785	100	12300	100	10407	69	5135	100
PRD	2	1862	90	1039	87	497	100	781	67
PQ	4	5346	100	3884	100	2189	100	1733	100
RG	10	98303	100	167173	100	0	0	79447	98
RB	10	265700	2	51089	100	0	0	15219	94
SF1	2	3079	63	2314	27	474	100	712	100
SF2	2	1990	100	1232	100	1503	41	1317	13
SBT	2	2581	100	1515	77	418	7	961	100
SWF	10	28157	100	35693	100	0	0	19142	90
S5	4	5734	95	4465	94	3239	95	1800	55
S7	4	4707	100	4072	100	2973	98	1773	70
S10	4	4788	100	4121	100	3074	100	1758	72
SIN	20	61242	100	26946	100	20788	3	11805	100
ST	9	900090	100	26879	100	10128	100	16395	100
WP	4	14598	96	8193	100	3919	100	3192	100
tr		2131089	4298	1319777	4344	288496	3477	229913	3932

Table 2. Comparison of de, de^1, de^2 and de^3

	de	de^1	de^2	de^3
fe	2131089	1508851	1319777	1802733
sr	4298	4324	4344	4445
cpu	73.85	72.93	63.02	52.59

de^1=de(P_{gsd},$\cdot$), de^2=de(P_{gsd},P_{lb}) and de^3=de(P_{gsd}, P_{lb})$|_{\alpha_1(k)=0}$

that all the de type algorithms were able to solve. While in terms of cpu de($P_{gsd}, \cdot$) and de($P_{gsd}, \cdot$)$|_{\alpha_1(k)=0}$ are comparable, in terms of fe and sr de($P_{gsd}, \cdot$) is superior to de($P_{gsd}, \cdot$)$|_{\alpha_1(k)=0}$ by about 29% and 1% respectively. This clearly demonstrates that the probabilistic combination of the simplex scheme P_{gs} and the differential scheme P_{gd} has improved the original de, especially in terms of reducing fe in de. A comparison of de($P_{gsd}, \cdot$) with de(P_{gsd}, P_{lb}) and de($P_{gsd}, \cdot$)$|_{\alpha_1(k)=0}$ with de(P_{gsd}, P_{lb})$|_{\alpha_1(k)=0}$ shows that P_{lb} has got a great effect both in reducing fe and cpu while increasing sr. For example, de(P_{gsd}, P_{lb}) is superior to de($P_{gsd}, \cdot$) in all respects. Similarly, a comparison between de($P_{gsd}, \cdot$)$|_{\alpha_1(k)=0}$ and de(P_{gsd}, P_{lb})$|_{\alpha_1(k)=0}$ shows that de(P_{gsd}, P_{lb})$|_{\alpha_1(k)=0}$ is superior to the original de with respect to fe, sr and cpu. This clearly shows the incorporation of P_{lb} improves both the original de and de($P_{gsd}, \cdot$).

Results from Table 2 show that P_{lb} on its own improved the original de significantly in terms of sr and cpu. On the other hand, the introduction of P_{gs} in de significantly reduced the fe, with about 29% compared to 15% reduction of fe due to P_{lb} alone. The shows that the probabilistic adaptation greatly improves de in terms of fe without compromising its robustness in sr. Therefore, the introduction of probabilistic adaptation of P_{gs} and P_{gd} in de is fully justified.

To answer the second question, we compare the crs2 and crs(P_{gsd},P_{lm}) using results from Table 1 to show the effects of P_{gd} and P_{lm} in crs2. The total results from Table 1 show that in terms of sr, crs(P_{gsd}, P_{lm}) is much superior to crs2 with about 455 more successes. In terms of fe, we see that crs(P_{gsd}, P_{lm}) is also superior to crs2 with about 20% less fe. This is so despite the fact that crs(P_{gsd}, P_{lm}) solved three more 10 dimensional difficult problems (RG, RB and SWF) compared to crs2. Moreover, if we compare these two algorithms based only on those problems that both solved, we see that crs(P_{gsd}, P_{lm}) becomes much superior to crs2 both in terms of fe and sr, with 173 more successes for about 60% less fe. This shows that the introduction of P_{gd} together with P_{lm} has a positive effect in fe and sr.

To see the effects of the new features, P_{gd} and P_{lm} separately in crs2, we compare crs2 with crs($P_{gsd}, \cdot$), crs($P_{gsd}, \cdot$) with crs(P_{gsd}, P_{lm}) and crs2 with crs(P_{gsd}, P_{lm})$|_{\alpha_1(k)=1}$ respectively using the total results for only those problems that were solved by all the algorithms, i.e. excluding the three difficult problems that were not solved by crs2. These results, together with results for crs4, the best known crs algorithm [9], are presented in Table 3. The crs2, crs4 and crs(P_{gsd}, P_{lm})$|_{\alpha_1(k)=1}$

