Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers & Operations Research 31 (2004) 1703–1725
computers & operations research
www.elsevier.com/locate/dsw

Population set-based global optimization algorithms: some modifications and numerical studies☆

M.M. Ali[a,*], A. Törn[b]

[a] *School of Computational and Applied Mathematics, Witwatersrand University, Private Bag 3, Wits 2050, Johannesburg, South Africa*
[b] *Department of Computer Science, Åbo Akademi University, Turku, Finland*

Abstract

This paper studies the efficiency and robustness of some recent and well known population set-based direct search global optimization methods such as Controlled Random Search, Differential Evolution and the Genetic Algorithm. Some modifications are made to Differential Evolution and to the Genetic Algorithm to improve their efficiency and robustness. All methods are tested on two sets of test problems, one composed of easy but commonly used problems and the other of a number of relatively difficult problems.

Keywords: Global optimization; Direct search method; Controlled random search; Differential evolution; Genetic algorithm; Continuous variable

1. Introduction

Direct search methods [1] are widely used in applied science and in engineering. They are a class of optimization methods which are easy to program, do not require any properties of the function $f(x)$, $x \in \Omega \subset R^n$ (Ω is assumed to be defined by specifying upper and lower limits of the domain of each variable), being optimized and are often claimed to be robust for problems with noisy function values. Hence, when the optimizing function is nonlinear, non-differentiable and non-smooth, direct search methods are the methods of choice. Over the years, several direct search methods for solving the global optimization problems for continuous variables x have been proposed. These are based on

☆ The paper was revised when the first author was spending his sabbatical leave at IMA, University of Minnesota, USA. Financial support from IMA is acknowledged.
* Corresponding author. Tel.: +27-11-717-6139; fax: +27-11-717-6143.
E-mail address: mali@cam.wits.ac.za (M.M. Ali).

0305-0548/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/S0305-0548(03)00116-3

global exploration (search of Ω) and localization of search. Some well-known direct search methods are the Downhill Simplex (DS) method of Nelder and Mead [2], the method of Hooke and Jeeves [3]. These methods are really local optimization methods and are usually used in a repeated fashion from random starting points when used on multi-modal problems. These methods have been used in many industrial and scientific applications, particularly by the engineering community.

A different strategy to solve multi-modal problems is used by population set-based methods. Here a set S of initial samples in Ω is successively transformed into samples concentrated on the global minimum. The basic population set-based methods can be described as follows:

- generate the initial set S randomly in Ω;
- iteratively replace points in S with better points;
- stop when some stopping condition is met.

One of the population set-based direct search methods is the Controlled Random Search (CRS) algorithm of Price [4,5]. Although this method proved very robust in many applications [6,7], it is somewhat less known to the engineering community. Another popular population set-based algorithm is the genetic algorithm (GA) [8]. Most recently, Storn and Price [9] proposed a new population set-based direct search method, the differential evolution (DE) Method.

The objective of this paper is to describe the three population set-based methods using the same general algorithm and in this way show the similarity of them and how they differ from each other. Further, we will investigate the improvement in efficiency and robustness of modifications to DE and to GA.

To this end, first the robustness and efficiency of these direct search techniques, their strengths and weaknesses, are thoroughly investigated by their implementation on classes of test problems, classified as easy, moderately difficult and hard problems [10]. Second, we propose modifications to DE [9] and to GA [11,12] to improve their robustness.

In Section 2 we briefly describe all CRS, GA and DE algorithms. Following the descriptions, we introduce our proposed modifications to GA and DE in the respective subsections of Section 2. The test problems considered for comparison purpose are briefly discussed in Section 3. Numerical results and comparisons are made in Section 4. Section 5 contains a discussion and conclusions based on the results obtained by different algorithms. Similarities and dis-similarities of all population set-based methods are presented in Appendix A. A set B of test problems are given in Appendix B.

2. Three classes of population set-based algorithms

All population set-based direct search methods use a population set S. The initial S consists of N points with corresponding function values. A contraction process is then used to drive these points to the vicinity of the global minimizer. Different population set-based method uses different strategies in the contraction process, per generation. For instance, the CRS algorithm makes use of simplexes for generating alternatives to replace a single sample (worst point) in the set S. In GA a subset of S is successively selected for which mutations and crossovers are used to generate m_1 new samples to replace m_1 old samples (bad points) of S. Unlike CRS which attempts to replace a single point in S per generation, and GA which replaces m_1 points (parents) of S by the new m_1 points (children) per

generation, DE attempts to replace all points in S by new points at each generation. Here, mutation and crossover are used to generate trial points. A point-to-point comparison is then made and better trial points are accepted.

2.1. CRS algorithms

Several versions of the CRS algorithm can be found in the literature; these are various modifications made to the original controlled random search (CRS1) algorithm. The first two improvements (CRS2 and CRS3) [13,14] was proposed by Price himself. Subsequently, Ali and Storey [15] presented CRS4 and CRS5, Mohan and Shanker [16] CRSI and Ali et al. [17] proposed CRS6. In the original version, CRS1, the search region Ω is sampled and then a simplex is formed from a subset of this sample. One of the points of the simplex is reflected in the centroid of the remaining points (as in the Nelder and Mead) to obtain a new trial point. If the new trial point is better than the current worst point in S then it replaces the worst point. The process of forming a new simplex and generating the trial point is then repeated until some stopping condition is met. In the second version, CRS2, a more sophisticated use is made of the simplexes in obtaining new trial points. In the third version, CRS3, a Nelder–Mead-type local search (DS) is incorporated. In versions CRS4 and CRS5 further local techniques are introduced. The version CRS6 incorporates a local technique with a quadratic interpolation. The use of the quadratic interpolation replacing the simplexes within the framework of CRS (CRSI) is also proposed in Mohan and Shanker [16]. It is easy to describe all CRS algorithms based on the description of CRS1.

Algorithm 1. The CRS1 algorithm.

Step 1: Determine the initial set S

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

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

Step 2: Determine best, worst point in S. Determine the points $x_{\max}, x_{\min}$ and their function values $f_{\max}, f_{\min}$, such that

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

If the stopping condition (e.g., $f_{\max} - f_{\min} < \varepsilon$) is satisfied, then stop.

Step 3: Generate point to replace point in S. Choose randomly $n+1$ distinct points $x_{p(1)}, x_{p(2)}, \ldots, x_{p(n+1)}$ from S (selection) and compute (by, say mutation)

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

where the centroid G is given by

$$G = \frac{1}{n} \sum_{j=1}^{n} x_{p(j)}. \tag{2}$$

If $\tilde{x} \notin \Omega$ go to Step 3; otherwise compute $f(\tilde{x})$. If $f(\tilde{x}) \geqslant f_{\max}$ then go to Step 3.

Step 4: Update S. Update S by

$$S = S \cup \{\tilde{x}\} - \{x_{\max}\},$$

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

Notice that the core operation, i.e., the operation through which the trial points are generated in the CRS1 algorithm is in Step 3. Therefore, all other CRS algorithms are derived by modifying this step.

