Chapter 7

Differential Evolution

Differential Evolution (DE) is arguably one of the most powerful stochastic real-parameter optimization algorithms in current use. DE operates through the similar computational steps as employed by a standard Evolutionary Algorithm (EA). However, unlike the traditional EAs, the DE-variants perturb the current-generation population members with the scaled differences of randomly selected and distinct population members. Therefore, no separate probability distribution has to be used, which makes the scheme self-organizing in this respect. Since its inception in 1995, DE has drawn the attention of many researchers all over the world resulting in a lot of variants of the basic algorithm with improved performance. This chapter begins with a conceptual outline of classical DE and then presents several significant variants of the algorithm in greater details. It also discusses the theoretical advances made in the field of DE for understanding the search mechanisms and the convergence behavior of the algorithm.

1. Introduction

The Differential Evolution (DE) [1 – 5] algorithm emerged as a very competitive form of evolutionary computing more than a decade ago. The first written article on DE appeared as a technical report by Rainer Storn and Kenneth V. Price in 1995 [1]. One year later, the success of DE was demonstrated at the First International Contest on Evolutionary Optimization in May 1996, which was held in conjunction with the 1996 IEEE International Conference on Evolutionary Computation [2]. DE finished third at the First International Contest on Evolutionary Computation (1st ICEO), which was held in Nagoya, Japan. DE turned out to be the best evolutionary algorithm for solving the real-valued test function suite of the 1st ICEO (the first two places were given to non-evolutionary algorithms, which are not universally applicable but solved the test-problems faster than DE). Price presented DE at the Second International Contest on Evolutionary Optimization in 1997 [3] and it turned out as one of the best among the competing algorithms. Two journal articles [5, 6] describing the algorithm in sufficient details followed immediately in quick sucesion.

In DE community, the individual trial solutions (which constitute a population) are called *parameter vectors* or *genomes*. DE operates through the same computational steps as employed by a standard EA. However, unlike traditional EAs, DE employs difference of the parameter vectors to explore the objective function landscape. In this respect, it owes a lot to its two ancestors namely – the Nelder-Mead algorithm [7], and the Controlled Random Search (CRS) algorithm [8], which also relied heavily on the difference vectors to perturb the current trial solutions. Like other population-based search techniques, DE

generates new points (trial solutions) that are perturbations of existing points, but these deviations are neither reflections like those in the CRS and Nelder-Mead methods, nor samples from a predefined probability density function, like those in Evolutionary Strategies (ES). Instead, DE perturbs current generation vectors with the scaled difference of two randomly selected population vectors. In its simplest form, DE adds the scaled, random vector difference to a third randomly selected population vector to create a *donor* vector corresponding to each population vector (also known as *target* vector). Next the components of the target and donor vectors are mixed through a crossover operation to produce a *trial* vector. In the selection stage, the trial (or offspring) vector competes against the population vector of the same index, i.e. the parent vector. Once the last trial vector has been tested the survivors of all the pair wise competitions become parents for the next generation in the evolutionary cycle.

Since late 1990s, DE started to find several significant applications to the optimization problems arising from diverse domains of science and engineering. DE is a very simple algorithm whose implementation requires only a few lines of code in most of the existing programming languages. Additionally, it takes very few control parameters, which makes it easy to use. Nonetheless, DE exhibits remarkable performance in optimizing a wide variety of multi-dimensional and multi-modal objective functions in terms of final accuracy, convergence speed, and robustness. Perhaps these issues triggered the popularity of DE among researchers all around the globe within a short span of time. A look into the bibliography of DE research [9] maintained by J. Lampinen on the Internet reveals that its maintenance has been halted after 2002. The reason for this stop was that the number of papers began to increase at such a large rate after the year 2002 that it was impossible to keep the bibliography up-to-date and complete.

On the algorithmic front, researchers have frequently attempted to modify the basic form of the algorithm in many ways to adapt DE for tackling multi-objective optimization problems, constrained optimization problems, very high-dimensional (large scale) optimization problems, and optimization problems in dynamic and noisy environments. Several different ways of selection and automatic tuning of the control parameters of DE have been investigated. Some preliminary but important results on the theoretical analysis of the search behavior of DE (for better understanding of the algorithm) have also been reported from statistical and control-theoretic view points. Furthermore, a plethora of research articles have been published on the synergy of DE with other state-of-the-art metaheuristics.

This chapter provides a comprehensive overview of the DE algorithm - basic concepts, control parameters, different structures and variants for solving both real and discrete optimization problems, and theoretical analyses. The rest of the chapter is arranged as follows. In Section 2, the basic concepts related to classical DE are explained along with the original formulation of the algorithm in the real number space. Section 3 presents the complete family of classical DE-variants as proposed by Storn and Price. Section 4 discusses the effects of control parameters like *crossover rate*, *scale factor* and *population size* on the performance of DE. It also focuses on the selection and adaptation scheme for these parameters. Section 5 provides an overview of several recently developed and improved variants of the DE algorithm for continuous single-objective optimization. Section 6 presents the hybrid DE variants in two parts: first it discusses the synergy of DE with other global optimization algorithms and next it outlines how DE can be hybridized with local search techniques. Section 7 describes the modifications of DE for making it

applicable to binary and discrete optimization problems. Advances in the theoretical studies of DE have been reviewed in Section 8. Finally, Section 9 summarizes the chapter.

2. Differential Evolutions: A First Glance

DE is a simple real-coded evolutionary algorithm. It works through a simple cycle of stages, presented in Figure 1. We explain each stage separately in subsections 2.1 to 2.4.

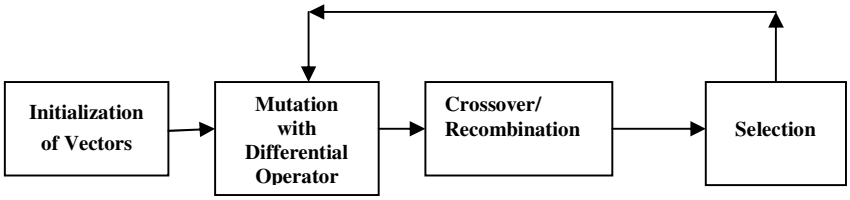


Fig. 1: The main stages of the DE algorithm

2.1 Initialization of the Parameter Vectors

DE searches for a global optimum point in a *D*-dimensional continuous hyperspace. It begins with a randomly initiated population of *NP D* dimensional real-valued parameter vectors. Each vector, also known as *genome/chromosome*, forms a candidate solution to the multi-dimensional optimization problem. We shall denote subsequent generations in DE by $G = 0,1...,G_{\max}$. Since the parameter vectors are likely to be changed over different generations, we may adopt the following notation for representing the *i*-th vector of the population at the current generation:

$$\vec{X}_{i,G} = [x_{1,i,G}, x_{2,i,G}, x_{3,i,G}, ..., x_{D,i,G}]. \quad (1)$$

For each parameter of the problem, there may be a certain range within which the value of the parameter should lie for better search results. The initial population (at $G = 0$) should cover the entire search space as much as possible by uniformly randomizing individuals within the search space constrained by the prescribed minimum and maximum bounds: $\vec{X}_{\min} = \{x_{1,\min}, x_{2,\min}, ..., x_{D,\min}\}$ and $\vec{X}_{\max} = \{x_{1,\max}, x_{2,\max}, ..., x_{D,\max}\}$.

Hence we may initialize the *j*-th component of the *i*-th vector as:

$$x_{j,i,0} = x_{j,\min} + rand_{i,j}[0,1) \cdot (x_{j,\max} - x_{j,\min}), \quad (2)$$

where $rand_{i,j}[0,1)$ is a uniformly distributed random number lying between 0 and 1 (actually $0 \le rand_{i,j}[0,1) < 1$) and is instantiated independently for each component of the *i*-th vector. The process is illustrated in Figure 2 for 10 parameter vectors in two-dimensional search space. Closed curves in Figure 2 denote constant cost contours, which for a given cost function *f*, corresponds to $f(x_1, x_2) = \text{constant}$.

Example 1:

Suppose we are to minimize the five-dimensional sphere function $f(\vec{X}) = \sum_{i=1}^{5} x_i^2$. Suppose we start with a toy population of 5 vectors. If the search range permitted for each variable

lie from -10 to +10 then a particular chromosome say the 2nd one may be initialized as follows:

$x_{1,2,0}$ = -10 + $rand_{2,1}$ (0, 1).{10-(-10)}=-10+0.621×20 = 2.42.

$x_{2,2,0}$ = -10 + $rand_{2,2}$ (0, 1).{10-(-10)}=-10+0.519×20 = 0.38.

$x_{3,2,0}$ = -10 + $rand_{2,3}$ (0, 1).{10-(-10)}=-10+0.982×20 = 9.64

$x_{4,2,0}$ = -10 + $rand_{2,4}$ (0, 1).{10-(-10)}=-10+0.279×20 = -4.42

$x_{5,2,0}$ = -10 + $rand_{2,5}$ (0, 1).{10-(-10)}=-10+0.025×20 = -9.50

Continuing this way, all the 5 vectors are initialized and we start with a population, which looks like:

$\vec{X}_{1,0}$ = [3.82, 4.78, -9.34, 5.36, -3.77]^T

$\vec{X}_{2,0}$ = [2.42, 0.38, 9.64, -4.42, -9.50]^T

$\vec{X}_{3,0}$ = [2.76, 8.41, -0.92, -5.09, -1.32]^T

$\vec{X}_{4,0}$ = [9.12, 7.93, -3.27, -2.08, -9.26]^T

$\vec{X}_{5,0}$ = [5.72, -8.52, -5.04, 3.36, -2.51]^T

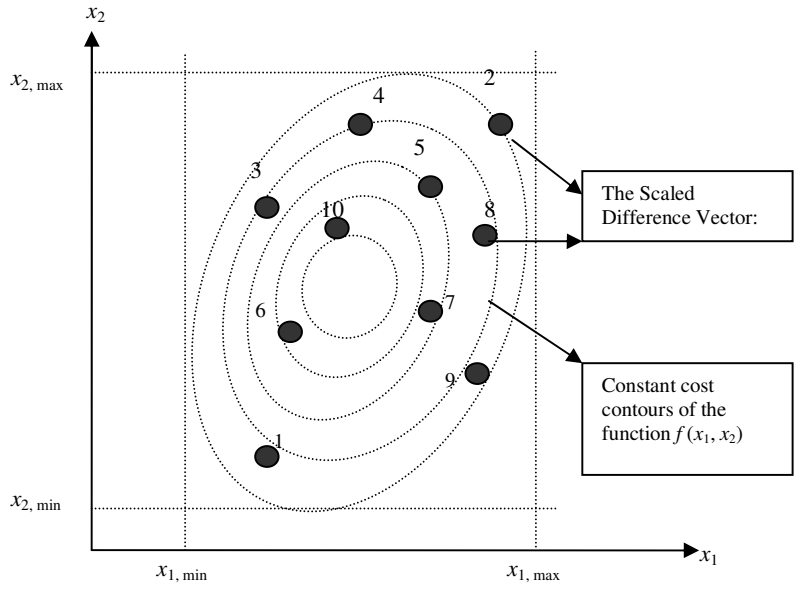


Fig. 2: Initializing a DE population of *NP* = 10, on a two-dimensional parametric space

2.2 Mutation with Difference Vectors

Biologically 'mutation' means a sudden change in the gene characteristics of a chromosome. In the context of the evolutionary computing paradigm, however, mutation is

also seen as a change or perturbation with a random element. Most of the real-coded EAs typically simulate the effects of mutation with additive increments, which are randomly generated by a predetermined Probability Density Function (PDF) [10]. DE, however, applies a uniform PDF not to generate increments, but to randomly sample vector differences like $\Delta\vec{X}_{r_2,r_3} = (\vec{X}_{r_2} - \vec{X}_{r_3})$. In DE, mutation amounts to creating a donor vector $\vec{V}_{i,G}$ for changing each population member $\vec{X}_{i,G}$ (say), in each generation (or in an iteration of the algorithm). In one of the easiest or simplest forms of DE-mutation, to create $\vec{V}_{i,G}$ for each *i*-th member of the current population (also called the *target* vector), three other distinct parameter vectors, say the vectors $\vec{X}_{r_1^i}$, $\vec{X}_{r_2^i}$, and $\vec{X}_{r_3^i}$ are picked up randomly from the current population. The indices r_1^i, r_2^i and r_3^i are mutually exclusive integers randomly chosen from the range [1, *NP*], which are also different from the base vector index *i*. These indices are randomly generated once for each mutant vector. Now the difference of any two of these three vectors is scaled by a scalar number *F* and the scaled difference is added to the third one whence we obtain the donor vector $\vec{V}_{i,G}$. We can express the process as,

$$\vec{V}_{i,G} = \vec{X}_{r_1^i,G} + F \cdot (\vec{X}_{r_2^i,G} - \vec{X}_{r_3^i,G}). \tag{3}$$

The process is illustrated on a two-dimensional parameter space (showing constant cost contours of an arbitrary objective function) in Figure 3.