algorithms solved 42 problems while the crs($P_{gsd}, \cdot$) and crs(P_{gsd}, P_{lm}) solved 45 problems. It is therefore clear that the probabilistic based algorithms solved more problems. We begin with the effect of P_{gd} in crs2. crs($P_{gsd}, \cdot$) and crs2 are identical for $\alpha_1(k) = 1$. In terms of fe and cpu, crs2 is much superior to crs($P_{gsd}, \cdot$). However, crs($P_{gsd}, \cdot$) is much superior to crs2 in terms of sr with 246 more successes. Moreover, crs($P_{gsd}, \cdot$) solved three more difficult problems than crs2. This shows that although P_{gd} seems to have a negative effect in terms of fe and cpu in crs2, it improves the robustness of crs2 in terms of success rate and the number of problems solved. We now compare crs($P_{gsd}, \cdot$) with crs(P_{gsd}, P_{lm}) to see the effect of local mutation. A comparison of crs($P_{gsd}, \cdot$) with crs(P_{gsd}, P_{lm}) shows that P_{lm} reduced fe and cpu of crs($P_{gsd}, \cdot$) significantly by about 74% and 67% respectively. However, P_{lm} reduced slightly the sr of crs($P_{gsd}, \cdot$) by 73 successes. To see the effect of P_{lm} in crs2, we compare it with crs(P_{gsd}, P_{lm})$|_{\alpha_1(k)=1}$. We see from Table 3 that crs(P_{gsd}, P_{lm})$|_{\alpha_1(k)=1}$ is much superior to crs2 in terms of fe, sr and cpu. crs(P_{gsd}, P_{lm})$|_{\alpha_1(k)=1}$ has about 63% less fe and 48% less cpu for about 11% more sr compared to crs2. crs(P_{gsd}, P_{lm})$|_{\alpha_1(k)=1}$ shows greater improvements than the other two crs algorithms with probabilistic adaptation. However, if we compare the algorithms based on all the problems solved then the probabilistic algorithms are superior in terms of fe and sr.

Table 3. Comparison of crs2, crs2^1, crs2^2 ,crs^3 and crs4

	crs2	crs2^1	crs2^2	crs2^3	crs4
fe	288496	449643	116105	105902	208954
sr	3477	3723	3650	3855	3756
cpu	6.45	11.04	3.64	3.37	4.86

crs2^1=crs($P_{gsd}, \cdot$), crs2^2=crs(P_{gsd}, P_{lm}) and crs2^3=crs(P_{gsd}, P_{lm})$|_{\alpha_1(k)=1}$

We also compare crs4 with crs(P_{gsd}, P_{lm}). crs4 is a modified crs2 algorithm that uses a β-distribution as a local technique to explore the region around the best point x_l in S. We see from Table 3 that in terms of fe and cpu, crs(P_{gsd}, P_{lm}) is much superior to crs4. However, crs4 obtained more sr than crs(P_{gsd}, P_{lm}). A comparison between crs4, crs(P_{gsd}, P_{lm}) and crs(P_{gsd}, P_{lm})$|_{\alpha_1(k)=1}$ (all have a local technique) shows that in terms of fe, cpu and sr, crs(P_{gsd}, P_{lm})$|_{\alpha_1(k)=1}$ is the best performer while crs(P_{gsd}, P_{lm}) is runner-up in terms of fe and cpu. However, crs(P_{gsd}, P_{lm}) solved more problems than the other local search based crs algorithms. Therefore, the incorporation of the probabilistic adaptation and local mutation in crs2 is fully justified.

Finally, we show how the probabilistic adaptation takes place within an algorithm. We use the algorithm de(P_{gsd},$\cdot$) to demonstrate the probabilistic adaptation. We present three figures to illustrate these adaptations. The figures have been plotted using the number of iterations k as the horizontal axis and values of $\alpha_1(k)$ as the

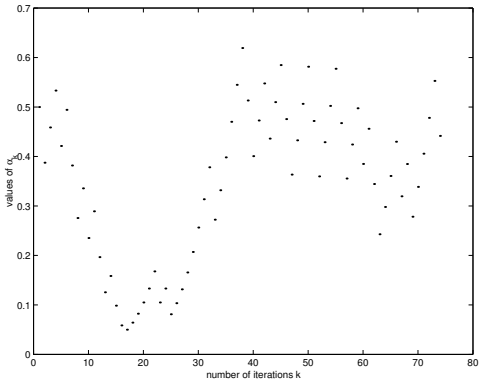


Figure 1. Probabilistic adaptation of de(P_{gsd},·) on EXP

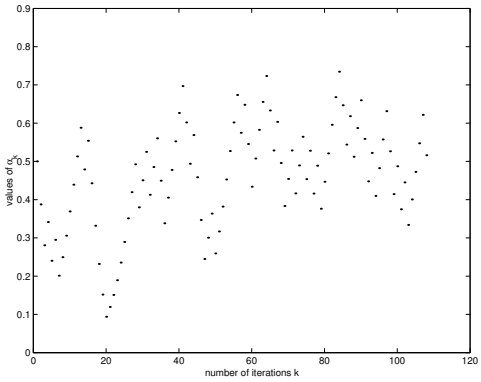


Figure 2. Probabilistic adaptation of de(P_{gsd},·) on HV

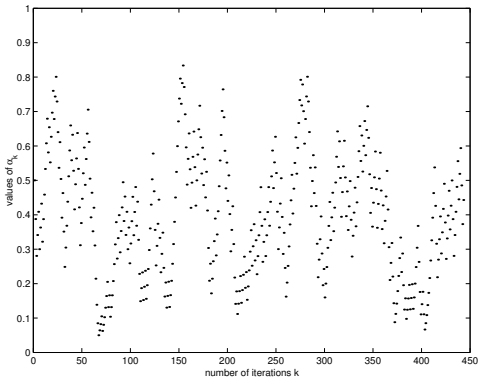


Figure 3. Probabilistic adaptation of de(P_{gsd},·) on RB

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.