The CRS algorithms, CRS2–5, use simplexes but they differ from CRS1 in that they make a random selection of n points in S and include the current best point bringing the number of points up to $n+1$. Therefore, the modification of Step 3 is that n distinct points $x_{p(2)}, x_{p(3)}, \ldots, x_{p(n+1)}$ are sampled from S and that G is calculated by

$$G = \frac{1}{n}\left(x_{\min} + \sum_{j=2}^{n} x_{p(j)}\right). \tag{3}$$

The CRS2–5 algorithms then calculate a trial point using (1). CRS1 becomes CRS2 by simply replacing (2) with (3). Each of CRS3–5 has an extra local feature added to the global feature of CRS2. For instance, CRS3 replaces (2) with (3) but each time (1) in Step 3 finds a new $x_{\min}$, CRS3 incorporates a Nelder–Mead-type local search from the best $n+1$ points of the set S. CRS4 and CRS5 also implements (3) but instead of using a Nelder–Mead-type local search whenever a new $x_{\min}$ is found, CRS5 uses a gradient-based local search from $x_{\min}$, and CRS4 evaluates f in r points (e.g., $r = 4$) from the β-distribution using the current $x_{\min}$ as its mean and the distance between $x_{\min}$ and $x_{\max}$ as standard deviation. CRSI simply uses a quadratic interpolation in Step 3 to generate the trial point $\tilde{x}$. The quadratic interpolation uses $a = x_{\min}$ and two other randomly selected points $\{b, c\}$ from S to determine the coordinates of the trial point $\tilde{x} = (\tilde{x}^1, \ldots, \tilde{x}^n)$, where

$$\tilde{x}^i = \frac{1}{2}\frac{(b^{i2} - c^{i2})f(a) + (c^{i2} - a^{i2})f(b) + (a^{i2} - b^{i2})f(c)}{(b^i - c^i)f(a) + (c^i - a^i)f(b) + (a^i - b^i)f(c)}. \tag{4}$$

CRS6 also uses the quadratic interpolation but as soon as the trial point $\tilde{x}$ becomes the best point in S it uses β-distribution as in CRS4.

To summarize, CRS1 uses (1) together with (2) to generate trial points away from $x_{p(n+1)}$ in the direction of the centre of gravity of the remaining n point, whereas CRS2 uses (1) with (3) to generate points in the direction of the centre of gravity of n points including the best point, $x_{\min}$. CRS3–5 includes localization features added onto the point generation feature of CRS2, i.e., each time rule (1) and (3) produces a best point a local exploration feature is applied. This local feature is different for different algorithms. CRS6 and CRSI uses the quadratic interpolation method to generate trial points. This interpolation method uses the best point in the set S. A localization feature is added to CRS6 but not in CRSI. We use Table 1 to distinguish between the CRS algorithms.

Remark 1. The three main features of CRS are (i) selection, (ii) reproduction and (iii) acceptance. The selection of n points for CRS2–5, $n+1$ for CRS1 and 2 points for CRS6/CRSI is random. Reproduction consists of mutation only, e.g., the operation defined by (1), for CRS1–5, a single

Table 1
Local and global features of CRS

Algorithm	Local and global features
CRS1	Global feature only: use (1) and (2)
CRS2	Global feature only: use (1) and (3)
CRS3	Global feature: use (1) and (3); local feature DS
CRS4	Global feature: use (1) and (3); local feature: β-distribution
CRS5	Global feature: use (1) and (3); local feature: Gradient based search
CRSI	Global feature only: use (4)
CRS6	Global feature: use (4); local feature: β-distribution

interpolation for CRSI and the interpolation and β distribution for CRS6. Acceptance of the trial point is not compulsory.

2.2. Genetic algorithms (GA)

The genetic algorithm is a global optimization technique based on natural selection and the genetic reproduction mechanism. Like CRS, GA maintains a set S of candidate solutions, where each solution is coded as a binary string known as chromosome. Central to GA is the natural evolution of the set S. At each generation of GA a new S evolves from the old S, i.e., each generation updates the set S. As the generation proceeds, the set of solutions in S converges to the global minimum. In the basic GA, three steps are involved in the evolution from one generation to the next. These are:

- evaluation of f at each new member of the current set S;
- stochastic selection of points (parents) from the current set S with a bias in the selection towards better points;
- reproduction of new points (children) from the selected points (parents) using the two genetic operations: crossover and mutation. The crossover operation is achieved by taking two selected points, cutting the strings at random index and exchanging parts. Mutation is achieved by simply flipping the bit at some random index.

This cycle of evaluation, selection and reproduction terminates when a convergence criterion is met.

2.2.1. Genetic algorithm (GA1) for function optimization

The binary coded GA can be modified for function optimization [18,19]. Recent applications of GA on function optimization, however, have shown that the real coded GA is superior to its conventional binary coded counterpart [11,12,19]. A real-coded GA treats chromosomes as vectors (points) of real valued numbers and it adapts the genetic operators accordingly. Several versions of real coded GA have been proposed. Yen and Lee [20] proposed a real coded GA which combines GA with a stochastic variant of the DS method. Recently, a hybrid algorithm comprising of GA and DS was proposed by Yang and Douglas [21]. One main problem with these methods is that at each generation the whole population needs rank ordering in accordance with the function values, which is expensive in terms of cpu time even for a moderate population size. Moreover, numerical

investigations with a wide range of problems were not carried out. Most recently, Hu et al. [11,12] has proposed a new version of real coded GA. Numerical studies were carried out on a relatively large set of test problems with the conclusion that the convergence of the algorithm is rapid.

In this implementation the crossover is done arithmetically, i.e., given the parents $x=(x^1,x^2,\ldots,x^n)$ and $y=(y^1,y^2,\ldots,y^n)$ one calculates:

$$\tilde{x}^i=\alpha_i x^i+(1-\alpha_i)y^i,\quad \tilde{y}^i=\alpha_i y^i+(1-\alpha_i)x^i,\quad i=1,2,\ldots,n, \tag{5}$$

where α_i are uniform random numbers in $[-0.5,1.5]$. The crossover rule defined by (5) does not maintain the convexity property, but it is exploratory. For instance, points will be generated not only on the lines joining x and y but around the line as well as further sides of both the points. Mutation for a string is carried out by setting

$$\tilde{x}^i=\tilde{x}^i+\gamma\times(U^i-L^i), \tag{6}$$

for a randomly selected index i and $\gamma=0.1$. Here, U^i and L^i are the upper and lower bound of the element x^i. An important factor in GA is the selection scheme used. A tournament selection scheme with three players was found robust [11,12]. Therefore, rank ordering is no longer needed in this implementation of GA. Here we restrict ourselves to the real coded GA and in particular to the algorithm of Hu et al. [11,12].

Although the real coded genetic algorithm (GA1) proposed by Hu et al. [11,12] is noise tolerant and claimed to be robust some aspects of this GA need to be addressed. For instance, the first drawback is that the selection procedure used was the so-called tournament selection where each time the best of three randomly chosen parents was selected, and the process continued until m_1 parents were selected; m_1 children were then produced from these m_1 parents using crossover and mutation. We believe that this selection criterion is too biased towards the better points and therefore it is too greedy and may only work for extremely easy problems. Indeed, GA1 was tested on a large class of test problems, and all these problems are easy to solve as shown in a recent study [10]. The second drawback is that the mutation operator is rather randomly chosen, in particular, the choice of γ. If the difference between U^i and L^i is high then this will create a point far away from the point generated by crossover operation. This point will be accepted by the mandatory acceptance rule and it has a particular disadvantage of slowing down the convergence at the later stage of the algorithm when the N points are scattered around the global minimizer. Moreover, comparison of GA was made with the CRS6 algorithm only. Since the results of their paper were given only for the parallel implementations of GA and CRS, comparisons made by the authors [11,12] were not systematic. For instance, the population sizes used in the algorithms were not equal. We address these issues in the next section and propose a modification to GA1. We will also compare GA1 with other algorithms in a systematic manner using both easy and difficult problems. We now describe a generic real coded GA.

Algorithm 2. The genetic algorithm (GA1).

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

Step 2: Determine best, worst point in S. Same as in Algorithm-1.

Step 3: Generate points to replace points in S.