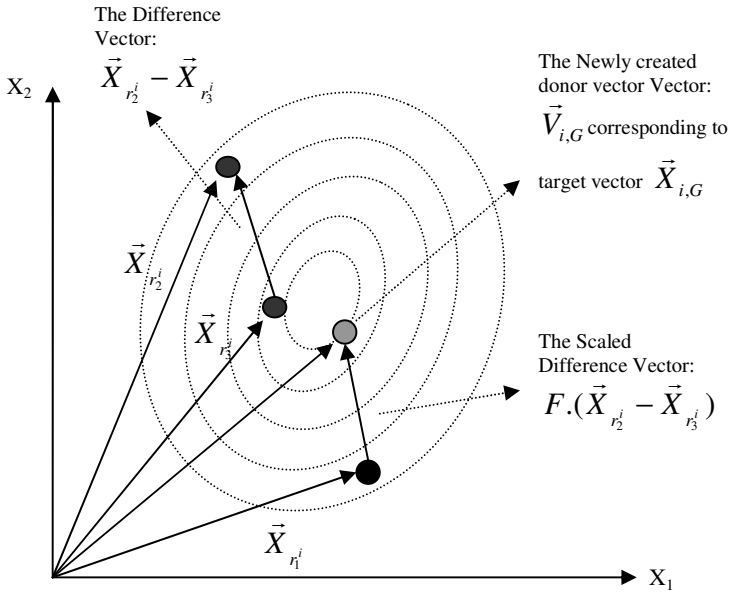


Fig. 3: Illustrating a simple DE mutation scheme in two-dimensional parametric space

Note that in conventional real-coded GAs, mutation takes the form of a random perturbation of a fixed type; for example, a gene value may be augmented by a random fraction of its allowed range. Such a perturbation, whose purpose is to prevent premature convergence, can be needlessly destructive. DE avoids this problem by mutating base vectors (secondary parents) with population-derived difference vectors. As generations pass, these differences tend to adapt to the natural scaling of the problem. For example, if

the population becomes compact in one variable but remains widely dispersed in another, the difference vectors sampled from it will be small in the first variable, yet large in the other. This automatic adaptation significantly improves the convergence of the algorithm.

The following example numerically illustrates the differential mutation scheme.

Example 2: Let us get back to the initialized population of vectors in Example 1. Suppose for the target vector $\vec{X}_{1,0}$ in our program randomly selects three other vectors, say $\vec{X}_{3,0}$, $\vec{X}_{4,0}$ and $\vec{X}_{2,0}$. If the scale factor *F* mentioned in equation (3) assume a value 0.8, then, the donor vector $\vec{V}_{1,0}$ can be formed as:

$$\begin{bmatrix} v_{1,1,0} \\ v_{2,1,0} \\ v_{3,1,0} \\ v_{4,1,0} \\ v_{5,1,0} \end{bmatrix} = \begin{bmatrix} 2.76 \\ 8.41 \\ -0.92 \\ -5.09 \\ -1.32 \end{bmatrix} + 0.8 \left\{ \begin{bmatrix} 9.12 \\ 7.93 \\ -3.27 \\ -2.08 \\ -9.26 \end{bmatrix} - \begin{bmatrix} 2.42 \\ 0.38 \\ 9.64 \\ -4.42 \\ -9.50 \end{bmatrix} \right\} = \begin{bmatrix} 8.12 \\ 14.45 \\ -11.24 \\ -3.22 \\ -1.12 \end{bmatrix}$$

Note that the 2nd and the 3rd elements of the donor vector go out of our previously designated search domain of [-10, 10]. Hence we fix them at 10 and -10 respectively, yielding finally,

$$\vec{V}_{1,0} = \begin{bmatrix} 8.12 \\ 10 \\ -10 \\ -3.22 \\ -1.12 \end{bmatrix}$$

2.3. Crossover

To enhance the potential diversity of the population, a crossover operation comes into play after generating the donor vector through mutation. The donor vector exchanges its components with the target vector $\vec{X}_{i,G}$ under this operation to form the *trial* vector $\vec{U}_{i,G} = [u_{1,i,G}, u_{2,i,G}, u_{3,i,G}, ..., u_{D,i,G}]$. The DE family of algorithms can use two kinds of crossover methods - *exponential* (or two-point modulo) and *binomial* (or uniform) [11]. In exponential crossover, we first choose an integer *n* randomly among the numbers $[1, D]$. This integer acts as a starting point in the target vector, from where the crossover or exchange of components with the donor vector starts. We also choose another integer *L* from the interval $[1, D]$. *L* denotes the number of components; the donor vector actually contributes to the target vector. After choosing *n* and *L* the trial vector is obtained as:

$$u_{j,i,G} = \begin{cases} v_{j,i,G}, & \text{for } j = \langle n \rangle_D, \langle n+1 \rangle_D, ..., \langle n+L-1 \rangle_D \\ x_{j,i,G}, & \text{for all other } j \in [1, D], \end{cases} \quad (4)$$

where the angular brackets $\langle\ \rangle_D$ denote a modulo function with modulus *D*. The integer *L* is drawn from $[1, D]$ according to the following pseudo-code:

```
L = 0;
DO
{
   L = L+1;
} WHILE (( (rand(0,1) < Cr) AND ( L < D )) ;
```

'*Cr*' is called the *crossover rate* and appears as a control parameter of DE just like *F*. Hence in effect, probability $(L \geq \upsilon) = (Cr)^{\upsilon-1}$ for any $\upsilon > 0$. For each donor vector, a new set of *n* and *L* must be chosen randomly as shown above.

Example 3:

This example illustrates the exponential crossover scheme of DE. From the last example, we see that, target vector $\vec{X}_{1,0}$ and the corresponding donor vector $\vec{V}_{1,0}$ are respectively given by,

$$\vec{X}_{1,0} = \begin{bmatrix} 3.82 \\ 4.78 \\ -9.34 \\ 5.36 \\ -3.77 \end{bmatrix} \qquad \vec{V}_{1,0} = \begin{bmatrix} 8.12 \\ 10 \\ -10 \\ -3.22 \\ -1.12 \end{bmatrix}$$

Suppose $n = 2$ and $L=3$ for this specific example. Then the components of $\vec{X}_{1,0}$ that get replaced by the corresponding components of $\vec{V}_{1,0}$ are, from $x_{3,1,0}$ to $x_{5,1,0}$. The newly formed offspring vector finally takes the following form:

$$\vec{U}_{1,0} = \begin{bmatrix} 3.82 \\ 4.78 \\ -10 \\ -3.22 \\ 1.12 \end{bmatrix}$$

Figure 4 provides a visual feel of the process.

Binomial crossover is performed on each of the *D* variables whenever a randomly generated number between 0 and 1 is less than or equal to the *Cr* value. In this case, the number of parameters inherited from the donor has a (nearly) binomial distribution.

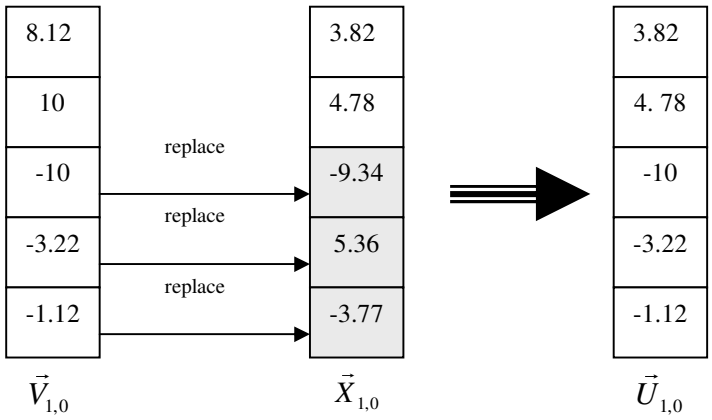


Fig. 4: An Illustration of Exponential Crossover in DE

The scheme may be outlined as:

$$u_{j,i,G} = \begin{cases} v_{j,i,G}, & \text{if } (rand_{i,j}[0,1) \le Cr \text{ or } j = j_{rand}) \\ x_{j,i,G}, & \text{otherwise,} \end{cases} \quad (5)$$

where, as before, $rand_{i,j}[0,1)$ is a uniformly distributed random number, which is instantiated anew for each j-th component of the i-th parameter vector. $j_{rand} \in [1,2,....,D]$ is a randomly chosen index, which ensures that $\vec{U}_{i,G}$ gets at least one component from $\vec{V}_{i,G}$. It is instantiated once for each vector per generation. We note that for this additional demand, Cr is only approximating the true probability p_{Cr} that a component of the trial vector will be inherited from the donor. Also, one may observe that in a two-dimensional search space, three possible trial vectors may result from uniformly crossing a mutant/donor vector $\vec{V}_{i,G}$ with the target vector $\vec{X}_{i,G}$. These trial vectors are:

i) $\vec{U}_{i,G} = \vec{V}_{i,G}$ such that both the components of $\vec{U}_{i,G}$ are inherited from $\vec{V}_{i,G}$.

ii) $\vec{U}'_{i,G}$, in which the first component (j = 1) comes from $\vec{V}_{i,G}$ and the second one (j = 2) from $\vec{X}_{i,G}$.

iii) $\vec{U}''_{i,G}$, in which the first component (j = 1) comes from $\vec{X}_{i,G}$ and the second one (j = 2) from $\vec{V}_{i,G}$.

The possible trial vectors due to uniform crossover are illustrated in Figure 5. Now consider the following example.

Example 4:

This example illustrates the 'binomial' crossover process. Suppose Cr = 0.8. Now again refer to the target vector $\vec{X}_{1,0}$ and the donor vector $\vec{V}_{1,0}$ of Example 2. Suppose for $\vec{X}_{1,0}$,

the random number $j_{rand} = 3$. For each component of $\vec{X}_{1,0}$ one random number $rand_{1,j}$ (0, 1) is generated between 0 and 1. If the value of the said random number is greater than 0.8 (i.e. value of *Cr*), or *j* = 3 then the corresponding component of $\vec{X}_{1,0}$ is replaced by $\vec{V}_{1,0}$ else it is left unaltered. The process is summarized in the following table. Note from Table 1 that although $rand_{1,3}(0, 1) = 0.58 > Cr$, as $j = 3 = j_{rand}$ holds, we have $u_{3,1,0} = v_{3,1,0} = -10$.

Table 1: The binomial crossover process in Example 3

Components of $\vec{X}_1$ at $G = 0$	Components of $\vec{V}_1$ at $G = 0$	$rand_{1,j}(0, 1)$	j_{rand}	Cr	Components of $\vec{U}_{1,0}$
3.82	8.12	0.85	3	0.8	8.12
4.78	10	0.73			10
-9.34	-10	0.89			-10
5.36	-3.22	0.12			-3.22
-3.77	-1.12	0.63			-1.12

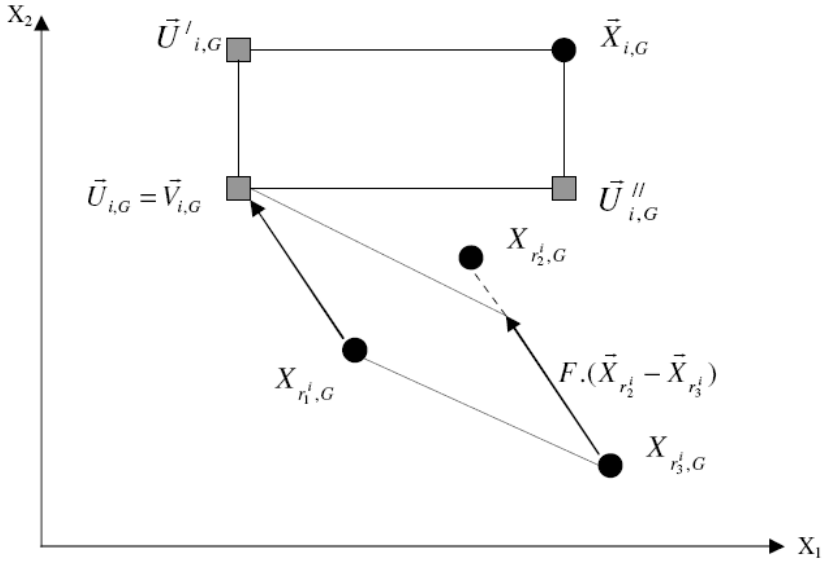


Fig. 5: Different possible trial vectors formed due to uniform/binomial crossover between the target and the mutant vectors in two-dimensional search space

2.4. Selection

To keep the population size constant over subsequent generations, the next step of the algorithm calls for *selection* to determine whether the target or the trial vector survives to the next generation i.e. at $G = G + 1$. The selection operation is described as:

$$\vec{X}_{i,G+1} = \vec{U}_{i,G}, \quad \text{if} \quad f(\vec{U}_{i,G}) \le f(\vec{X}_{i,G})$$

$$= \vec{X}_{i,G}, \quad \text{if } f(\vec{U}_{i,G}) > f(\vec{X}_{i,G}), \tag{6}$$

where $f(\vec{X})$ is the objective function to be minimized. Therefore, if the new trial vector yields an equal or lower value of the objective function, it replaces the corresponding target vector in the next generation; otherwise the target is retained in the population. Hence, the population either gets better (with respect to the minimization of the objective function) or remains the same in fitness status, but never deteriorates. Note that in (6) the target vector is replaced by the trial vector even if both yields the same value of the objective function – a feature that enables DE-vectors to move over flat fitness landscapes with generations. Note that throughout the chapter, we shall use the terms *objective function value* and *fitness* interchangeably. But, always for minimization problems, a lower objective function value will correspond to higher fitness.

Example 5:

This example illustrates the selection mechanism of the classical DE algorithm. The objective function we are trying to minimize is the five dimensional sphere function given by,

$$f(\vec{X}) = \sum_{i=1}^{5} x_i^2$$

Now as found in Example 4, the first population member $\vec{X}_{1,0}$ and its corresponding offspring vector $\vec{U}_{1,0}$ are given by,

$$\vec{X}_{1,0} = \begin{bmatrix} 3.82 \\ 4.78 \\ -9.34 \\ 5.36 \\ -3.77 \end{bmatrix} \qquad \vec{U}_{1,0} = \begin{bmatrix} 3.82 \\ 4.78 \\ -10 \\ -3.22 \\ 1.12 \end{bmatrix}$$

The fitness (or objective function value) of the parent genome $\vec{X}_{1,0}$ is calculated as,

$$f(\vec{X}_{1,0}) = (3.82)^2 + (4.78)^2 + (-9.34)^2 + (5.36)^2 + (-3.77)^2$$
$$= 167.6189$$

Similarly, objective function value for the offspring genome is given by,

$$f(\vec{U}_{1,0}) = (3.82)^2 + (4.78)^2 + (-10)^2 + (-3.22)^2 + (1.12)^2$$
$$= 149.2976$$

clearly we see, $f(\vec{U}_{1,0}) < f(\vec{X}_{1,0})$. Hence at $G = 1$, the target vector $\vec{X}_{1,0}$ is replaced by the trial $\vec{U}_{1,0}$. The vector $\vec{X}_{1,1}$ looks like,

$$\vec{X}_{1,1} = \begin{bmatrix} 3.82 \\ 4.78 \\ -10 \\ -3.22 \\ 1.12 \end{bmatrix}$$

2.5. Summary of DE Iteration

An iteration of the classical DE algorithm consists of the four basic steps – initialization of a population of search variable vectors, mutation, crossover or recombination and finally selection. After having illustrated these stages, we now formally present the whole of the algorithm in a pseudo-code below.