We use figures 1 - 3 to illustrate the probabilistic adaptation of P_{gs} and P_{gd} within de(P_{gsd},·). Figures 1 - 3 are respectively for the Exponential (EXP), Helical Valley (HV) and Rosenbrock (RB) problems. The de(P_{gsd},·) algorithm uses the simplex scheme P_{gs} for $\alpha_1(k) = 1$, differential scheme P_{gd} for $\alpha_1(k) = 0$ and a mixture of the two for any $\alpha_1(k) \in (0, 1)$. We see from Figure 1 that for the first 40 iterations, the de(P_{gsd},·) algorithm generates trial points using mostly the differential scheme before switching to almost equal probabilities for the two schemes. From Figure 2, we see that for the Helical Valley problem, on average, the algorithm uses two schemes almost equally in generating the trial points. In Figure 3, we see that the algorithm on average used mostly the differential scheme. However there are cases where $\alpha_1(k)$ reached 0.8, thus favouring the simplex scheme occasionally. Looking at Table 1, we see that this is one problem where both de and crs2 had difficulties in solving. This shows how the probabilistic adaptation aids an algorithm in using the strengths of the two schemes to solve a problem where each scheme on its own had difficulties.

From the above discussions using results and figures, it is clear that the probabilistic adaptation has a significant role to play in both algorithms. It can be seen from the results that the differential scheme, P_{gd}, mainly improves the robustness of crs2 in terms of finding the global minimum. On the other hand, the simplex scheme, P_{gs}, mainly improves the effectiveness of de in terms of the number of function evaluations. We also see that the localizations have a great effect in improving the rate of convergence of the algorithms in terms of both fe and cpu. Convergence of purely heuristic algorithms depends on the effort, in terms of the number of function evaluations and cpu time, the algorithms need to solve problems. Thus, heuristic algorithms prove their usefulness by numerical testing. We see from the results that the new algorithms de(P_{gsd},P_{lb}) and crs(P_{gsd},P_{lb}) needed less number of fuction evaluations and cpu time to reach the global minimum. Therefore, the new algorithms converge faster than their old counterparts.

5 Conclusion

We have introduced a new version of the de algorithm and a new version of the crs algorithm. There are a number of algorithms that can be derived from each new algorithm. The original version of each algorithm is a special case of the new one. The new algorithms were tested on 50 benchmark test problems. The results show that the new de and crs are considerably superior to the original de and crs2 respectively. We have shown the effects of the point generation schemes as well as the effects of the local technique in both the new algorithms. Some examples were also presented

to show how the probabilistic adaptation guides an algorithm. Further research is underway in developing even more efficient de and crs for large dimensional problems.

References

[1] Michalewicz, Z., 1996, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, Berlin.
[2] Ali, M. M. and Törn, A., 2004, Population set based global optimization algorithms : some modifications and numerical studies. Computers and Operations Research, **31 (10)**, 1703-1725.
[3] 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.
[4] Price, W. L., 1977, A controlled random search procedure for global optimization. Computer Journal,**20**, 367-370.
[5] Price, W. L., 1983, Global optimization by controlled random search. Journal of Optimization Theory and Applications, **40**, 333-348.
[6] Kaelo, P., 2005, Some population set based methods for unconstrained global optimization. PhD thesis, to be submitted.
[7] Nelder, J. A. and Mead, R., 1965, A simplex method for function minimization. Computer Journal, **7**, 308-313.
[8] Price, W. L., 1987, Global optimization algorithms for a CAD workstation. Journal of Optimization Theory and Applications,**55**, 133-146.
[9] Ali, M. M. and Storey, C., 1994, Modified controlled random search algorithms. Int. J. of Computer Mathematics, **54**, 229-235.
[10] Ali, M. M., Törn, A. and Viitanen, S., 1997, A numerical comparison of some modified controlled random search algorithms. Journal of Global Optimization, **11**, 377-385.
[11] 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.
[12] Bressloff, P. C., 1995, Stochastic dynamics of reinforcement learning. Network: Computation in Neural Systems **6**, 289-307.
[13] Törn, A., Ali, M. M. and Viitanen, S., 1999, Stochastic global optimization: problem classes and solution techniques, Journal of Global Optimization **14**, 437-447.
[14] Ali, M. M., Khompatraporn, C. and Zabinsky, Z. B., 2004, A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. Journal of Global Optimization. To appear.