- Selection: select $m_1 \leqslant N$ points from S as parents.
- Crossover: pair the points (parents) and generate m_1 new points (offsprings), which replaces the m_1 worst points (least fittest strings) in S.
- Mutation: mutate an element of each point (string) with probability p_μ, say $p_\mu = 0.001$. Mutation is repeated if the mutated solution is infeasible.

Step 4: Update S. Update S, set $k := k + 1$ and go to Step 2.

The number of children, m_1, to be created per generation will affect the performance of the GA. For the sequential GA, a smaller m_1, e.g., the nearest even integer to $0.1N$, is suggested [12,21].

Remark 2. Selection of m_1 points is biased towards the better points. Reproduction consists of crossover and mutation. Acceptance is compulsory as the m_1 children replaces the m_1 parents in each generation.

2.2.2. The modified genetic algorithm (GA2) for function optimization

To address the first drawback we replace the tournament selection with a random selection. In addition to the crossover rule (5), a learning component is introduced in the crossover phase to make GA more exploratory. As in GA1, children are created two at a time in GA2. However, unlike GA1 where two children are created using two parents obtained by using tournament selection GA2 creates two children from randomly selected $n+2$ parents, $x_{p(1)}, x_{p(2)}, \ldots, x_{p(n+1)}, x_{p(n+2)}$, from S. The selected $n+2$ points are used to calculate the centroid G of the n points remaining after excluding the two worst points, say $x_{p(n+1)}$ and $x_{p(n+2)}$. The first child is then taken as the best point of $\{\hat{x}_1, \hat{x}_2\}$, where

$$\hat{x}_1 = 2 \times G - x_{p(n+1)} \quad \text{and} \quad \hat{x}_2 = 2 \times G - x_{p(n+2)}. \tag{7}$$

If the jth point ($j = 1, 2$) $\hat{x}_j$ is not in Ω then it is calculated as $\hat{x}_j = \frac{1}{2}(G + x_{p(n+j)})$. The second child is found from the best of $\{\hat{x}_3, \hat{x}_4\}$, where $\hat{x}_3$ and $\hat{x}_4$ are obtained using the crossover rule (5). This crossover is carried out between two parents selected randomly from the $n+2$ points, again excluding two worst points, say $x_{p(n+1)}$ and $x_{p(n+2)}$. If the trial points fall outside Ω, random selection of $\alpha_i \in [-0.5, 1.5]$ continues until $\hat{x}_3, \hat{x}_4 \in \Omega$. For the next pair of children $n+2$ points are again selected randomly from S and the above process continued. The m_1 children are therefore generated two at a time. Although the above-mentioned rules take two extra function evaluations for each pair generated, this rule has proved robust and exploratory and therefore a good alternative to the tournament selection. The mutation rate was set to $p_\mu = 0.001$ for both GA1 and GA2. For each child, x_j generated the mutation is carried out with probability p_μ from a randomly chosen component x_j^i of x_j. We ran into trouble when using (6), especially for problems with large Ω. For such problems, points determined by (6) frequently fell outside Ω or move far away from the points in S, delaying the convergence. Therefore, we carried out numerical studies to empirically obtain the value of γ in (6). A good value of γ is found to lie randomly on an interval. In particular, we modify the mutation operator (6) to

$$x^i = x^i + \gamma \times (U^i - L^i), \tag{8}$$

where γ is a random number in $[-0.01, 0.01]$. This mutation rule was used for both GA1 and GA2.

2.3. Differential evolution (DE)

DE [9] is a population set-based algorithm designed for minimizing a function of real variables. It is extremely robust in locating the global minimum. The overall structure of DE resembles that of CRS and GA. Like the other population set-based direct search methods DE also attempts to guide an initial set $S=\{x_1,x_2,\ldots,x_N\}$ of points in Ω to the vicinity of the global minimum through repeated cycles of selection, reproduction (mutation and crossover) and acceptance, see Algorithm 3. However, unlike CRS which attempts to replace a single point in S per generation, and GA which replaces m_1 points (parents) of S by the new m_1 points (children) per generation, DE attempts to replace all points in S by new points at each generation. We now describe how this is done. In each generation, N competitions are held to determine the members of S for the next generation. 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 point $\hat{x}_i$ 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:

$$\hat{x}_i = x_{p(1)} + F \times (x_{p(2)} - x_{p(3)}), \tag{9}$$

where $F \leqslant 1$ is a scaling factor. The trial point y_i is found from its parents x_i and $\hat{x}_i$ using the following crossover rule:

$$y_i^j = \begin{cases} \hat{x}_i^j & \text{if } R^j \leqslant C_R \text{ or } j = I_i, \\ x_i^j & \text{if } R^j > C_R \text{ and } j \neq I_i, \end{cases} \tag{10}$$

where I_i is a randomly chosen integer in the set I, i.e., $I_i \in I = \{1,2,\ldots,n\}$; the superscript j represents the jth component of respective vectors; $R^j \in (0,1)$, drawn randomly for each j. The entity C_R is a constant (e.g., 0.5). The ultimate aim of the crossover rule (10) is to obtain the trial vector y_i with components coming from the components of target vector x_i and mutated vector $\hat{x}_i$. And this is ensured by introducing C_R. Notice that for $C_R = 1$ the trial vector y_i is the replica of the mutated vector $\hat{x}_i$, i.e., only the mutation operation is used in reproduction. The effect of this will be studied later. The acceptance mechanism follows the crossover. 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. The process continues until all members of S are targeted. The stopping condition in Step 2, can for all algorithms be defined as:

$$f_{\max} - f_{\min} \leqslant \varepsilon, \tag{11}$$

where ε is a small number, say $\varepsilon = 10^{-6}$.

Algorithm 3. The DE algorithm

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

Step 2: Determine best, worst point in S. Same as in Algorithm 1.

Step 3: Generate points to replace points in S. For each $x_i \in S$, determine y_i by the following two reproduction operations:

- Mutation: Randomly select three points from S except x_i, the running target and find the second parent $\hat{x}_i$ by the mutation rule (9).
- Crossover: Calculate the trial vector y_i corresponding to the target x_i from x_i and $\hat{x}_i$ using the crossover rule (10).

Step 4: Replace points in S. Select each trial vector y_i for the $(k+1)$th generation 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.

Design of DE certainly brought a new dimension to the direct search techniques in the field of global optimization. However, we believe that some aspects of DE need addressing. First, the sensitivity of the parameters F and C_R was not studied, authors just chose their values randomly with F mainly between 0.5 and 0.9 and C_R between 0.1 and 0.9. Second, the implementation of DE required some estimate of the global minimum to be provided; which is not possible for an arbitrary function. Third, although the authors claim the robustness of DE, comparisons of DE with other credible direct search methods of similar kind, such as CRS and GA were not made. And finally, the main difficulty with the DE algorithm appears to lie in the slowing down of the convergence as the region of global minimum is approached. We try to address these problems in the next section.

Remark 3. Selection of 3 points used in mutation is random. Reproduction consists of crossover and mutation. Point-to-point comparisons are made for acceptance and thus the acceptance is not compulsory. Notice that the trial point corresponding to the best point in S will be rejected if the corresponding function value at the trial point is not better than $f_{\min}$, but even if it is better than the function value at second best point in S. The point-to-point comparison makes the re-search procedure exploratory.

2.3.1. The modified DE algorithms: DEPD, DE1-3

To overcome some of the above-mentioned drawbacks of DE and to make DE more efficient we propose some modifications to the original DE. In particular, we introduce an auxiliary set S_a of N points alongside S, propose a rule for calculating the scaling factor F automatically and use the pre-calculated differential vectors $x_{p(2)} - x_{p(3)}$ in (9). We also empirically obtain an optimal value for the C_R. These will now be discussed.