Pseudo-code for the DE algorithm

Step 1. Set the generation number $G = 0$ and randomly initialize a population of NP individuals $P_G = \{\vec{X}_{1,G}, ..., \vec{X}_{NP,G}\}$ with $\vec{X}_{i,G} = [x_{1,i,G}, x_{2,i,G}, x_{3,i,G}, ..., x_{D,i,G}]$ and each individual uniformly distributed in the range $[\vec{X}_{\min}, \vec{X}_{\max}]$, where $\vec{X}_{\min} = \{x_{1,\min}, x_{2,\min}, ..., x_{D,\min}\}$ and $\vec{X}_{\max} = \{x_{1,\max}, x_{2,\max}, ..., x_{D,\max}\}$ with $i = [1,2,...., NP]$.

Step 2. WHILE the stopping criterion is not satisfied
DO
FOR $i = 1$ to NP //do for each individual sequentially

Step 2.1 ***Mutation Step***

Generate a donor vector $\vec{V}_{i,G} = \{v_{1,i,G}, ..., v_{D,i,G}\}$ corresponding to the i-th target vector $\vec{X}_{i,G}$ via the differential mutation scheme of DE (equation (3))

Step 2.2 ***Crossover Step***

Generate a trial vector $\vec{U}_{i,G} = \{u_{1,i,G}, ..., u_{D,i,G}\}$ for the i-th target vector $\vec{X}_{i,G}$ through exponential crossover (equation (4)) or binomial crossover (equation (5)).

Step 2.3 ***Selection Step***

Evaluate the trial vector $\vec{U}_{i,G}$

IF $f(\vec{U}_{i,G}) \leq f(\vec{X}_{i,G})$, THEN $\vec{X}_{i,G+1} = \vec{U}_{i,G}$, $f(\vec{X}_{i,G+1}) = f(\vec{U}_{i,G})$

IF $f(\vec{U}_{i,G}) < f(\vec{X}_{best,G})$, THEN $\vec{X}_{best,G} = \vec{U}_{i,G}$, $f(\vec{X}_{best,G}) = f(\vec{U}_{i,G})$

END IF

ELSE $\vec{X}_{i,G+1} = \vec{X}_{i,G}$, $f(\vec{X}_{i,G+1}) = f(\vec{X}_{i,G})$

END IF

END FOR

Step 2.4 Increase the Generation Count $G = G + 1$

END WHILE

The parameters used in the algorithm namely scale factor '*F*' and crossover rate '*Cr*' should be submitted before invoking the main computational part of the algorithm – the while

loop. The terminating condition can be defined in a few ways like: 1) by a fixed number of iterations G_{max}, with a suitably large value of G_{max} depending upon the complexity of the objective function, 2) when best fitness of the population does not change appreciably over successive iterations, and alternatively 3) attaining a pre-specified objective function value.

Example 6:

This example illustrates the complete search on the fitness landscape of a two dimensional sphere function by a simple DE. Sphere is perhaps one of the simplest two dimensional functions and has been chosen to provide easy visual depiction of the search process. The function is given by,

$$f(\vec{X}) = x_1^2 + x_2^2$$

As can be easily perceived, the function has only one global minima $f^* = 0$ at $\vec{X}^* = [0, 0]^T$. We start with a randomly initialized population of five vectors in the search range [-10, 10]. Initially, these vectors are given by:

$\vec{X}_1 = [5, -9]^T$

$\vec{X}_2 = [6, 1]^T$

$\vec{X}_3 = [-3, 5]^T$

$\vec{X}_4 = [-7, 4]^T$

$\vec{X}_5 = [6, 7]^T$

Figure 6 illustrates the initial orientation of the search variable vectors in the two dimensional $x_1 - x_2$ space. The concentric circular lines are the *constant cost contours* of the function i.e. locus in the $x_1 - x_2$ plane along which $f(\vec{X}) = x_1^2 + x_2^2 =$ constant. Now following the mutation and recombination schemes as presented in previous sections, we form five donor vectors and then create five trial vectors at generation $G = 0$. Next we apply the selection method of DE and evolve the entire population at generation $G = 1$. These steps are summarized in Table 2.

Table 2: Evolution of the population from $G = 0$ to $G = 1$ in example 6

Population at $G = 0$	Objective function value at $G = 0$	Donor vector at $G = 0$	Offspring Vector at $G = 0$	Fitness of offspring at $G = 1$	Evolved population at $G = 1$
$\vec{X}_{1,0} = [2,-1]^T$	5	$\vec{V}_{1,0} = [-0.4,10.4]^T$	$\vec{U}_{1,0} = [-0.4,-1]^T$	1.16	$\vec{X}_{1,1} = [-0.4,-1]^T$
$\vec{X}_{2,0} = [6, 1]^T$	37	$\vec{V}_{2,0} = [1.2, -0.2]^T$	$\vec{U}_{2,0} = [1.2, 1]^T$	2.44	$\vec{X}_{2,1} = [1.2, 1]^T$
$\vec{X}_{3,0} = [-3, 5]^T$	34	$\vec{V}_{3,0} = [-4.4, -0.2]^T$	$\vec{U}_{3,0} = [-4.4, -0.2]^T$	19.4	$\vec{X}_{3,1} = [-4.4, -0.2]^T$
$\vec{X}_{4,0} = [-2, 6]^T$	40	$\vec{V}_{4,0} = [9.2, -4.2]^T$	$\vec{U}_{4,0} = [9.2, 6]^T$	120.64	$\vec{X}_{4,1} = [-2, 6]^T$
$\vec{X}_{5,0} = [6, 7]^T$	85	$\vec{V}_{5,0} = [5.2, 0.2]^T$	$\vec{U}_{5,0} = [6, 0.2]^T$	36.04	$\vec{X}_{5,1} = [6, 0.2]^T$

Figure 7 illustrates how the solutions eventually converge towards the global optima with the progress in generations.

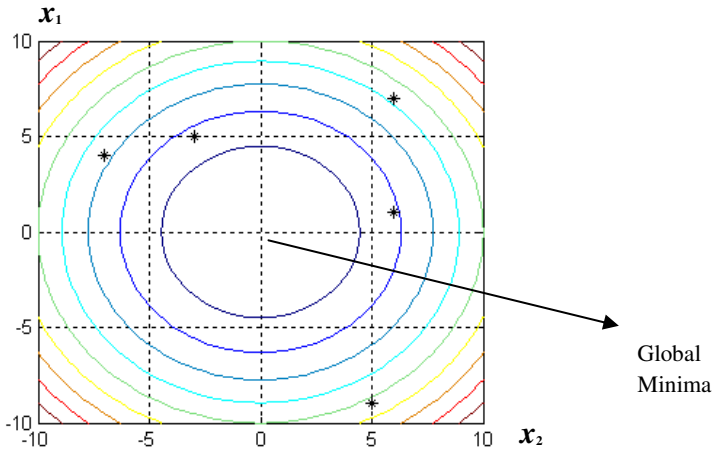
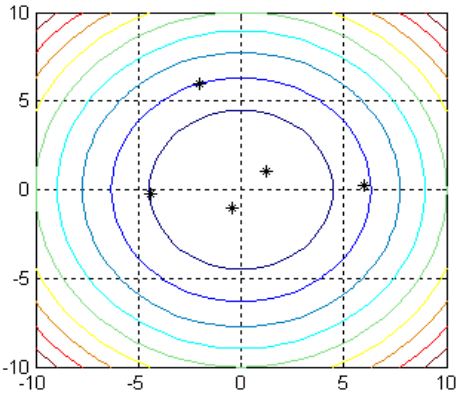
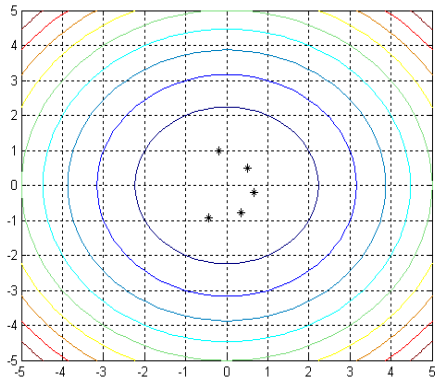


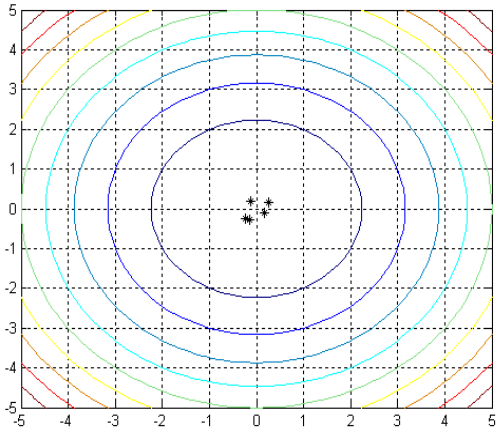
Fig. 6: Orientation of the initial solutions in the two dimensional search space. Each * mark denotes the tip of a parameter vector.



(a) at generation G= 1



(b) at generation G= 4



(c) at generation G= 6

Fig. 7: Orientation of the solutions in two-dimensional search space at different generations

From Figure 7, it is not difficult to observe that all the five vectors tend to converge towards the global minima at t = 6 i.e. only within 6 iterations of the DE algorithm. The best solution vector found at G = 6 is [0.1, -0.05] and it yields a cost function value $f(0.1,-0.05) = (0.1)^2 + (-0.05)^2 = 0.0125$, which is very close to the global minimum at 0.

3. The DE Family of Storn and Price

Actually it is the process of mutation that demarcates one DE scheme from another. In the previous section, we have illustrated the basic steps of a simple DE. The mutation scheme in equation (3) uses a randomly selected vector $\vec{X}_{r1}$ and only one weighted difference vector $F.(\vec{X}_{r2} - \vec{X}_{r3})$ to perturb it. Hence, in literature, the particular mutation scheme given by equation (3) is referred to as DE/rand/1. We can now have an idea of how the different DE schemes are named. The general convention used above is DE/x/y/z, where DE stands for Differential Evolution, x represents a string denoting the base vector to be perturbed, y is the number of difference vectors considered for perturbation of x, and z stands for the type of crossover being used (exp: exponential; bin: binomial).

The other four different mutation schemes, suggested by Storn and Price [32, 33] are summarized below:

"DE/best/1": $$\vec{V}_{i,G} = \vec{X}_{best,G} + F \cdot (\vec{X}_{r_1^i,G} - \vec{X}_{r_2^i,G}). \quad (7)$$

"DE/target-to-best/1": $$\vec{V}_{i,G} = \vec{X}_{i,G} + F \cdot (\vec{X}_{best,G} - \vec{X}_{i,G}) + F \cdot (\vec{X}_{r_1^i,G} - \vec{X}_{r_2^i,G}). \quad (8)$$

"DE/best/2": $$\vec{V}_{i,G} = \vec{X}_{best,G} + F \cdot (\vec{X}_{r_1^i,G} - \vec{X}_{r_2^i,G}) + F \cdot (\vec{X}_{r_3^i,G} - \vec{X}_{r_4^i,G}). \quad (9)$$

"DE/rand/2": $$\vec{V}_{i,G} = \vec{X}_{r_1^i,G} + F \cdot (\vec{X}_{r_2^i,G} - \vec{X}_{r_3^i,G}) + F \cdot (\vec{X}_{r_4^i,G} - \vec{X}_{r_5^i,G}). \quad (10)$$

The indices r_1^i, r_2^i, r_3^i, r_4^i, and r_5^i are mutually exclusive integers randomly chosen from the range [1, *NP*], and all are different from the base index *i*. These indices are randomly generated once for each donor vector. The scaling factor *F* is a positive control parameter for scaling the difference vectors. $\vec{X}_{best,G}$ is the best individual vector with the best fitness (i.e. lowest objective function value for a minimization problem) in the population at generation *G*. Note that some of the strategies for creating the donor vector may be mutated recombinants, for example, equation (8) listed above basically mutates a two-vector recombinant: $\vec{X}_{i,G} + F \cdot (\vec{X}_{best,G} - \vec{X}_{i,G})$. This variant is also known as DE/current-to-best/1

Storn and Price [12] suggested total ten different working strategies of DE and some guidelines in applying these strategies to any given problem. These strategies were derived from the five different DE mutation schemes outlined above. Each mutation strategy was combined with either the 'exponential' type crossover or the 'binomial' type crossover. This yielded a total of 5× 2 = 10 DE strategies, which are listed below.

1. DE/best/1/exp
2. DE/rand/1/exp

3. DE/target-to-best/1/exp
4. DE/best/2/exp
5. DE/rand/2/exp
6. DE/best/1/bin
7. DE/rand/1/bin
8. DE/target-to-best/1/bin
9. DE/best/2/bin
10. DE/rand/2/bin

In fact many other linear vector combinations can be used for mutation. A generalized DE mutation scheme may be put forward as:

$$\vec{V}_{i,G} = \vec{Y}_{i,G} + F.\frac{1}{N}\sum_{n=0}^{N-1}(\vec{X}_{r_{(2n+1),},G} - \vec{X}_{r_{(2n+2),}G}), \tag{11}$$

where $\vec{Y}_{i,G}$ is the base vector, which must be distinct from the other vectors used in equation (11). Recently Price and Rönkkönen [13] investigated equation (11) for the case $\vec{Y}_{i,G} = \vec{X}_{i,G}$, there is no division by N, and $N = 0$. So far no single perturbation method has turned out to be best for all problems and this fact, of course, is not surprising in context to the No Free Lunch (NFL) theorem [14]. Nevertheless all the various methods need further investigation under which circumstances they perform well. Some initial work in this direction was undertaken by Mezura-Montes *et al.* who empirically compared 8 different DE-schemes over a test-suite of 13 benchmark problems in [15]. Their results indicated that the most competitive approach, regardless the characteristics of the problem to be solved was the *"DE/best/*1*/bin"* (using always the best solution to find search directions and also binomial crossover), based on final accuracy and robustness of results.