Population sets: Instead of working with one population set S, we use two sets, S and S_a. The reason for this is to make use of potential trial points which are normally rejected in DE. It has been shown in our earlier work [22] that the introduction of S_a increases the exploration of the search in the case of a very practical large-scale global optimization problem. Initially, two sets each containing N points are generated in the following way; iteratively sample two points from Ω, the best point x_i going to S and the other x_i' to S_a. At each generation, if the trial point y_i, corresponding to the target x_i, does not satisfy the greedy criterion $f(y_i) < f(x_i)$ then the point y_i is not abandoned altogether, rather it competes with its corresponding target in the set S_a. If $f(y_i) < f(x_i')$ then y_i replaces x_i' in the auxiliary set S_a. Some good points from S_a can then be used to replace some bad points in S periodically.

The scaling factor: The calculation of F is based on the demand that the search be diversified at early stages and intensified at latter stages. Therefore, the following scheme is proposed:

$$F = \begin{cases} \max\left(l_{\min}, 1 - \left|\dfrac{f_{\max}}{f_{\min}}\right|\right) & \text{if } \left|\dfrac{f_{\max}}{f_{\min}}\right| < 1, \\ \max\left(l_{\min}, 1 - \left|\dfrac{f_{\min}}{f_{\max}}\right|\right) & \text{otherwise,} \end{cases} \tag{12}$$

ensuring that $F \in [l_{\min}, 1)$. $f_{\max}$ and $f_{\min}$ are respectively the maximum and minimum values in S and $l_{\min}$ is a lower bound for F.

The pre-calculated differentials (PD): DE proposed by Storn and Price generates differential vectors (one for each targeted point) in the mutation phase at each generation of the algorithm. Therefore, N differential vectors will be calculated by (9) in each generation. Although mean of the distribution of such differentials is always zero (since $x_{p(2)} - x_{p(3)}$ and $x_{p(3)} - x_{p(2)}$ occur with equal frequency) these differentials vectors will gradually be shorter and shorter as the points in S become closer and closer. And this has two effects: (i) calculation of N differentials at each generation makes DE time consuming and (ii) it (calculation of the differential vectors at each generation) limits the exploratory feature of DE. The motivation of PD is to make DE faster and exploratory, and one way this could be achieved is to use pre-calculated differentials in the mutation operator.

We now explain how the pre-calculated differentials are used in DE. At the very first iteration of the algorithm all the differential vectors generated are kept in an array A. In the consecutive generations (say, the next R generations) when the point $x_i \in S$ are targeted, the intermediate point $\hat{x}_i$ is calculated using (9) by choosing a random point $x_{p(1)}$ from S and a random differential from A. This process continues in the mutation phase for R generations of the algorithm before A is updated again with new differentials. Therefore, in this version of DE the mutation operation has two different modes: mode 1 where new differentials are used in (9) (mutation use three distinct vectors from S) and mode 2 where pre-calculated differentials stored in A are used in (9). Mode 1 of the mutation is switched on after every R generations i.e., after every R generations the array A is updated with new differentials. By simply making this change in DE, the algorithm can be made substantially faster and exploratory. This change is implemented in the mutation phase. We call this implementation of DE the differential evolution using pre-calculated differential (DEPD) which replace the Step 3 of Algorithm-3 with:

Step 3: For each $x_i \in S$, determine y_i by the following two operations.

- Mutation: If $(k = 0)$ or $k \equiv 0 (\mathrm{mod}\, R)$ execute mode 1 else mode 2.
- Mode 1: Randomly select three points $x_{p(1)}, x_{p(2)}$ and $x_{p(3)}$ from S except x_i, the running target and find the point $\hat{x}_i$ by the mutation rule (9) using the differential vector $(x_{p(2)} - x_{p(3)})$. If a component $\hat{x}_i^j$ falls outside Ω then it is found randomly in-between the jth lower and upper limits. Update the ith element of the array A with this differential.
- Mode 2: Randomly select a point $x_{p(1)}$ from S and a differential vector from A and find the point $\hat{x}_i$ by the mutation rule (9). If a component $\hat{x}_i^j$ falls outside Ω then it is found randomly in-between the jth lower and upper limits.
- Crossover: Calculate the trial vector y_i corresponding to the target x_i from x_i and $\hat{x}_i$ using the crossover rule (10).

Although we will justify the use of PD in the DE algorithm, it is not our intention to present DEPD as a stand-alone algorithm for our current piece of work. We use PD as an embedded component of some modified DE algorithms whose features are given below.

Features of the modified DE: The modified DE utilizes the potential points of S_a; after each M number of generations (R is not necessarily equal to M) we replace the m_2 worst points in S with the m_2 best points from S_a. The value of M can be chosen as constant or variable. In the case of variable M, the initial value is set to an integer and it is gradually decreased so that the convergence becomes quicker as the points in S converge to the region of the global minimum. In this version of the modified DE the stopping condition remains the same, namely (11), the one which was used for all the methods (CRS, GA and DE) described so far. We call this DE, the DE1 algorithm. We also incorporate a local search in the modified DE which operates from the worst (DE2) or best (DE3) $n+1$ points of S, immediate before the worst (DE2) or the best (DE3) $n+1$ points of S is replaced with the best $n+1$ of S_a. As indicated above we call these modified versions, DE2 and DE3, respectively. Notice that for DE3 even if S contains the best points it gets replaced. The DE2/DE3 algorithm stops when the same minimum is found t times or when (11) is met. We have implemented the Nelder–Mead's DS algorithm as a local search algorithm in DE2 and DE3. The DS algorithm was stopped when $f_{hi} - f_{lo} < 10^{-4}$, f_{lo} and f_{hi}, respectively, being the best and the worst values within $n+1$ function values of DS. With this stopping condition for DS we found that sufficiently good accuracies were obtained for all the problems considered. We also considered two minima as the same when their difference fell within the tolerance of 0.005. We now present the modified DE algorithms.

Algorithm DE*n*. The DE1, DE2 and DE3 algorithms now can be described by the following pseudo Pascal procedure.

begin
initialize S and S_a;
initialize M, M_1, M_2 and R;
find $f_{\min}$ and $f_{\max}$;
calculate F;
$k := 0$;
repeat
  $k := k + 1$;
  call procedure mutation (for $k = 0$ initialize A and update A if (k mod R) $= 0$);
  call procedure crossover;
  call procedure acceptance;
  find $f_{\min}$ and $f_{\max}$;
  calculate F;
  if $((k \bmod M) = 0)$ **then**
  begin
  (For DE2 (DE3) activate local search and keep record of local minima found);
  Replace the worst (best) m_2 points of S with the best m_2 points of S_a;
  $M := M - n$;

if $M <= M_1$ **then** $M := M_1$;
find $f_{\min}$ and $f_{\max}$;
calculate F;
end;
until stopping condition;
end

3. Problems used for comparisons

Performance of all the population set-based direct search algorithms described thus far were judged using two classes of test problems. They are the classes of easy and difficult problems [10]. We consider a set A of nine easy test problems for our study. These are the two-dimensional Goldstein and Price (GP) problem, the four dimensional Shekel family (S5, S7 and S10), the Hartmann family (the three- and six-dimensional Hartmann problems; H3 and H6), the Schubert family (the three- and five-dimensional Schubert problems; P8 and P16) and the 10-dimensional problem of Levy (L10). Although some of these problems have a multitude of local minima, in particular the Schubert family (5^3 and 15^5 minima, respectively) and the problem of Levy (10^{10} minima), a recent study have shown that all these problems are easy to solve and that one should rely not only on these test problems when testing a particular algorithm. Details of some properties of these test problems are given in [10], and will not be repeated here. Besides these, we also consider another set, B, of six relatively difficult test problems. These are the 10-dimensional Extended Rosenbrock (ER10), Schewefel (SF10), Griewank (GW10), the five-dimensional Shekel's Foxholes (FX5), Rastrigin (RG5) and the modified Langerman (ML5). Of these FX5 is a difficult problem and the rest are moderately difficult problems (according to the study done by Törn et al. [10]). The problems of set B are given in Appendix A.