4. Control Parameters of DE

There are three main control parameters of the DE algorithm: the mutation scale factor *F*, the crossover constant *Cr*, and the population size *NP*. In this section, we focus on the effect of each of these parameters on the performance of DE as well as the state-of-the-art methods for tuning these parameters. A good volume of research work has been undertaken so far to improve the ultimate performance of DE by tuning its control parameters. Storn and Price in [1] have indicated that a reasonable value for *NP* could be chosen between 5*D* and 10*D* (*D* being the dimensionality of the problem), and a good initial choice of *F* was 0.5. The effective value range of *F* is usually between 0.4 and 1.

Gamperle *et al.* [16] evaluated different parameter settings for DE on the Sphere, Rosenbrock's, and Rastrigin's functions. Their experimental results revealed that the global optimum searching capability and the convergence speed are very sensitive to the choice of control parameters *NP, F* and *Cr.* Furthermore, a plausible choice of the population size *NP* is between 3*D* and 8*D*, the scaling factor $F = 0.6$, and the crossover rate *Cr* is between [0.3, 0.9]. Recently, the authors in [17] state that typically $0.4 < F < 0.95$ with $F = 0.9$ can serve as a good first choice. They also opine that *Cr* should lie in (0, 0.2) when the function is separable, while in (0.9, 1) when the function's parameters are dependent.

As can be perceived from the literature, several claims and counter-claims were reported concerning the rules for choosing the control parameters, confusing the engineers who try to solve practical problems with the DE. Further, most of these claims lack sufficient experimental justifications. Some objective functions are very sensitive to the proper choice

of the parameter settings in DE [18]. Therefore, researchers naturally started to consider some techniques such as self-adaptation to automatically find an optimal set of control parameters for DE [19 – 24]. Usually self-adaptation is applied to tune the control parameters *F* and *Cr*. Liu and Lampinen [20] introduced a Fuzzy Adaptive Differential Evolution (FADE) using fuzzy logic controllers whose inputs incorporate the relative function values and individuals of successive generations to adapt the parameters for the mutation and crossover operation. In this context, Qin *et al*. [22] came up with a Self-adaptive DE (SaDE) algorithm, in which both the trial vector generation strategies and their associated control parameters *F* and *Cr* are gradually self-adapted by learning from their previous experiences of generating promising solutions. The scheme will be discussed in details in Section 5.6.

In [23] a fitness-based adaptation has been proposed for *F*. A system with two evolving populations has been implemented. The crossover rate *Cr* has been set equal to 0.5 after an empirical study. Unlike *Cr*, the value of *F* is adaptively updated at each generation by means of the following scheme:

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

where $l_{\min} = 0.4$ is the lower bound of *F*, $f_{\min}$ and $f_{\max}$ are the minimum and maximum objective function values over the individuals of the populations, obtained in a generation.

Recently Brest *et al.* proposed a self adaptation scheme for the DE control parameters [24]. They encoded control parameters *F* and *Cr* into the individual and adjusted introducing two new parameters τ_1 and τ_2. In their algorithm (called 'jDE'), a set of *F* and *Cr* values was assigned to each individual in the population, augmenting the dimensions of each vector. The better values of these encoded control parameters lead to better individuals that in turn, are more likely to survive and produce offspring and, thus, propagate these better parameter values. The new control parameters for the next generation are computed as follows:

$$\left.\begin{aligned} F_{i,G+1} &= F_l + rand_1 * F_u && \text{if } rand_2 < \tau_1 \\ &= F_{i,G} && \text{else,} \end{aligned}\right\} \tag{13.a}$$

and

$$\left.\begin{aligned} Cr_{i,G+1} &= rand_3 && \text{if } rand_4 < \tau_2 \\ &= Cr_{i,G} && \text{else,} \end{aligned}\right\} \tag{13.b}$$

where $rand_j$, $j \in \{1,2,3,4\}$ are uniform random values in the interval [0, 1]. τ_1 and τ_2 may be interpreted as the probabilities of adjusting the factors *F* and *Cr*. In [24] Brest *et al.* used $\tau_1 = \tau_2 = 0.1$. As $F_l = 0.1$ and $F_u = 0.9$, the new *F* takes a value from [0.1, 0.9] while the new *Cr* takes a value from [0, 1]. As the $F_{i,G+1}$ and $CR_{i,G+1}$ values are obtained before the mutation is performed, they influence the mutation, crossover, and selection operations of the new vector $\vec{X}_{i,G+1}$. However, it remains unclear whether the scheme would maintain it superiority if not a fixed number of evaluations but a fixed value-to-reach (VTR) would have been chosen as a goal.

Zaharie [25] proposed a parameter adaptation strategy for DE (ADE) based on the idea of controlling the population diversity, and implemented a multi-population approach. Following the same line of thinking, Zaharie and Petcu [26] designed an adaptive Pareto DE algorithm for multi-objective optimization and also analyzed its parallel implementation. Abbass [27] self-adapted the crossover rate *Cr* for multi-objective optimization problems, by encoding the value of *Cr* into each individual, simultaneously evolved with other search variables. The scaling factor *F* was generated for each variable from a Gaussian distribution $N(0,1)$. The upper limit of the scale factor *F* is empirically taken as 1. Although it does not necessarily mean that a solution is not possible with $F > 1$, however, until date, no benchmark function that was successfully optimized with DE required $F > 1$. Zaharie [28] derived a lower limit of *F* and her study revealed that if *F* be sufficiently small, the population can converge even in the absence of selection pressure. With a few simplifying assumptions, she proved the following relation between the expected variance of the original population $P_{x,t}$ at time step $G = G$ and the variance of the trial population $P_{u,G}$:

$$E(Var(P_{u,G})) = \left(2.F^2.p_{Cr} - \frac{2.p_{Cr}}{NP} + \frac{p_{Cr}^2}{NP} + 1 \right) .Var(P_{x,G}), \tag{14}$$

where p_{Cr} is the probability of crossover (Zaharie neglected the j_{rand} part in equation (5) and therefore p_{Cr} is the absolute probability that a component of the target vector is exchanged with that of the donor vector). Consequently, the DE control parameter combinations that satisfy the equation:

$$2.F^2 - \frac{2}{NP} + \frac{p_{Cr}}{NP} = 0, \tag{15}$$

may be considered as critical since they result in a population whose variance remains constant except for random fluctuations. Thus, when the selection step is absent, according to equation (13), *F* will display a critical value F_{crit} such that the population variance decreases when $F < F_{crit}$ and increases if $F > F_{crit}$. Solving equation (13) we have,

$$F_{crit} = \sqrt{\frac{\left(1 - p_{Cr}/2\right)}{NP}} \tag{16}$$

Zaharie experimentally confirmed that F_{crit} establishes a lower limit on the value of *F* in the sense that smaller values will induce convergence even on a flat objective function landscape (when all trial vectors are accepted, i.e. selection pressure is absent). Omran *et al*. [29] introduced a self-adaptive scaling factor parameter *F*. They generated the value of *Cr* for each individual from a normal distribution $N(0.5,0.15)$. This approach (called 'SDE') was tested on four benchmark functions and performed better than other versions of DE. Besides adapting the control parameters *F* or *Cr*, some researcher also adapted the population size. Teo [30] proposed DE with self adaptive population size *NP* (abbreviated as DESAP), based on self-adaptive Pareto DE proposed by Abbas [27].

Mallipeddi and Suganthan [31] empirically investigated the effect of population size on the quality of solutions and the computational effort required by the Differential evolution (DE) Algorithm with a set of 5 problems chosen from the problem set of CEC 2005 Special Session on Real-Parameter Optimization [32]. In [33] the authors present a method for gradually reducing population size of DE. The method improves the efficiency and robustness of the algorithm and can be applied to any variant of a DE algorithm. In [34]

Mallipeddi and Suganthan propose a DE algorithm with an ensemble of parallel populations, where the number of function evaluations allocated to each population is self-adapted by learning from their previous experiences in generating superior solutions. Consequently, a more suitable population size along with its parameter settings can be determined adaptively to match different search / evolution phases.

Apart from self-adaptation, frequently F has been made to vary randomly for improving the performances of DE. Price *et al*. [11] defined two new terms: *jitter* and *dither* in context to the randomization of F. The practice of generating a new value of F for every *parameter* is called *jitter* and it is signified by subscripting F with the parameter index, j. Alternatively, choosing F anew for each *vector*, or *dithering*, is indicated by subscripting F with the population's running index, i. Dithering scales the length of vector differentials because the same factor, F_i, is applied to all components of a difference vector. Das *et al*. used dither in [35] where F was made to vary randomly between 0.5 and 1 for each vector. In the same paper, they also suggested decreasing F linearly from 1.0 to 0.5 in their second scheme (called DETVSF: DE with Time varying Scale Factor). This encourages the individuals to sample diverse zones of the search space during the early stages of the search (promoting exploration). During the later stages a decaying scale factor helps to adjust the movements of trial solutions finely so that they can explore the interior of a relatively small space in which the suspected global optimum lies (thus promoting exploitation).

Recently in works like [36, 37], chaotic sequences are combined with DE in order to enhance its population diversity and thus to avoid stagnation. Chaos theory [38] deals with the qualitative study of unstable aperiodic behavior in deterministic nonlinear dynamical systems. In chaotic DE the scale factor F is varied over generations by using the logistic map iterator, which is one of the simplest dynamic systems evidencing chaotic behavior, in the following way:

$$F_G = \mu . F_{G-1} . [1 - F_{G-1}] . \tag{17}$$

5. Important Variants of DE for Continuous Single-objective Optimization

Since its advent in 1995, DE has been attracting the attention of the researchers from diverse domains of knowledge, all over the world. This has resulted into the wealth of variants of the basic DE algorithm. Some of these variants are application specific while others are generalized for global numerical optimization. Without forgetting the NFL [14], one may note that each of these modified DE algorithms has its own advantages and disadvantages and none is suitable for tackling all kinds of optimization problems that appear in real world.

5.1. Differential Evolution Using Trigonometric Mutation

Fan and Lampinen [39] proposed a trigonometric mutation operator for DE to speed up its performance. To implement the scheme, for each target vector, three distinct vectors are randomly selected from the DE population. Suppose for the i-th target vector $\vec{X}_{i,G}$, the selected population members are $\vec{X}_{r_1,G}$, $\vec{X}_{r_2,G}$ and $\vec{X}_{r_3,G}$. The indices r_1, r_2 and r_3 are mutually exclusive integers randomly chosen from the range [1, *NP*], which are also

different from the index *i*. Now three weighing coefficients are formed according to the following equations:

$$p' = \left|f(\vec{X}_{r_1})\right| + \left|f(\vec{X}_{r_2})\right| + \left|f(\vec{X}_{r_3})\right|, \tag{18a}$$

$$p_1 = \left|f(\vec{X}_{r_1})\right| \Big/ p', \tag{18b}$$

$$p_2 = \left|f(\vec{X}_{r_2})\right| \Big/ p', \tag{18c}$$

$$p_3 = \left|f(\vec{X}_{r_3})\right| \Big/ p', \tag{18d}$$

where $f()$ is the function to be minimized. Let Γ be the trigonometric mutation rate in the interval (0, 1).Then the trigonometric mutation scheme may now be expressed as:

$$\vec{V}_{i,G+1} = (\vec{X}_{r1} + \vec{X}_{r2} + \vec{X}_{r3})/3 + (p_2 - p_1).(\vec{X}_{r1} - \vec{X}_{r2}) + (p_3 - p_2).(\vec{X}_{r2} - \vec{X}_{r3}) + (p_1 - p_3).(\vec{X}_{r3} - \vec{X}_{r1}) \quad \text{if } rand(0,1) < \Gamma$$

$$\vec{V}_{i,G+1} = \vec{X}_{r1} + F.(\vec{X}_{r2} - \vec{X}_{r3}) \quad \text{else.} \tag{19}$$

Thus the scheme proposed by Fan *et al*. used trigonometric mutation with a probability of Γ and the mutation scheme of DE/rand/1 with a probability of $(1-\Gamma)$.

5.2. Differential Evolution Using Arithmetic Recombination

The binomial crossover scheme, usually employed in most of the DE variants, creates new combinations of parameters; it leaves the parameter values themselves unchanged. Binomial crossover is in spirit same as the discrete recombination used in conjunction with many EAs. However, in *continuous* or *arithmetic* recombination, the individual components of the trial vector are expressed as a linear combination of the components from mutant/donor vector and the target vector. The common form of the arithmetic recombination between two vectors $\vec{X}_{r_1,G}$ and $\vec{X}_{r_2,G}$ adopted by most of the EAs [10] may be put as:

$$\vec{W}_{i,G} = \vec{X}_{r_1,G} + k_i.(\vec{X}_{r_1,G} - \vec{X}_{r_2,G}) \tag{20}$$

The *coefficient of combination* k_i can either be a constant or a random variable (e.g. $rand(0,1)$). Generally speaking, if this coefficient is sampled anew for each vector then the resulting process is known as line recombination. However, if the combination coefficient is elected randomly anew for each component of the vectors to be crossed, then the process is known as intermediate recombination and may be formalized for the *j*-th component of the recombinants as:

$$w_{i,j,G} = x_{r_1,j,G} + k_j.(x_{r_1,j,G} - x_{r_2,j,G}) \tag{21}$$

Figure 8 schematically shows the regions searched by discrete, line and arithmetic recombination between donor vector $\vec{V}_{i,G}$ and the target vector $\vec{X}_{i,G}$ when the coefficient of combination is a uniformly distributed random number between 0 and 1. The two recombinant vectors occupy the opposite corners of a hypercube whose remaining corners are the trial vectors $\vec{U}'_{i,G}$ and $\vec{U}''_{i,G}$ created by discrete recombination. Line recombination, as its name suggests, searches along the axis connecting the recombinant vectors, while the

intermediate recombination explores the entire *D*-dimensional volume contained within the hypercube. As can be perceived from Figure 9, both the discrete as well as the intermediate recombination are not rotationally invariant processes. If the coordinate system rotates through an angle, the corners of the hypercube are relocated, which in turn redefines the area searched by the intermediate recombination. On the other hand, the line recombination is rotationally invariant.

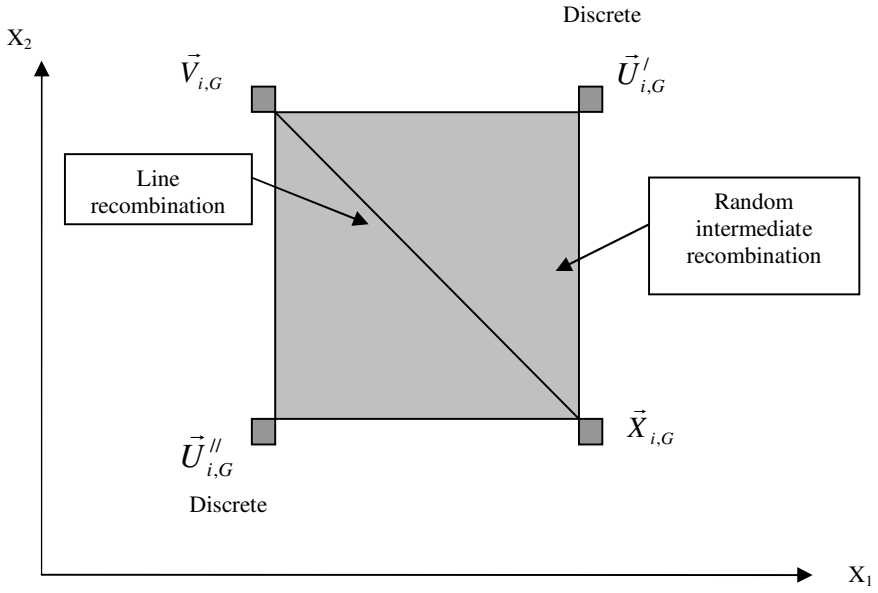


Fig. 8: Domains of the different recombinant vectors generated using discrete, line and random intermediate recombination.

Since a good global optimizer needs to be rotationally invariant, to overcome the limitation of the discrete recombination usually employed in DE, a new trial vector generation strategy 'DE/current-to-rand/1' is proposed in [12], which replaces the binomial crossover operator with the rotationally invariant arithmetic line recombination operator to generate the trial vector $\vec{U}_{i,G}$ by linearly combining the target vector $\vec{X}_{i,G}$ and the corresponding donor vector $\vec{V}_{i,G}$ as follows:

$$\vec{U}_{i,G} = \vec{X}_{i,G} + k_i.(\vec{V}_{i,G} - \vec{X}_{i,G}) \tag{22}$$

Now incorporating equation (3) in (22) we have:

$$\vec{U}_{i,G} = \vec{X}_{i,G} + k_i.(\vec{X}_{r_1,G} + F.(\vec{X}_{r_2,G} - \vec{X}_{r_3,G}) - \vec{X}_{i,G}), \tag{23}$$

which further simplifies to:

$$\vec{U}_{i,G} = \vec{X}_{i,G} + k_i.(\vec{X}_{r_1,G} - \vec{X}_{i,G}) + F'.(\vec{X}_{r_2,G} - \vec{X}_{r_3,G}), \tag{24}$$

where k_i is the combination coefficient, which has been experimentally shown [32, 33] to be effective when it is chosen with a uniform random distribution from [0, 1] and $F' = k_i \cdot F$ is a new constant here.

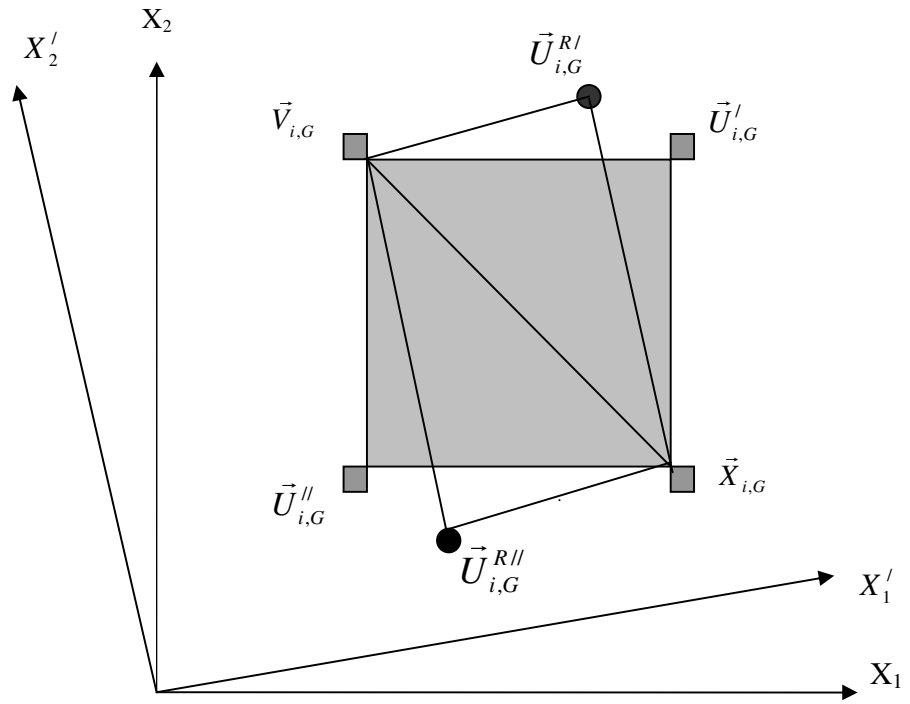


Fig. 9: Change of the trial vectors generated through the discrete and random intermediate recombination due to rotation of the coordinate system. $\vec{U}_{i,G}^{R/}$ and $\vec{U}_{i,G}^{R//}$ indicate the new trial vectors due to discrete recombination in rotated coordinate system.

5.3. The DE/rand/1/Either-Or Algorithm

DE/rand/1/either-or is a state-of-the-art DE variant described by Price *et al*. [11, page 118]. In this algorithm, the trial vectors that are pure mutants occur with a probability p_F and those that are pure recombinants occur with a probability $1 - p_F$. The scheme for trial vector generation may be outlined as:

$$\left.\begin{aligned} \vec{U}_{i,G} &= \vec{X}_{r_1,G} + F.(\vec{X}_{r_2,G} - \vec{X}_{r_3,G}) \quad \text{if } rand_i(0,1) < p_F \\ &= \vec{X}_{r_0,G} + k.(\vec{X}_{r_1} + \vec{X}_{r_2} - 2.\vec{X}_{r_0}) \quad \text{otherwise} \end{aligned}\right\} \qquad (25)$$

Price *et al.* recommended $k = 0.5.(F + 1)$ as a good choice for the parameter k for a given *F*. The DE/rand/1/either-or algorithm provides a simple way to implement the dual axis search in the *k*-*F* plane (*k* indicating the combination coefficient of the arithmetic crossover and *F* being the scale factor). The scheme provides efficient solutions for functions that are best minimized by either mutation-only ($p_F = 1$) or recombination only

($p_F = 0$), as well as generic functions that can be solved by randomly interleaving both operations ($0 < p_F < 1$).

5.4. The Opposition-based Differential Evolution

The concept of *opposition-based learning* was introduced by Tizhoosh [40] and its applications were introduced in [41 - 43]. Rahnamayan *et al.* [43] have recently proposed an *Opposition-based DE* (ODE) for faster global search and optimization. The algorithm also finds important applications to the noisy optimization problems [44]. The conventional DE was enhanced by utilizing *opposition number* based optimization concept in three levels, namely, population initialization, generation jumping, and local improvement of the population's best member. *Opposite numbers* may be defined in the following way:

***Definition 1*:** Let *x* be a real number defined in the closed interval $[a,b]$, i. e. $x \in [a,b]$. Then the opposite number $\overset{\cup}{x}$ of *x* may be defined as:

$$\overset{\cup}{x} = a + b - x \tag{26}$$

The ODE changes the classical DE using the concept of opposite numbers at the following three different stages:

1) Opposition based Population Initialization

In most of the existing versions of the DE, random number generation is the only choice to create initial population. But as demonstrated by Tizhoosh [40, 41], concept of opposition-based optimization can help in obtaining fitter starting points in the search space even when there is no a priori knowledge about solutions. Rahnamayan *et al.*[43] proposed the following scheme:

(a) Generating uniformly distributed random population $P(NP)$, *NP* is the population size

(b) Calculate the opposite population $OP(NP)$. The *k*-th opposite individual corresponding to the *k*-th parameter vector of $P(NP)$ may be given as (following equation (26),

$$OP_{k,j} = a_{k,j} + b_{k,j} - P_{k,j}, \quad \text{where } k = 1,2,....,NP \text{ and } j = 1,2,....,D$$

$a_{k,j}$ and $b_{k,j}$ denote the interval boundaries of *j*-th parameter of the *k*-th vector i. e. $x_{k,j} \in [a_{k,j}, b_{k,j}]$.

(c) Select *NP* fittest individuals from set the $\{P(NP), OP(NP)\}$ as initial population.

2) Opposition Based Generation Jumping

Generation jumping means after each iteration, depending upon a predetermined probability *Jr*, instead of generating new population by evolutionary process, the opposite population may be calculated (if $rand(0,1) < Jr$) and the *NP* fittest individuals are selected from the current population and the corresponding opposite population (exactly similar to what was performed for opposition-based population initialization) depending on some. The authors suggest taking a small value of *Jr* such that approximately $Jr \in (0,0.04)$.

3) Opposition-based Best Individual Jumping

This stage improves the best candidate solution (the fittest member) in the current population by applying small perturbations and through the following steps:

(a) Create difference-offspring of the best individual in the current population by:

$$new\_best = best + F'.(\vec{X}_{r1} - \vec{X}_{r2}) \quad (27)$$

where $r1$ and $r2$ are mutually different random integer indices selected from {1, 2, ..., n}. F' is a real constant which determines amplification of the added differential vector. F' should be set to a small number $F' \in [0,0.02]$ because we need a small/local exploration to improve the current best member. In contrast, a large value for F' can reduce the chance to obtain a better candidate.

(b) Calculate opposite of offspring created in (a) and call it *op_newbest*,

(c) Replace the current best member by the fittest member of the set {*best, newbest, op_newbest*}.

It should be mentioned that to calculate the opposite individuals for generation jumping and also for the best individual jumping (step (3)), the opposite of each variable is calculated dynamically. It means, the maximum and minimum values of each variable in *current population* ($[a_j^p, b_j^p]$) are used to calculate opposite point instead of using variables' predefined interval boundaries i.e.

$$OP_{k,j} = a_{k,j}^p + b_{k,j}^p - P_{k,j}, \quad (28)$$

where $k = 1,2,....,NP$ and $j = 1,2,....,D$. This dynamic behavior of the opposite point calculation increases the chance to find fitter opposite points. A flow chart representation of the ODE algorithm has been provided in Figure 10. The experiments conducted in [43] confirmed that the proposed ODE algorithm performs better than the DE in terms of convergence speed over noisy and noise-free functions.

5.6 DE with Adaptive Selection of Mutation Strategies

DE can encompass a number of trial vector generation strategies, each of which may be effective over certain problems but poorly perform over the others [15]. In [22], Qin *et al.* for the first time proposed a new self-adaptive variant of DE (SaDE), in which both trial vector generation strategies and their associated control parameter values are gradually self-adapted by learning from their previous experiences in generating promising solutions. Consequently, it is possible to determine a more suitable generation strategy along with its parameter settings can be determined adaptively to match different phases of the search process / evolution.

In SaDE, four effective trial vector generation strategies namely the DE/rand/1/bin, DE/rand-to-best/2/bin, DE/rand/2/bin and finally DE/current-to-rand/1 were chosen to constitute a strategy candidate pool. The first three DE-variants are equipped with binomial type crossover (a most popular kind of recombination in DE-community [11]), which is described in equation (5).