4. Numerical results

In this section the computational results are summarized. Each of the algorithms was run 100 times on each of the test problems to determine the percentages of success. Let TS be the number of runs out of 100 runs that succeeded in locating the global minimum by the respective algorithm. A solution to the problem will of course not be the global minimum f^* exactly, but for instance a value less than f_ε^*, where $f_\varepsilon^* = f^* + \varepsilon$, where $\varepsilon = 10^{-4}$. To even out the stochastic fluctuations in the number of function evaluations and cpu times by computing averages FE and *cpu* for those runs for which the global minima were obtained. The performance is measured by criteria based on FE, *cpu* and TS. All algorithms were run on a SGI-Indy Irix-5.3. Random numbers were generated using the well-tested procedure given in [23]. A common parameter of all population set-based direct search methods is the size N of the population used. Different values for N are suggested for different methods. For example, CRS uses $N = 10(n + 1)$; for DE a value between $5n$ to $10n$ is suggested; for GA no fixed and first rule of choice is given. Therefore, as a compromise we use two values for N, namely $7n$ and $12n$. In an earlier investigation [6,7] of the robustness of some stochastic global optimization algorithms [5,24–26] using test problems and some difficult practical problems, CRS4

Table 2
Comparison of GA1 and GA2 using percentage of TS

	GA1		GA2		
$N \rightarrow$	$7n$	$12n$	$7n$	$12n$	
GP	4	25	44	77	Set *A*
S5	4	14	58	69	
S7	2	22	65	82	
S10	2	26	67	83	
H3	73	94	100	100	
H6	58	100	93	95	
P8	22	66	97	100	
P16	23	79	96	100	
L10	64	96	100	100	
Total	252/900	522/900	721/900	806/900	
ER10	0	0	93	100	Set *B*
RG5	0	4	3	3	
FX5	0	2	1	2	
GR10	0	22	13	27	
SF10	0	4	0	0	
ML5	6	14	36	52	
Total	6/600	46/600	146/600	184/600	

was found to be the best and CRS2 the runner up amongst all of a set of stochastic algorithms. Therefore, in our present investigation of the population set-based direct search methods we consider CRS2, CRS4, CRS6 and CRSI where CRS6 and CRSI are two recent versions of CRS.

4.1. Comparison: GA1 vs GA2

Choice of parameter values: Parameters involved in GA's are the mutation probability p_μ, the number of children m_1 and the model parameters α_i and γ, respectively, in the crossover and mutation rule. Common values of these are used for both GA1 and GA2.

Comparison: In this section we compare GA1 and GA2 using the test sets A and B. We first find the percentage of success (TS) in locating the global minima. Table 2 represents the success rate (%) for 100 runs. Table 2 shows that the reliability of GA2 is superior to that of GA1 for both values of N. It is clear from the total figures that for the test set A, the total number of successes of GA2 is about the double of that of GA1 on the average. Since the performance of GA1 is considerably better for $N = 12n$, we compare the total FE and *cpu* for this value. The total (FE, cpu) is $(13776, 2.06)$ corresponding to the total of 522 successes for GA1, and is $(24108, 4.11)$ corresponding to the total of 806 successes for GA2. However, although GA2 is superior to GA1 in locating the global minimum it needs double (FE, cpu) as compared with that needed by GA1. We, therefore, study further to see if GA1 outperforms GA2 on set A in finding the global minimum given (FE, cpu) nearly the same. For this reason, we ran GA1 on the test set A with increasing

Table 3
Successful runs out of 900 runs for DE using C_R and F

$F \downarrow$	$C_R = 0.25$	$C_R = 0.5$	$C_R = 0.75$	$C_R = 1$
0.45	865	866	861	774
0.65	897	894	887	866
0.85	899	898	886	870
1	896	899	895	658
Total	3557	3557	3529	3168

values of N. For instance, $N = 12n, 15n, 18n, 21n$ and $24n$ and so on. We found that for $N = 24n$ the (FE, *cpu*) was $(23972, 5.75)$ for a total of 701 successes on A. Comparing this with the result of GA2 for $N = 12n$ we see that even though both used more or less the same (FE, *cpu*) GA2 is still superior in TS. In a similar manner, we ran GA2 on the test set A with decreasing values of N. We found that for $N = 8n$, the (FE, *cpu*) was $(13885, 2.86)$ for a total of 754 successes on A. This result $(13885, 2.86)$ is the nearest (FE, *cpu*) needed by GA1 for $N = 12n$ but with less successes (TS=522) in obtaining the global minimum. It is now clear that for the same (FE, *cpu*) GA2 is much superior to GA1 on A in locating the global minimum.

For the test set B the results are $(25174, 9.98)$, TS = 46 for GA1 and $(23904, 6.91)$, TS = 84 for GA2. For set B we must exclude ER10 from the comparison as GA1 has a 0% success rate on ER10. Moreover, GA2 needed on average (FE, *cpu*) = $(494470, 58.87)$ per run for ER10. From this we can confer that GA2 is superior on both sets. Therefore, in comparing the direct search methods later we will only consider GA2.

4.2. Comparison: DE vs DEPD and DE1-3

Choice of parameter values: The original DE has two parameters: F and C_R. We begin here by studying the effect of C_R in DE. Although the effect of calculating F using (12) is shown next, we study the correlation between F and C_R using some sample values. In particular, we take $F = 0.45, 0.65$ 0.85 and 1 and each of these values we conducted a series of DE runs using various values for C_R. For instance, we ran DE for $C_R = 0.25, 0.5, 0.75$ and 1 using $N = 7n$. The results on both sets were in agreement and we present only the results for set A. There were 900 runs on set A and the total number of successes on these runs are presented in Table 3. The last column in Table 3 shows that the higher number of failures has occurred when using $C_R = 1$, i.e., when the trial points are obtained using the mutation rule (9) only. The performance of DE is significantly worse in this case. A comparison of the results under this column with that of columns 2–4 establishes the fact that both the crossover and mutation are necessary in obtaining trial points in DE. For columns 2–4, TS increases, on average, with the increase of F. From the results in the last row it can be seen that total number of successes for the second and third columns are the same, but it (total of TS) is much less for column four. We now compare the number of function evaluations required for the results in Table 3. Except the Levy function, the FE decreases with the increase of C_R, for all F and for all test functions, in all runs. Since the total FE is dominated by FE for L10, we exclude the FE

Table 4
Comparison of total FE using C_R and F

$F \downarrow$	$C_R = 0.25$	$C_R = 0.5$	$C_R = 0.75$	$C_R = 1$
0.45	68941	25490	19439	20245
0.65	76589	29525	25664	25471
0.85	91144	38751	36783	39440
1	102788	46306	48150	46663
Average	84865	35018	32509	32954

required for Levy. The results of 8 test functions are presented in Table 4. Here, FE is the average number of function evaluation per successful run and the total FE in Table 4 represents total of 8 such FE in set A.

We now compare Table 3 with Table 4. We first compare columns 3 and 4 on both the tables. Comparing average FE in the last row on Table 4, it can be seen that FE for column 3 is about 7% more than that of column 4. However, in term of success column 3 on Table 3 is superior to column 4 (3557 vs 3529). Comparing columns 2 and 3, we see that average FE of column 3 on Table 4 is about 59% superior to that of column 2, given the same number of total successes for these two columns in Table 3. Even if we include FE for L10, still it (average FE for column 3 on Table 4) remains about 43% superior to that of column 2. This motivates us to use $C_R = 0.5$, halfway between the two parents, throughout our numerical investigation. This policy ensures that each parent's components has 50% chance of being selected to produce a new point.