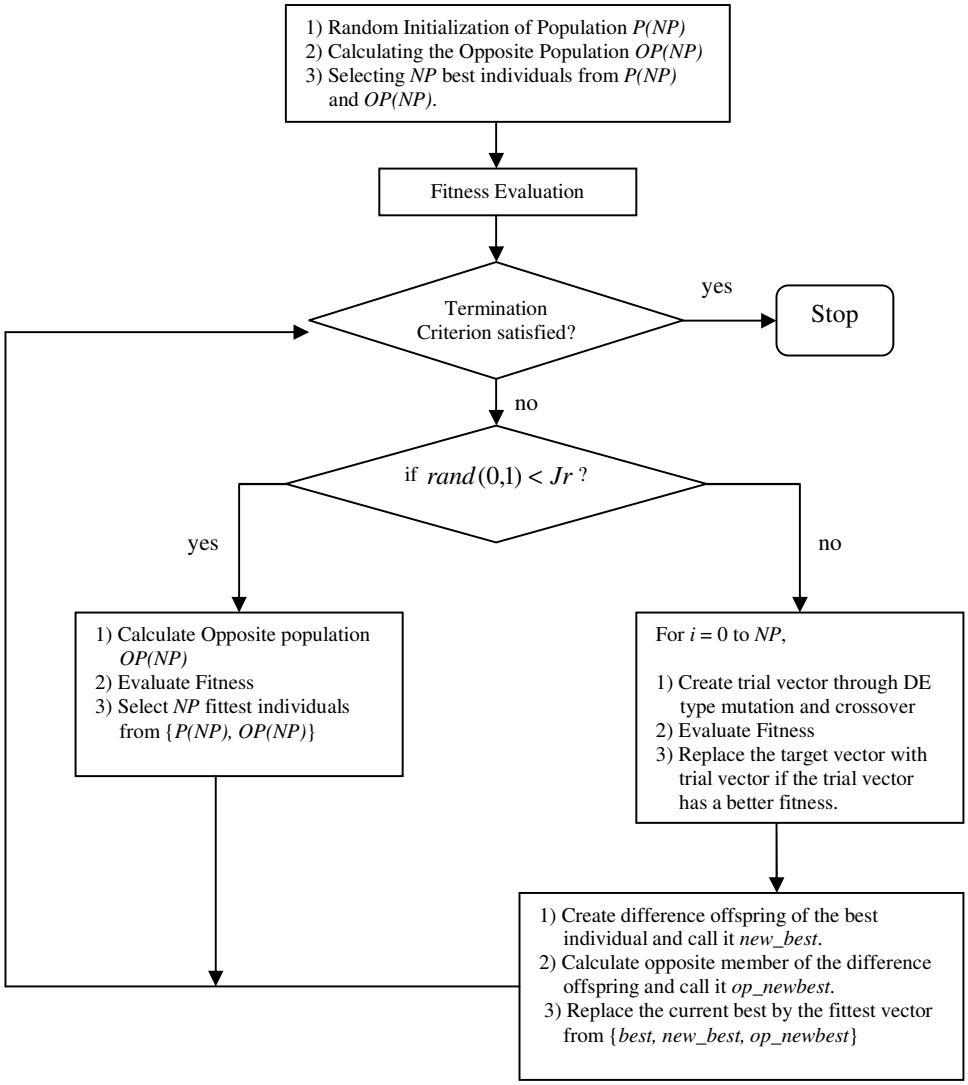


Fig. 10: Flow-chart Representation of the ODE Algorithm

The DE/rand-to-best/2/bin scheme may be illustrated for the j-th component of the i-th target vector as:

$$u_{i,j} = \begin{cases} x_{i,j} + F.(x_{best,j} - x_{i,j}) + F.(x_{r1,j} - x_{r2,j}) + F.(x_{r3,j} - x_{r4,j}) & \text{if } rand[0,1) < CR \text{ or } j = j_{rand} \\ x_{j,i} & \text{otherwise,} \end{cases} \tag{29}$$

where the symbols bear their usual meanings.

In the SaDE algorithm, for each target vector in the current population, one trial vector generation strategy is selected from the candidate pool according to the probability learned from its success rate in generating improved solutions (that can survive to the next

generation) within a certain number of previous generations, called the *Learning Period* (*LP*). The selected strategy is subsequently applied to the corresponding target vector to generate a trial vector. More specifically, at each generation, the probabilities of choosing each strategy in the candidate pool are summed to 1. These probabilities are initially equal (1/*K* for *K* total number of strategies in the pool) and are then gradually adapted during evolution, based on the *Success* and *Failure Memories* over the previous *LP* generations. The adaptation of the probabilities take place in such a fashion that, the larger the success rate for the *k-th* strategy in the pool within the previous *LP* generations, the larger is the probability of applying it to generate the trial vectors at the current generation.

Now, as far as the control parameters of DE are concerned, SaDE leaves *NP* as a user-specified parameter since it highly replies on the complexity of a given problem. The parameter *F*, in SaDE, is approximated by a normal distribution with mean value 0.5 and standard deviation 0.3, denoted by *N* (0.5, 0.3). A set of *F* values are randomly sampled from such normal distribution and applied to each target vector in the current population. This way, SaDE attempts to maintain both exploitation (with small *F* values) and exploration (with large *F* values) power throughout the entire evolution process. Following suggestions in [14], the control parameter *K* in the strategy "DE/current-to-rand/1" is hereby randomly generated within [0, 1] so as to eliminate one additional parameter. SaDE gradually adjusts the range of *CR* values for a given problem according to previous *CR* values that have generated trial vectors successfully entering the next generation. Specifically, it is assumed that *CR* obeys a normal distribution with mean value CR_m and standard deviation *Std* =0.1, denoted by $N(CR_m, Std)$, where CR_m is initialized as 0.5. The *Std* should be set as a small value to guarantee that most *CR* values generated by *N* (CR_m, *Std*) are between [0, 1], even when CR_m is near 0 or 1. Hence, the value of *Std* is set as 0.1.

The performance of SaDE was compared with the conventional DE and 3 adaptive DE-variants over a suite of 26 bound constrained numerical optimization problems, and the authors reported that, SaDE was more effective in obtaining better quality solutions, which are more stable with the relatively smaller standard deviation, and had higher success rates

5.7 Adaptive DE with DE/current-to-*p*best Mutation

In order to avoid the need for problem specific parameter tuning as well as improve the convergence characteristics of DE, an adaptive DE-variant, called JADE, was recently proposed [45]. The algorithm implements a new mutation strategy, referred by the authors as: DE/current-to-*p*best and uses an optional external archive to track the previous history of success and failure. It also updates the control parameters in an adaptive manner with generations.

The "DE/current-to-*p*best strategy is a generalization of the "DE/current-to-best/" strategy. Instead of only adopting the best individual in the "DE/current-to-best/1" strategy, the "current-to-*p*best/1" strategy also utilizes the information of other good solutions. Moreover, the recently explored inferior solutions are also incorporated in this strategy. The DE/current-to-*p*best/1 without external archive generates the donor vector as:

$$\vec{V}_{i,G} = \vec{X}_{i,G} + F_i \cdot (\vec{X}^{\,p}_{best,G} - \vec{X}_{i,G}) + F_i \cdot (\vec{X}_{r_1^i,G} - \vec{X}_{r_2^i,G}), \tag{30}$$

where $\vec{X}^{p}_{best,G}$ is randomly chosen as one of the top 100p% individuals of the current population with $p \in (0,1]$. F_i is the scale factor associated with the i-th individual and it is updated dynamically in each generation. JADE can optionally make use of an external archive, which stores the recently explored inferior solutions. Let **A** denote the archive of inferior solutions and **P** denote the current population. Then DE/current-to-$p$best/1 with external archive generates the donor vector as:

$$\vec{V}_{i,G} = \vec{X}_{i,G} + F_i \cdot (\vec{X}^{p}_{best,G} - \vec{X}_{i,G}) + F_i \cdot (\vec{X}_{r_1^i,G} - \vec{X}'_{r_2^i,G}), \tag{31}$$

where $\vec{X}_{i,G}$, $\vec{X}^{p}_{best,G}$, and $\vec{X}_{r_1^i,G}$ are selected from **P** as before in (30), but $\vec{X}'_{r_2^i,G}$ is selected at random from the union **P** ∪ **A,** of the current population and archive. The archive operation is made very simple to avoid significant computation overhead. The archive is remains initially empty. Then, after each generation, the parent solutions that fail in the selection process are added to the archive. If the archive size exceeds a certain threshold, then some solutions are randomly eliminated from the archive to keep the archive size fixed.

Among the control parameters, Cr for each individual and at each generation is randomly generated from a normal distribution $N(\mu_{Cr},0.1)$ and then truncated to [0, 1]. The mean of the normal distribution is updated as follows: let S_{Cr} be the set of all successful crossover probabilities Cr_i's at generation G. The mean μ_{Cr} is initialized to be 0.5 and then updated at the end of each generation as:

$$\mu_{Cr} = (1-c).\mu_{Cr} + c.mean_A(S_{Cr}), \tag{32}$$

where c is a positive constant between 0 and 1 and $mean_A(.)$ denotes the usual arithmetic mean.

Similarly for each individual the scale factor F_i is randomly generated from a Cauchy distribution $C(\mu_F,0.1)$ with location parameter μ_F and scale parameter 0.1. The generated value of F_i is then truncated to 1 if $F_i \geq 1$ or regenerated if $F_i \leq 0$. Let S_F be the set of all successful scale factors in generation G. μ_F is initialized to 0.5 and then updated as:

$$\mu_F = (1-c).\mu_F + c.mean_L(S_F), \tag{33}$$

where $mean_L(.)$ stands for Lehmer mean and is given by:

$$mean_L(S_F) = \frac{\sum_{F \in S_F} F^2}{\sum_{F \in S_F} F} \tag{34}$$

Note that no parameter adaptation takes place if c = 0. In other words the *life span* of a successful Cr_i or F_i is roughly 1/c generations; i.e., after 1/c generations, the old value of μ_{Cr} or μ_F is reduced by a factor of $(1-c)^{1/c} \to 1/e \approx 37\%$, when c is close to zero. JADE usually performs best with 1/c ∈ [5, 20] and p ∈ [5%, 20%] as demonstrated in [44].

5.8 DE with Neighborhood-based Mutation

The efficiency of most EAs depends on their extent of explorative and exploitative tendencies during the course of search. Exploitation means the ability of a search algorithm to use the information already collected and thus to orient the search more towards the goal while exploration is the process that allows introduction of new information into the population. Exploration helps the algorithm to quickly search the new regions of a large search-volume. A proper trade–off between exploration and exploitation is necessary for the efficient and effective operation of a population-based stochastic search technique like DE. Das *et al.* [46] proposed two kinds of neighborhood models for DE with an objective of balancing its explorative and exploitative tendencies in DE in the following way.

Suppose we have a DE population $P_G = [\vec{X}_{1,G}, \vec{X}_{2,G}, ..., \vec{X}_{NP,G}]$ at generation $G = G$. The vector indices are sorted only randomly (as obtained during initialization) in order to preserve the diversity of each neighborhood. Now for every vector $\vec{X}_{i,G}$ we define a neighborhood of radius k (where k is a non-zero integer from 0 to $(NP-1)/2$ as the neighborhood size must be smaller than the population size i.e. $2k+1 \le NP$), consisting of vectors $\vec{X}_{i-k,G}, ..., \vec{X}_{i,G}, ..., \vec{X}_{i+k,G}$. We assume the vectors to be organized on a ring topology with respect to their indices, such that vectors $\vec{X}_{NP,G}$ and $\vec{X}_{2,G}$ are the two immediate neighbors of vector $\vec{X}_{1,G}$. For each member of the population a local donor vector is created by employing the best (fittest) vector in the neighborhood of that member and any two other vectors chosen from the same neighborhood. The model may be expressed as:

$$\vec{L}_{i,G} = \vec{X}_{i,G} + \alpha \cdot (\vec{X}_{n\_best_i,G} - \vec{X}_{i,G}) + \beta \cdot (\vec{X}_{p,G} - \vec{X}_{q,G}), \quad (35)$$

where the subscript n_best_i indicates the best vector in the neighborhood of $\vec{X}_{i,G}$ and $p, q \in [i-k, i+k]$ with $p \ne q \ne i$. Similarly the global donor vector is created as:

$$\vec{g}_{i,G} = \vec{X}_{i,G} + \alpha \cdot (\vec{X}_{g\_best,G} - \vec{X}_{i,G}) + \beta \cdot (\vec{X}_{r_1,G} - \vec{X}_{r_2,G}), \quad (36)$$

where the subscript *g_best* indicates the best vector in the entire population at iteration $G = G$ and $r_1, r_2 \in [1, NP]$ with $r_1 \ne r_2 \ne i$. α and β are the scaling factors. Now we combine the local and global donor vectors using a scalar weight $w \in (0,1)$ to form the actual donor vector of the proposed algorithm:

$$\vec{V}_{i,G} = w.\vec{g}_{i,G} + (1-w).\vec{L}_{i,G}. \quad (37)$$

Clearly if $w = 1$ and in addition $\alpha = \beta = F$, the donor vector generation scheme in equation (37) reduces to that of DE/current-to-best/1. Hence the latter may be considered as a special case of this more general strategy involving both global and local neighborhood of each vector synergistically. The algorithm was referred to as DEGL (DE with Global and Local Neighborhoods) by the authors.

6. Hybrid DE Algorithms

Hybridization, in context to metaheuristics, primarily refers to the process of combining the best features of two or more algorithms together, to form a new algorithm that is expected to outperform its ancestors over application-specific or general benchmark problems. Over

the past few years, DE has been successfully hybridized with several other global optimization algorithms like PSO (Particle Swarm Optimization), ACS (Ant Colony Systems), AIS (Artificial Immune Systems), BFOA (Bacterial Foraging Optimization Algorithm), and SA (Simulated annealing), which have been briefly introduced in Chapter 1 and have been discussed in greater details in other chapters of the book. Also researchers attempted to embed different local search techniques in basic DE, to improve its exploitation abilities. In this section, we shall discuss the hybrid DE algorithms in two parts: first one will present the synergy between DE and other global search methods while the second one will review the blending of DE with local search algorithms.

6.1 Synergy of DE with Global Optimization Algorithms

The first synergy between DE and PSO was reported by Hendtlass, who proposed a combined swarm differential evolution algorithm [47] serving as a hybrid optimizer based on PSO and DE. In this optimizer, particle positions are updated only if their offspring have better fitness. DE acts on the particles in the PSO swarm at specified intervals. Zang and Xie [48] proposed another hybrid algorithm called DEPSO, in which the original PSO algorithm and the DE operator alternate at the odd iterations and at the even iterations. DEPSO achieved better convergence results than both the original algorithms over certain constrained optimization problems. In this section we shall present in details, a tightly coupled synergy of PSO and DE, called Particle Swarm Optimization with Differentially perturbed Velocity (PSO-DV) [49].

PSO-DV introduces a differential operator (borrowed from DE) in the velocity-update scheme of PSO. The operator is invoked on the position vectors of two randomly chosen particles (population-members), not on their individual best positions. Further, unlike the PSO scheme, a particle is actually shifted to a new location only if the new location yields a better fitness value, i.e., a selection strategy has been incorporated into the swarm dynamics. In the proposed algorithm, for each particle i in the swarm two other distinct particles, say j and k ($i \neq j \neq k$), are selected randomly. The difference between their positional coordinates is taken as a difference vector:

$$\vec{\delta} = \vec{X}_k - \vec{X}_j \tag{38}$$

Then the d-th velocity component ($1 < d < D$) of the target particle i is updated as:

$$\begin{aligned} v_{i,d}(t) &= \omega.v_{i,d}(t-1) + \beta.\delta_d + \varphi_2.rand2_{i,d}(0,1).(p_d^g - x_{i,d}(t-1)) \quad \text{if } rand_d(0,1) \leq Cr \\ &= v_{i,d}(t-1), \text{otherwise,} \end{aligned} \tag{39}$$

where Cr is the crossover probability, δ_d is the d-th component of the difference vector defined earlier, and β is a scale factor in [0, 1]. Following PSO convention, we denote the current iteration number or time step as t here. In essence the cognitive part of the velocity update formula in PSO is replaced with the vector differential operator to produce some additional exploration capability. Clearly, for $Cr \leq 1$, some of the velocity components will retain their old values. Now, a new trial location $\vec{Tr}_i$ is created for the particle by adding the updated velocity to the previous position $\vec{X}_i$:

$$\vec{Tr}_i = \vec{X}_i(t-1) + \vec{V}_i(t) \tag{40}$$

The particle is placed at this new location only if the coordinates of the location yield a better fitness value. Thus if we are seeking the minimum of the objective function $f(\vec{X})$, then the target particle is relocated as follows:

$$\vec{X}_i(t) = \vec{Tr}_i \qquad \text{if } f(\vec{Tr}_i) \le f(\vec{X}_i(t))$$
$$\vec{X}_i(t) = \vec{X}_i(t-1) \qquad \text{otherwise} \tag{41}$$

Therefore, every time its velocity changes, the particle either moves to a better position in the search space or sticks to its previous location. The current location of the particle is thus the best location it has ever found. In other words, unlike the classical PSO, in the present scheme, $p^l_{i,d}$ always equals $x_{i,d}$. So the cognitive part involving $\left|p^l_{i,d} - x_{i,d}\right|$ is automatically eliminated in our algorithm. If a particle gets stagnant at any point in the search space (i.e., if its location does not change for a predetermined number of iterations), then the particle is shifted by a random mutation (explained below) to a new location. This technique helps escape local minima and also keeps the swarm "moving":

If ($(\vec{X}_i(t-1) = \vec{X}_i(t) = \vec{X}_i(t+1) = ... = \vec{X}_i(t+N-1))$ and ($f(\vec{X}_i(t+N) \neq f^*$) **then**

for (r = 1 to D)

$$x_{i,r}(t+N) = x_{\min,j} + rand_r(0,1).(x_{\max,r} - x_{\min,r}), \tag{42}$$

where f^* is the global minimum of the fitness function, N is the maximum number of iterations up to which stagnation can be tolerated and ($x_{\max,\,r}$, $x_{\min,\,r}$) define the permissible bounds of the search space in the r-th dimension.

Among other reported synergies of PSO and DE, in [50], Moore and Venayamoorthy proposed a new hybrid of DE and PSO, which is similar in spirit to the algorithm proposed in [51], but the DE and PSO in it are DE/*rand*/2/*bin* and a modified PSO with '*Ring*' topology respectively. In [51], Liu *et al*. proposed a similar DEPSO and used it to train artificial neural networks. Like [48], the PSO in this hybrid optimizer is also based on *Gbest* model; however, the DE in it is DE/*target-to-best*/*1*/*bin*. In particular, this hybrid also adopts a chaotic local search to improve its local exploitation ability. In 2004, Kannan *et al*. [52] proposed a distinctive DEPSO (named C-PSO in [52]). The DE algorithm in it is employed to select three control parameters on-line for PSO. In other words, DE serves as a meta-optimizer for the optimization of PSO's search behavior. Recently Hao *et al*. [53] constructed a new hybrid optimizer, where DE and PSO are regarded as two operators to generate candidate solutions, and they act on the level of dimensional components of individuals.

In [54], Omran *et al*. presented two hybrids of DE and PSO. The first DEPSO (named DEPSO-OES) is somewhat similar to the hybrid described in [53]. The DE (DE/*rand*/*1*/*bin*) and PSO-cf (PSO with constriction factor) in it also alternate in a stochastic way, but both DE and PSO act on the level of a whole individual, that is to say, each individual at each generation has only one updating method (DE or PSO). Besides, the probability for controlling the selection of updating method and the scaling factor in DE are dynamic and adaptive. The second hybrid method combined the bare bones PSO (BBPSO) proposed by Kennedy [55] and DE in an embedding way. Xue *et al.* described another scheme of mixing DE operators with PSO in [56].

Das *et al.* [57] modified the selection mechanism of the classical DE family by using the concepts of SA such that the probability of accepting the inferior solutions may be dynamically altered with iterations. Unlike the *greedy* selection strategy employed in the classical DE, the Annealed DE (AnDE) [57] incorporates a typical SA-type selection mechanism that conditionally accepts the inferior solutions to the next generation. Suppose

at generation $G = G$, , the i-th chromosome is $\vec{X}_{i,G}$ and the corresponding trial vector, created through the DE-type mutation and crossover operations is $\vec{U}_{i,G}$. Now if $f(\vec{U}_{i,G}) \leq f(\vec{X}_{i,G})$, i.e. if the trial vector is better than or comparable to the target w.r.t the objective function, then $\vec{X}_{i,G}$ is surely replaced in the next generation with $\vec{U}_{i,G}$. But even if $f(\vec{U}_{i,G}) > f(\vec{X}_{i,G})$, $\vec{U}_{i,G}$ may replace the parent chromosome $\vec{X}_{i,G}$ with a probability of:

$$P_i = \exp\left[-\left(\frac{f(\vec{U}_{i,G}) - f(\vec{X}_{i,G})}{T}\right)\right] \tag{43}$$

The SA algorithm requires an annealing schedule for decreasing the control temperature T from an intial value T_0 to a final value T_f. This allows it to escape the local optima at the early stages of the search and to efficiently hill-climb as the temperature approaches zero. We should note that as $T \rightarrow 0$, P_i also approaches zero. The AnDE algorithm employs a very common temperature decrement rule known as the exponential cooling schedule (ECS), proposed by Kirkpatrick *et al.* [58]. According to this rule, the temperatures T_G and T_{G+1} over two successive time steps are related as:

$$T_{(G+1)} = \alpha . T_G , \tag{44}$$

where α is constant close to but smaller than 1.

Biswas *et al.* [59] proposed a synergistic coupling of DE and BFOA. In [83] the computational chemotaxis of BFOA, which may also be viewed as a stochastic gradient search, has been coupled with DE type mutation and crossing over of the optimization agents leading to the new hybrid algorithm called CDE (Chemotactic Differential Evolution).

6.2 Synergy of DE with Local Search Methods

Local search algorithms primarily explore a small neighborhood of a candidate solution in the search space until a locally optimal point is found or a time bound is elapsed. In this section we shall discuss a crossover based adaptive Local Search (LS) operation to improve the performance of the classical DE following [60]. Typically in LS, every candidate solution has more than one neighbour solution; the choice of which one to move to is taken using only information about the solutions in the neighbourhood of the current one, hence the name local search. If the choice of the neighbouring solution is done by taking the one locally maximizing the criterion, the metaheuristic takes the name *hill-climbing*. The authors in [60] propose an LS, whose length of the search can be adjusted adaptively using a hill-climbing heuristic. The incorporation of a crossover-based local search (XLS) with adaptive length (adaptive length XLS, shortened as AHCXLS) in DE resulted into a DE-variant called by the authors: DEahcSPX, where SPX is the simplex-based crossover scheme proposed by Tsutsui *et al.* for real-coded GAs [61].

Below we provide the pseudo-codes for AHXLS, DEahcSPX and the SPX algorithms.

AHCXLS (*I*, n_p)
Step 1: P[1] = *I* ;
Step 2: Repeat from *i* = 2 to n_p times
Step 3: P[i] = select random individuals from the population.

Step 4: End Repeat
Step 5: **C = *Crossover***(P)
Step 6: If C is better than P[1],
P[1] = C;
Step 7: Else
Return (P[1]).
Step 8: Go to Step 5.

DEahcSPX

Step 1: Generate an Initial Population ***P****(t)*
Step 2: ***Evaluate*** the fitness of ***P****(t)*
Step 3: *B* = ***Best_Index*** (***P****(t)*)
Step 4: ***P****(t)*.[*B*] = AHCXLS (***P****(t)*.[*B*], n_p)
Step 5: for each individual *I* in ***P****(t)*
Reproduce an offspring *J* from *I*.
Step 6: ***P****(t*+1*)* = ***P****(t*+1*)* $\cup$ $Select(I, J)$
Step 7: Ste *t* = *t*+1
Step 8: Repeat Steps 3 to 8 until termination criteria is met.

SPX

Step 1: Choose n_p parents $\vec{X}_i(t)$, $i = 1,2,....,n_p$ according to the generational model used and calculate their center of mass $\vec{O}$ as:

$$\vec{O} = \frac{1}{n_p}\sum_{i=1}^{n_p}\vec{X}_i(t). \tag{45}$$

Step 2: Generate random number $\vec{r}_i$ as: $\vec{r}_i = u^{\frac{1}{i+1}}, i = 1,2,...,n_p - 1$ where u is a uniform random number and $u \in [0,1]$.

Step 3: Calculate $\vec{Y}_i$ and $\vec{C}_i$ as:

$$\vec{Y}_i = \vec{O} + \varepsilon.(\vec{X}_i(t) - \vec{O}), i = 1,2,....n_p \tag{46}$$

$$\vec{C}_i = \begin{cases} 0 \quad for\, i = 1, \\ r_{i-1}(\vec{Y}_{i-1} - \vec{Y}_i + C_{i-1}) \quad for\, i = 2,....,n_p. \end{cases} \tag{47}$$

where $\varepsilon = 1.0$ is the expansion rate, a control parameter of SPX.

Step 4: Generate and offspring $\vec{C}$ as:

$$\vec{C} = \vec{Y}_{n_p} + \vec{C}_{n_p} \tag{48}$$

The experimental results reported by Noman and Iba [60] indicate that the DEahcSPX could outperform the classical DE (more precisely the DE/rand/1/bin variant) in terms of convergence speed over a set of carefully chosen numerical benchmarks. The overall performance of the adaptive LS scheme was reported better than the other crossover-based LS strategies and the overall performance of the newly proposed DE algorithm was shown to be superior to or at least comparable with some other Memetic Algorithms (MAs) [62] selected from literature. The proposed LS scheme was also found prospective for adaptive DE variants.

Yang *et al.* [63] proposed a hybridization of DE with the Neighborhood Search (NS), which appears as a main strategy underpinning an EP [64]. The resulting algorithm, known as NSDE, performs mutation by adding a normally distributed random value to each target-vector component in the following way:

$$\vec{V}_{i,G} = \vec{X}_{r_1^i,G} + \begin{cases} \vec{d}_{i,G}.N(0.5,0.5) \text{ , if } rand_i(0,1) < 0.5 \\ \vec{d}_{i,G}.\delta \text{ , otherwise,} \end{cases} \quad (49)$$

where $\vec{d}_{i,G} = \vec{X}_{r_2^i,G} - \vec{X}_{r_3^i,G}$ is the usual difference vector, $N(0.5, 0.5)$ denotes a Gaussian random number with mean 0.5 and standard deviation 0.5, and δ denotes a Cauchy random variable with scale parameter $t = 1$. Recently Yang *et al.* [65] used a Self-adaptive NSDE in the cooperative coevolution framework that is capable of optimizing large scale non-separable problems (up to 1000 dimensions). They proposed a random grouping scheme and adaptive weighting for problem decomposition and coevolution. Somewhat similar in spirit to the present paper is the study by Yang *et al.* [66] on self-adaptive differential evolution with neighborhood search (SaNSDE). SaNSDE incorporates self-adaptation ideas from the Qin *et al*'s SaDE [22] and proposes three self-adaptive strategies: self-adaptive choice of the mutation strategy between two alternatives, self-adaptation of the scale factor *F*, and self-adaptation of the crossover rate *Cr*. In contrast to Yang *et al*.'s works on NSDE and SaNSDE, in the topological neighborhood-based mutation scheme proposed in [46], the authors keep the scale factor non-random and use a ring-shaped neighborhood topology (inspired by PSO [67]), defined on the index graph of the parameter vectors, to derive a local neighborhood-based mutation model. Also instead of *F* and *Cr*, the weight factor that unifies two kinds of mutation models, have been made self-adaptive in one of the variants of DE/target-to-best/1 scheme, proposed in [46].

Memetic Algorithms (MAs) represent one of the recent growing areas of research in evolutionary computation. The term MA is now widely used as a synergy of evolutionary or any population-based approach with separate individual learning or local improvement procedures for problem search. Neri and Tirronen [68] proposed a DE-based Memetic Algorithm (MA), which employs within a self-adaptive scheme, two local search algorithms. The algorithm was referred by authors as the Scale Factor Local Search Differential Evolution (SFLSDE). These local search algorithms aim at detecting a value of the scale factor corresponding to an offspring with a higher fitness, while the generation is executed. The local search algorithms thus assist in the global search and generate offspring with a higher fitness, which are subsequently supposed to promote the generation of enhanced solutions within the evolutionary framework. In [69] Tirronen *et al*. propose a DE-based MA employing three local search algorithms coordinated by means of fitness diversity adaptation and a probabilistic scheme for designing digital filters, which aim at detecting defects of the paper produced during an industrial process. In [70] Caponio *et al*. introduce the concept of super-fit adaptation is introduced and employed for coordination of the various algorithmic components in memetic DEs. This adaptive scheme controls the performance of the best individual with respect to the average performance of other individuals of the population.

7. DE-variants for Discrete and Binary Optimization

Although DE was devised mainly for real parameter optimization, over the years researchers have tried to modify it for tackling binary and discrete optimization problems as

well. In the early days of DE research, Lampinen and Zelinka first focused in this direction through their conference article in MENDEL'99 [71]. For handling of integer variables, they recommended truncating the parameter values for objective function evaluation such that the population of DE still works with floating-point values. They pointed out that although such truncation changes the effective objective function landscape from DE's point of view by introducing flat areas to the fitness landscape, DE's self-adaptive reproduction scheme is well able to move across to those flat areas. A discrete value is optimized indirectly so that DE optimizes an integer value (index) that points to the actual discrete value. First, the discrete set of available values is arranged to an ascending sequence, and then an index is assigned to refer each available value. DE works with these indices by optimizing the index like any integer variable. However, for objective function evaluation the actual discrete value pointed by the index is used.