Next we motivate the choice of F. We propose a good strategy for F: the value of F should be large at the early stages and small at the later stages. To this end, we ran DE with $CR = 0.5$ and F being calculated using (12). For this, at the beginning of each generation new $f_{\min}$ and $f_{\max}$ are found prior to the calculation of F. In this implementation of DE on set A the TS was 892, three more than average TS of column 3 of Table 3, and the total result (excluding L10) on FE was 29596 which is about 15% less than average FE of column 3 of Table 4. Similarly, the $cpu = 1.39$ is about 7% less than the corresponding cpu in Table 4. The results motivated us to use F from (12) throughout the rest the paper. A good value of $l_{\min}$ was empirically found to be in $[0.4, 0.5]$. However, in all implementations we have used $l_{\min} = 0.4$.

So far we have obtained the empirical choice of F, C_R and $l_{\min}$ in (12) for the DE algorithm. Next we introduce the pre-calculated differentials (PD) in the DE algorithm to make it faster. We begin our numerical studies by comparing DE with DEPD. DEPD is DE using PD. We compare these two using a number of values for the parameter R. Notice that for $R = 0$ DE and DEPD are identical. We again use the test set A with $N = 7n$. About the same level of TS and the same stopping condition conveniently allowed us to compare the efficiency of the algorithms in terms of cpu and FE. We present the results for DEPD using $R = 1, 2, \ldots, 5$. Hence for a set of results for DE there are five sets for DEPD. We summarize these results in Table 5. We present the percentage of improvement by which DEPD outperforms DE. Improvements are recorded on total results. Total is the total of average cpu and FE for each function. Averages are taken on successful runs over 100 runs. As predictable, Table 5 clearly shows the superiority of DEPD over DE in cpu. However, it is

Table 5
Improvements of DEPD over DE

M	1	2	3	4	5
cpu	13%	14%	17%	26%	10%
FE	0.12%	−0.22%	−1.33%	−1.40%	−6.64%

inferior to DE in FE for some cases. This superiority of DE over DEPD is by a very small margin and except for $R=5$ this superiority can be ignored. Clearly the overall best results are obtained at $R=4$ and for the rest of our numerical studies we use this value for R.

With the introduction of PD we have shown that DE can be made significantly faster. Therefore, together with the other good choice of F, C_R and $l_{\min}$ we use $R=4$ in all of our modified algorithms.

For our first modified algorithm, the DE1 algorithm, the other parameters involved are: M and m_2. The value of M is selected to lie in $[M_1, M_2]$. Initially, M is set to M_2, its highest value, and then gradually decreased to its lowest value M_1. To clarify further, after $M(=M_2)$ generations, m_2 best points from S_a replaces m_2 worst points in S and then M is reduced to, say $M-n$, before the next replacement takes place. The process continues until M reaches M_1. For $M=M_1$, the replacement process continues after every M_1 generations. The effect of m_2 in DE1 was also studied by considering $m_2=n+1$ and $2(n+1)$. A numerical investigation has shown that $M_1=5n$ and $M_2=7n$ work well for DE1.

The modified DE2 and DE3 algorithm also replaces m_2 points of S inasmuch as the same way as in DE1, but they explore the m_2 points using a local search from them just before their (m_2 points) replacement. For instance, in DE3 the local search starts from m_2 best points of S and as soon as the local search is completed they are replaced with the best m_2 points in S_a. Therefore, m_2 is fixed and for DE2/DE3 the value of m_2 is always set to $n+1$. The values of M_1 and M_2 are the same as in DE1. However, if the local search produces more than 5 local minima then these values were doubled, i.e., $M_1=10n$ and $M_2=14n$ to prevent finding multiple local minima. The value of t determines one's confidence in that points in S have converged to a minimum, and we have investigated this by choosing different values for t.

Comparison: We now compare DE and its modified versions, DE1, DE2 and DE3, using total FE, *cpu* and the total number of successes out of 900 runs for set A and 600 runs for set B. The results are summarized in Table 6. It is important to note that DE2 and DE3 were the only algorithms to locate the minimum value for ER10 with success rates about 20% on the average. Since neither DE nor DE1 succeeded in finding the minimum for this problem even once, and the fact that around 40% of the total FE is determined by the FE required for ER10 by DE2 and DE3, a fair comparison is also made excluding the results obtained for ER10, we call this test set B'. The best results were obtained by DE3 on set A and by DE1 on the sets B and B'. Table 6 also shows that comparatively less success occurred for DE2 with $t=3$ on both sets and therefore in our next comparison results using $t=4$ are considered.

For a further comparison we also ran DE($C_R=0.5, F=1$) and DE($C_R=0.5$, F from (12)) on both A and B'. The total number of successes of DE ($C_R=0.5, F=1$) on A is (899, 897) and on B' (394, 407) for the pair of population sizes $(7n, 12n)$. These results for DE($C_R=0.5$, F from (12)) are (892, 900) and (396, 406), respectively, on A and B'. From these results and the results presented in

Table 6
Comparison of DE1 and DE2 using FE, TS and *cpu*

	DE1		DE1		DE2 $t=3$		DE2 $t=4$		DE3 $t=4$	
$m_2 \rightarrow$	$n+1$		$2(n+1)$		$n+1$		$n+1$		$n+1$	
$N \rightarrow$	$7n$	$12n$	$7n$	$12n$	$7n$	$12n$	$7n$	$12n$	$7n$	$12n$
Set A										
FE	53242	97832	51394	94701	38265	62138	42194	71980	38989	62266
cpu	2.43	5.49	2.56	5.33	1.78	3.27	1.84	4.04	1.79	3.57
TS	886	893	887	889	882	879	893	898	897	900
Set B										
FE	113410	224303	105643	196101	219808	421298	233654	421228	292252	499114
cpu	9.56	22.07	9.42	20.65	14.49	30.11	15.30	33.08	17.84	38.08
TS	397	408	397	407	407	422	411	421	421	435
Set B'										
FE	113410	224303	105643	196101	131725	260027	144672	256906	172714	280502
cpu	9.56	22.07	9.42	20.65	11.15	22.43	11.79	25.52	13.08	26.14
TS	397	408	397	407	385	394	389	401	388	403

Table 7
Percentage of improvement made by DE1-3 over DE on FE

	DE1		DE1		DE2 $t=4$		DE3 $t=4$		
$m_2 \rightarrow$	$n+1$		$2(n+1)$		$n+1$		$n+1$		
$N \rightarrow$	$7n$	$12n$	$7n$	$12n$	$7n$	$12n$	$7n$	$12n$	
DE[a]	30	23	42	26	52	44	56	51	Set A
	47	42	50	50	32	33	19	27	Set B'
DE[b]	13	10	17	13	32	34	37	43	Set A
	37	30	41	39	20	20	4	12	Set B'

[a] DE($C_R = 0.5, F = 1$).
[b] DE($C_R = 0.5$, F from (12)).

Table 6 it is clear that TS attained by DE and by all the modified DE methods both on A and B' are very much the same. Therefore, it will be interesting to see how much improvement (say, on FE) has been achieved by the proposed modifications, given the same level of success. This is summarized in terms of percentage of improvement on FE in Table 7. Table 7 shows that a substantial improvement has been achieved by the proposed modifications. In Table 7 the figures in a row for DE($C_R = 0.5$, F from (12)) are less than in the corresponding row for DE($F = 1, C_R = 0.5$), meaning that a good percentage of reduction has been achieved by using (12). However, the majority of the improvements have occurred due to other features of the modified algorithms, i.e., other changes made to the DE. Although DE2 and DE3 are close, DE3 gave slightly better results on the average.

Table 8
Rank ordering based on set A

Rank	1	2	3	4	5	6	7	8	9	10
FE	CRS6	CRSI	GA2	CRS4	CRS2	DE3	DE2	DE1	DE^2	DE^1
cpu	CRSI	CRS6	CRS4	CRS2	DE3	DE2	GA2	DE1	DE^2	DE^1
TS	DE3	DE2	DE^1	DE^2	DE1	CRS4	GA2	CRS2	CRS6	CRSI

Table 9
Rank ordering based on set B

Rank	1	2	3	4	5	6	7	8	9	10
FE	DE1	DE^2	DE^1	CRSI	CRS6	DE2	DE3	GA2	CRS4	CRS2
cpu	DE1	DE^2	DE^1	DE2	DE3	CRSI	CRS6	GA2	CRS4	CRS2
TS	DE3	DE2	DE1	DE^2	DE^1	GA2	CRS2	CRS4	CRS6	CRSI

4.3. Comparison: CRS vs GA vs DE

In this section we compare all population set-based direct search methods to single out a general purpose and robust global optimization method. For this purpose, a rank ordering (from superior to inferior) based on some measure is needed. For the test set A all DE algorithms, except very few cases, were able to locate the global minimum for all problems. Performance of other direct search methods is also better for set A than for B. Therefore, the ranking based on TS, total FE and *cpu* can be made. All results are presented for $N = 12n$, and for the modified DE, $m_2 = n + 1$ is used. The rank ordering is shown in Table 8. Table 8 shows that in terms of FE and *cpu* the CRS and GA are superior but this ranking is reversed in favor of DEs in term of TS. Moreover, in terms of TS the worst performing DE-type method, DE1, is about 5% superior to the best performing CRS-type method, CRS4. On the other hand, in terms of FE the best performing DE-type, DE3, is about 39% worse-off than the worst performing CRS2 within CRS. However, in terms of *cpu* CRS2 is about 18% superior to DE3. For set A, DE and the modified DE are robust in finding the global minima but CRS are faster in *cpu* and use less FE. However, the problem with CRS was that CRSI failed 7 times to converge, twice for S10 and 5 times for S5. By convergence, here, we mean convergence either to a local or to the global minimum. Since in the above table the ranking is almost reversed on efficiency and reliability we try to measure the reliability by keeping the total FE roughly constant. To this end, we ran CRS4 the neighbour of DE1 in the third row of Table 8 with a increased value of $N = 17n$ to achieve roughly the same level of total FE. Although the total success TS of CRS4 is now increased from 856 to 871, but it is still far short of 893 the TS for DE1. If we now run DE1 with a smaller N (e.g., $N = 6n$) for which the FE is more or less the same with CRS4, it still retains superiority over CRS4 in TS. This clearly demonstrates that the DE-type algorithms are always superiors on A. Next we consider the test set B to rank order the algorithms, and the ranking is presented in Table 9. Table 9 shows that the DE-type algorithms are the best in terms of *cpu* and TS. In terms of TS the worst performing DE-type, $DE(C_R = 0.5, F = 1)$, is 55% superior to its close competitor GA2 and 59% superior to CRS2, the best performer of CRS. In

terms of FE the worst DE-type, DE3, is worse-off to both CRS6 and CRSI by about 8% but if we compare DE3, CRSI and CRS6 in terms of TS the superiority of DE3 becomes apparent from the third row. Above all, DE2 and DE3 become the best overall performers if we exclude ER10, the minimum of which was only located by GA2, DE2 and DE3. Therefore, the overall comparisons in Table 9 shows that for the moderate to difficult test problems in set B, DE and the modified DE algorithms always perform better. Moreover, except for GR10, CRSI failed to converge on the rest of the test problems of set B 9 times in all. The CRS2 algorithm also failed to converge 71 times for GR10, 3 times for RG5 and 62 times for SF10, respectively. The other algorithms did not have any such problems of convergence.

5. Discussion and conclusion

The optimization of non-differentiable functions (including noisy or not exactly known functions) is a common problem occurring in various applications. Therefore, using direct search methods which have shown to be robust and efficient could be rewarding. We have developed several versions of the differential evolution (DE) method. From the comparison in the previous section it is quite clear that the new algorithms are superior not only to the original DE but also to many other recent direct search global optimization algorithms. What makes the DE-type algorithms more robust than the CRS and GA-type algorithms is the feature that all points in the set S are possibly updated for each generation. In this way DE has the potential to explore Ω widely with the help of the existing points of the current generation. The CRS-type algorithms lack this features in that they repeatedly replace the $x_{\max}$ whenever a better points is found. This ignores the potential of the points $x_{\max}$ in driving the search and thus making it less exploratory. Unlike DE, where a replacement is done after a point to point comparison, in GA the replacement of m_1 worst points with m_1 better points is mandatory. This repeated replacement in CRS and mandatory replacement in GA makes the re-search procedure of the respective algorithm more intensified rather than diversified. Whereas the DE-type algorithms have the capacity to maintain a balance between this two features: intensification and diversification. The modified DEs have incorporated the features to make the search diversified at early stages and intensified at later stages. Moreover, even if DE2 and DE3 stop at a local minimum using the stopping rule (11) (and not using $t = 4$) it was noticed in many occasions that at least one local search was able to find the global minimum. The incorporation of local search, therefore, enhanced the robustness of the DE-type methods. Further research is underway in developing a hybrid DE global optimization method to find even more robust and efficient DE algorithms.

Appendix A.

Algorithm	Selection of parents	Reproduction	Child Per generation	Acceptance mandatory	Local phase	Added features
CRS1	Random: $n+1$ per trial point	Mutation	1	No	No	None
CRS2	Random: n per trial point	Mutation	1	No	No	None

CRS3	Random: n per trial point	Mutation	1	No	No	None
CRS4	Random: n per trial point	Mutation	1	No	Yes: β-distribution	None
CRS5	Random: n per trial point	Mutation	1	No	Yes: local descend	None
CRS6	Random: 2 per trial point	Interpolation	1	No	Yes: β-distribution	None
CRSI	Random: 2 per trial point	Interpolation	1	No	No	None
DE^1/DE^2	Random: 3 per each target point	Mutation and crossover	N	No	No	None
DE1	Random: 3 per each target point	Mutation and crossover	N	No	No	m_2 of S_a replace m_2 in S
DE2/DE2	Random: 3 per each target point	Mutation and crossover	N	No	DS	m_2 of S_a replace m_2 in S
GA1	Tournament: 2 per pair children	Mutation and crossover	m_1	Yes	No	None
GA2	Random: $n+2$ per pair children	Mutation, crossover and learned crossover	m_1	Yes	No	None

Appendix B.

1. Rastrigin ($RG5$): $f(x) = 10n + \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i)]$; $x_i \in [-5.12, 5.12]$, $n = 5$; $f(x) = 0$; $x^* = (0, 0, \ldots, 0)$.

2. Schwefel ($SF10$): $f(x) = -\sum_{i=1}^{n} x_i \sin(\sqrt{|x_i|})$; $x_i \in [-500, 500]$, $n = 10$; $f(x) = -418.9829 \times n$; $x^* = (s, s, \ldots, s)$, $s = 420.97$.

3. Griewank ($GR10$): $f(x) = 1 + \sum_{i=1}^{n} x_i^2/4000 - \Pi_i^n \cos(\frac{x_i}{\sqrt{i}})$; $x_i \in [-500, 500]$, $n = 10$; $f(x) = 0$; $x^* = (0, 0, \ldots, 0)$.

4. Rosenbrock ($ER10$): $f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i)^2 + (x_i - 1)^2]$; $x_i \in [-500, 500]$, $n = 10$; $f(x) = 0$; $x^* = (1, 1, \ldots, 1)$.

5. Modified Langerman (*ML*5):

$$f(x) = -\sum_{j=1}^{5} c_j \cos(d_j/\pi)\exp(-\pi d_j);\ x_i \in [0,10],\ n = 5;$$
$$d_j = \sum_{i=1}^{n}(x_i - a_{ji})^2,$$
$$f(x^*) = -0.965000;$$
$$x^* = (8.074, 8.777, 3.467, 1.867, 6.708).$$

c_j $j \rightarrow$
0.806,0.517,0.1,0.908,0.965

$j \downarrow$	a_{ji} $i \rightarrow$
	9.681, 0.667, 4.783, 9.095, 3.517
	9.400, 2.041, 3.788, 7.931, 2.882
	8.025, 9.152, 5.114, 7.621, 4.564
	2.196, 0.415, 5.649, 6.979, 9.510
	8.074, 8.777, 3.467, 1.867, 6.708

6. Shekel's Foxholes (*FX*5):

$$f(x) = -\sum_{j=1}^{30} \frac{1}{c_j + \sum_{i=1}^{n}(x_i - a_{ji})}^2;\quad x_i \in [0,10],\quad n = 5;$$
$$f(x^*) = -10.4056;$$
$$x^* = (8.025, 9.152, 5.114, 7.621, 4.564).$$

$j \downarrow c_j$	a_{ji} $i \rightarrow$
0.806	9.681, 0.667, 4.783, 9.095, 3.517
0.517	9.400, 2.041, 3.788, 7.931, 2.882
0.100	8.025, 9.152, 5.114, 7.621, 4.564
0.908	2.196, 0.415, 5.649, 6.979, 9.510
0.965	8.074, 8.777, 3.467, 1.863, 6.708
0.669	7.650, 5.658, 0.720, 2.764, 3.278
0.524	1.256, 3.605, 8.623, 6.905, 4.584
0.902	8.314, 2.261, 4.224, 1.781, 4.124
0.531	0.226, 8.858, 1.420, 0.945, 1.622
0.876	7.305, 2.228, 1.242, 5.928, 9.133
0.462	0.652, 7.027, 0.508, 4.876, 8.807
0.491	2.699, 3.516, 5.874, 4.119, 4.461
0.463	8.327, 3.897, 2.017, 9.570, 9.825
0.714	2.132, 7.006, 7.136, 2.641, 1.882
0.352	4.707, 5.579, 4.080, 0.581, 9.698
0.869	8.304, 7.559, 8.567, 0.322, 7.128
0.813	8.632, 4.409, 4.832, 5.768, 7.050

0.811	4.887, 9.112, 0.170, 8.967, 9.693
0.828	2.440, 6.686, 4.299, 1.007, 7.008
0.964	6.306, 8.583, 6.084, 1.138, 4.350
0.789	0.652, 2.343, 1.370, 0.821, 1.310
0.360	5.558, 1.272, 5.756, 9.857, 2.279
0.369	3.352, 7.549, 9.817, 9.437, 8.687
0.992	8.798, 0.880, 2.370, 0.168, 1.701
0.332	1.460, 8.057, 1.336, 7.217, 7.914
0.817	0.432, 8.645, 8.774, 0.249, 8.081
0.632	0.679, 2.800, 5.523, 3.049, 2.968
0.883	4.263, 1.074, 7.286, 5.599, 8.291
0.608	9.496, 4.830, 3.150, 8.270, 5.079
0.326	4.138, 2.562, 2.532, 9.661, 5.611

References

[1] Törn A, Žilinskas A. Global optimization. Berlin: Springer, 1989.
[2] Nelder JA, Mead R. A simplex method for function minimization. Computer Journal 1965;7:308–13.
[3] Walsh GR. Methods of optimization. London: Wiley, 1975.
[4] Price WL. Global optimization by controlled random search. Computer Journal 1977;20:367–70.
[5] Price WL. A controlled random search procedure for global optimization. In: Dixon LCW, Szegö GP, editors. Towards global optimization 2. Amsterdam, Holland: North-Holland, 1978. p. 71–84.
[6] Ali MM. Some modified stochastic global optimization algorithms with applications. Loughborough University of Technology, Ph.D thesis, 1994.
[7] Ali MM, Storey C, Törn A. Application of some stochastic global optimization algorithms to practical problems. Journal of Optimization Theory and Applications 1997;95(3):545–63.
[8] Goldberg D. Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley Publishing Company, 1989.
[9] 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.
[10] Törn A, Ali MM, Viitanen S. Stochastic global optimization: problem classes and solution techniques. Journal of Global Optimization 1999;14:437–47.
[11] Hu YF, Maguire KC, Cokljat D, Blake RJ. Parallel controlled random search algorithms for shape optimization. In: Emerson DR, Ecer A, Periaux J, Satofuka N, editors. Parallel computational fluid dynamics. Amsterdam, Holland: North-Holland, 1997. p. 345–52.
[12] Hu YF, Maguire KC, Cokljat D, Blake RJ. Parallel controlled random search algorithms. Parallel Computing 2003, in press.
[13] Price WL. Global optimization by controlled random search. Journal of Optimization Theory and Applications 1983;40:333–48.
[14] Price WL. Global optimization algorithms for a CAD workstation. Journal of Optimization Theory and Applications 1987;55:133–46.
[15] Ali MM, Storey C. Modified controlled random search algorithms. International Journal of Computer Mathematics 1995;54:229–35.
[16] Mohan C, Shanker K. A controlled random search technique for global optimization using quadratic approximation. Asia-Pacific Journal of Operational Research 1994;11:93–101.
[17] Ali MM, Törn A, Viitanen S. A numerical comparison of some modified controlled random search algorithms. Journal of Global Optimization 1997;11:377–85.

[18] Grefenstette J. Incorporating problem-specific knowledge in genetic algorithms. In: Davis L, editor. Genetic algorithms and simulated annealing. California, Los Altos: Morgan Kaufmann Publishers, 1987. p. 42–60.
[19] Michalewicz Z. Genetic algorithms + data structures = evolution programs. Berlin: Springer, 1996.
[20] Yen J, Lee B. A simplex genetic algorithm hybrid. Proceedings of the 1997 IEEE International Conference on Evolutionary Computing. Indianapolis, ID, April 13–16, 1997. p. 175–80.
[21] Yang R, Douglas I. Simple genetic algorithm with local tuning: efficient global optimizing technique. Journal of Optimization Theory and Applications 1998;98(2):449–65.
[22] Ali MM, Törn A. Optimization of carbon and silicon cluster geometry for Tersoff potential using differential evolution. In: Floudas CA, Pardalos PM, editors. Optimization in computational chemistry and molecular biology. Dordrecht: Kluwer Academic Publisher, 2000. p. 287–300.
[23] Tezuka S, L'Ecuyer P. Efficient and portable combined Tausworthe random number generators. ACM Transactions on Modeling and Computer Simulation 1991;1:99–112.
[24] Dekkers A, Aarts E. Global optimization and simulated annealing. Mathematical programming 1991;50:367–93.
[25] Ali MM, Storey C. Aspiration based simulated annealing algorithms. Journal of Global Optimization 1997;11: 181–91.
[26] Rinnoy Kan AHG, Timmer GT. Stochastic global optimization methods; part-II: Multilevel methods. Mathematical Programming 1987;39:57–78.