In [72] Pan *et al*. proposed a Discrete Differential Evolution (DDE) algorithm to solve the permutation flowhop scheduling problem with the makespan criterion. As a mutation operator in DDE, a destruction and construction procedure is employed to generate the mutant population. The authors also embedded a referenced local search in the DDE to further improve the solution quality. In [73] DDE was applied to solve the single machine total weighted tardiness problem with sequence dependent setup times

Below we present a binary DE-variant that can operate in binary problem spaces without deviating from the basic search mechanism of the classical DE. The algorithm was named by its authors as the *Angle Modulated DE (AMDE)* [74] as it employs a trigonometric function as a bit string generator. The angle modulation scheme draws inspiration from the domain of telecommunication engineering [38]. The trigonometric generating function used in the angle modulation function is a composite sinusoidal function which may be given as:

$$g(x) = \sin(2\pi(x-a)\times b\times \cos A)) + d\,, \tag{50}$$

where $A = 2\pi\times c\times(x-a)$ and x is a single element from a set of evenly separated intervals determined by the required number of bits that need to be generated. For example, a selection of 10 evenly spaced points between 0 and 10 is needed in order to generate 10 bit values. Figure 11 graphically represents the bit string generating function with default values a =0, b = 1, c = 1 and d = 0.

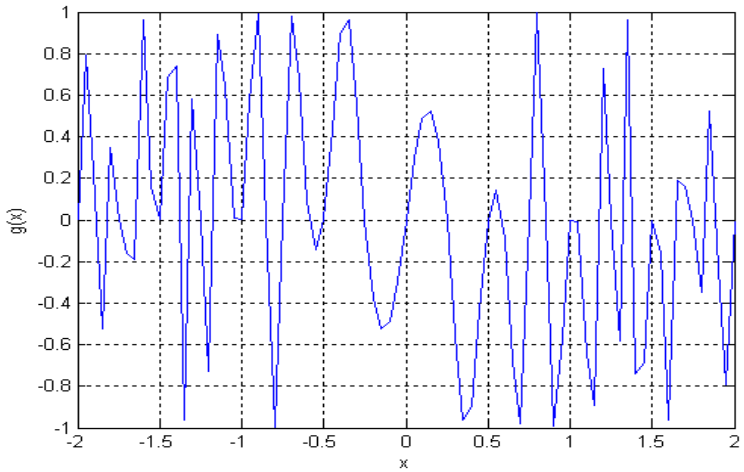


Fig. 11: Bit string generating function (a =0, b = 1, c = 1, and d = 0)

The optimization algorithm is then applied to evolve a 4-dimensional tuple (a, b, c, d). This tuple represents the coefficient values for equation (50). The optimization process evolves the tuple parameters instead of actually evolving the bit string. After an iteration of the optimization algorithm, the tuple parameters are substituted back into (50). The resulting generating function is sampled at evenly spaced intervals and for each interval a bit value is recorded. The generated bit vector represents the potential solution within the binary problem space of the original problem. The generation of the bit string vector values is a simple procedure. The sample points are fed into the generating function and the output of equation (50) is evaluated. If the resulting output value at the sample point is positive, a bit value of 1 is recorded, otherwise a bit value of 0 is recorded. The pseudo code of the AMDE algorithm has been provided below:

Angle Modulated Differential Evolution Algorithm

Initialize a population and set control parameter values
repeat
Select required individuals for reproduction scheme
Produce an offspring individual
Evaluate the fitness of the offspring individual by generating
bit string and passing to fitness function
if $f_{AMDE}(offspring) \leq f_{AMDE}(parent)$ **then**
Replace parent individual with offspring
else
Retain parent individual
end if
until Stopping condition is met

where $f_{AMDE}()$ is the fitness function of the AMDE algorithm. The AMDE algorithm was found to outperform the binary PSO (binPSO) algorithm over standard binary optimization problems.

8. Analytical Studies on DE

Compared to the plethora of works concerning the empirical study of parameter selection and tuning process in DE, not much research has so far been devoted to theoretically analyze the search mechanism and convergence properties of DE and this area remains largely open for the prospective future researchers. Below we discuss some of the analytical results so far obtained on DE.

8.1 Population Variance and Explorative Power

Some significant theoretical results on DE were first reported in [75] and then extended in [76, 28] by Zaharie, where she theoretically analyzed the influence of the variation operators (mutation and recombination) and their parameters on the expected population variance. In absence of selection pressure, Zaharie tracked the population diversity by tracking a single parameter of the population. To simplify her analysis, Zaharie also relaxed DE's usual condition that the base and target vectors be different, although the requirement that base and difference vectors be distinct was retained. By dropping the demand that at least one trial parameter be inherited from the donor, Zaharie assumed that the cross over rate Cr is equivalent to the absolute probability p_{Cr} that a component of the target vector is

exchanged with that of the donor vector. With these few simplifying assumptions, Zaharie proved that if $P_{x,0}$ be the initial population at $G = 0$, and $P_{x,G}$ be the same at generation $G = G$, then the variance of these two populations are linked by the following equation [75]:

$$E(Var(P_{x,G})) = \left(2.F^2.p_{Cr} - \frac{2.p_{Cr}}{NP} + \frac{p_{Cr}^2}{NP} + 1 \right)^G .Var(P_{x,G}), \quad (51)$$

Note that according to [77], the expected population variance for an ES after G generations without fitness-based selection and with a mutation based on addition of a random vector (having mutually independent normally distributed components with zero mean and a variance σ^2) and dominant recombination (global discrete recombination) is:

$$E(Var(P_{x,G})) = \left(1 - \frac{1}{NP}\right)^G .Var(P_{x,0}) + NP.\left[1 - \left(1 - \frac{1}{NP}\right)^G\right].\sigma^2, \quad (52)$$

where NP is the population size. For large NP, (52) reduces to:

$$\mathrm{E}(Var(P_{x,G})) \approx Var(P_{x,0}) + G.\sigma^2. \quad (53)$$

Also for large NP, (53) takes approximately the following form:

$$E(Var(P_{x,G})) \approx \left(2.F^2.p_{Cr} + 1\right)^G .Var(P_{x,G}). \quad (54)$$

Since $\left(2.F^2.p_{Cr} + 1\right) > 1$, the expected population variance (after applying mutation and recombination operators) can on the average be greater for DE than that for ES (under identical operating conditions) as analyzed in [77]. Thus DE can introduce more diversity in the population and has greater explorative power than some of the classical ES algorithms.

8.2 Evolutionary Search-dynamics in DE

The first theoretical studies on the evolutionary search-dynamics of DE were carried out by Dasgupta *et al*. in [78, 79]. The authors propose a simple mathematical model of the underlying evolutionary dynamics of a one-dimensional DE-population (evolving with the DE/rand/1/bin algorithm) [78]. The model reveals that the fundamental dynamics of each search-agent (one-dimensional parameter vector) in DE employs the gradient-descent type search strategy (although it uses no analytical expression for the gradient itself), with a learning rate parameter that depends on control parameters like scale factor F and crossover rate Cr of DE. The treatment is outlined below.

Suppose $f(x)$, function of single variable x, is to be optimized using the DE Algorithm. Let $\{x_1, x_2, \ldots\ldots x_{NP}\}$ be a set of trial solutions forming the population subjected to DE search where NP denotes the population size. In order to validate the analysis, the authors made certain assumptions, like:

i) The population of NP trial solutions is limited within a small region i.e. individual trial solutions are located very close to each other (say initial population variance is approximately 0.5). According to [11], this is usually the case during the latter stages of the search when the parameter vectors concentrate in a thick cluster around the global optimum, and especially when the scaling factor F is set at 0.5.

ii) The analysis is confined to flatter portions of the fitness landscape surrounding an optimum; so that the function-gradient associated with each vector may have moderate values (i.e. the gradient is not very large for all vectors).

iii) Dynamics is modeled in continuous time.

iv) The objective function was assumed to be continuous and locally uni-modal (containing a single optimum at the region of interest). The assumption was made to keep the mathematics less rigorous.

If the DE population may be modeled as a continuous-time, dynamic system, then the expectation value of the velocity of an individual point on the fitness landscape may be given as:

$$E(\frac{dx_m}{dt}) = -\frac{k}{8}Cr.\{(2F^2+1)Var(x) + (x_{av} - x_m)^2\}f^{'}(x_m) + \frac{1}{2}Cr.(x_{av} - x_m), \qquad (55)$$

where $f^{/}(x_m)$ is the derivative of the objective function at $x = x_m$. Also if x_{av} denote the centroid (mean of all points) of the current population and $x_{av} = \frac{1}{NP}\sum_{m=1}^{NP} x_m$ and $\varepsilon_m = x_{av} - x_m =$ deviation of individual from average, then expected velocity of the centroid of the population may be given by,

$$E(\frac{dx_{av}}{dt}) = -\frac{k}{8}Cr.(2F^2+1)Var(x)f_{av}^{'} - \frac{k}{8}Cr.(\frac{1}{NP}\sum_{m=1}^{NP}\varepsilon_m{}^2 f^{'}(x_{av} + \varepsilon_m)), \qquad (56)$$

From equation (55), we may write,

$$E(\frac{dx_m}{dt}) = -\alpha_{DE}f^{'}(x_m) + \beta_{DE}, \qquad (57)$$

where, $\alpha_{DE} = -\frac{k}{8}CR\{(2F^2+1)Var(x) + (x_{av} - x_m)^2\}$ and $\beta_{DE} = \frac{1}{2}CR(x_{av} - x_m)$

The classical gradient descent search algorithm is given by the following dynamics (continuous) in single dimension (see Chapter 1)

$$\frac{d\theta}{dt} = -\alpha.G + \beta, \qquad (58)$$

where α is the learning rate and β is the momentum.

The resemblance of equations (57) and (58) is not difficult to recognize and it suggests that, the dynamics of actual DE uses some kind of estimation for the gradient of the objective function. In equation (57), $-\alpha_{DE}f^{'}(x_m)$ term on the R.H.S. is responsible for moving along the direction of the negative gradient, whereas β_{DE} represents a component of velocity of a trial solution towards the mean vector (center of mass) of the population.
Evidently very near to an optimum, when $f^{'}(x_m) \to 0$,

$$E(\frac{dx_m}{dt}) \approx \beta_{DE} = \frac{1}{2} CR(x_{av} - x_m) \tag{59}$$

It is due to the gradient descent type search strategy, that DE converges much faster than some variants of GA or PSO over uni-modal benchmarks [80]. The stability and convergence-behavior of the proposed dynamics was analyzed in the light of Lyapunov's stability theorems very near to the islolated equilibrium points during the final stages of the search in [79] and the rate of convergence on smooth uni-modal functions were found to largely depend on *Cr*. However, proving the probabilistic convergence of DE on even very simple objective functions is still an open problem for the theorists working with EAs.

8.3 Role of Crossover

Very recently in [81, 82], the influence of the crossover rate on the distribution of the number of mutated components and on the probability for a component to be taken from the mutant vector (mutation probability) is theoretically analyzed for several variants of crossover, including classical binomial and exponential strategies in DE. For each crossover variant the relationship between the crossover rate and the mutation probability is identified and its impact on the choice and adaptation of control parameters is analyzed both theoretically and numerically. With numerical experiments, the author illustrates the fact that the difference between binomial and exponential crossover variants is mainly due to different distributions of the number of mutated components. On the other hand, the behavior of exponential crossover variants was found to be more sensitive to the problem size than the behavior of variants based on binomial crossover.

8.4 Runtime Complexity

Runtime-complexity analysis of the population-based stochastic search techniques like DE is a critical issue by its own right. In [83] Zielinski *et al.* first investigated the runtime complexity of DE for various stopping criteria, both theoretically and empirically. The authors pointed out that in each generation of DE a loop over *NP* is conducted, containing a loop over *D*. Since the mutation and crossover operations are performed at the component level for each DE vector, the number of fundamental operations in DE/rand/1/bin is proportional to the total number of loops conducted until the termination of the algorithm. Thus, if the algorithm is stopped after a fixed number of generations G_{max}, then the runtime complexity is $O(NP \cdot D \cdot G_{\max})$. Moreover the authors also inferred that maximum distance criterion *MaxDist* yields best overall performance for the DE algorithm. Note that this criterion stops the execution of the algorithms if the maximum distance from every vector to the best population member is below a given threshold.

9. Summary

With the increasing complexity of real world optimization problems, demand for robust, fast, and accurate optimizers is on the rise among researchers from various fields. DE emerged as a simple and efficient scheme for global optimization over continuous spaces more than a decade ago. Over the past few years, many researchers have contributed to make it a general and fast optimization method for any kind of objective function by twisting and tuning the various constituents of DE, i.e. initialization, mutation, diversity enhancement, and selection as well as by the choice of the control variables, even though

the NFL [14] suggested already that such a cure-all-optimization algorithm could not exist. Nonetheless the existing literature clearly indicates that DE exhibits remarkable performance in optimizing a wide variety of multi-dimensional, multi-objective and multi-modal optimization problems.

DE has three main control parameters: the scale factor *F*, the crossover rate *Cr* and the population size NP. Over the years, several methods have been proposed to adapt these parameters, especially *F* and *Cr*, so that the users may get rid of the problem-specific hand tuning of such parameters. Studies with the population size are scarce and usually *NP* is recommended to be 10 times the number of problem dimensions *D*. Since DE comes with a number of alternative strategies for generating the donor vectors, attempts have been made by the researchers to make DE select the most suitable mutation strategy for a particular problem based on its previous history of success and failure (e.g. SaDE). Attempts have also been made to incorporate the concept of a topological neighborhood of the vectors for generating a donor vector that can balance between the explorative and exploitative tendencies of DE. In this respect particularly DEGL and to some extent the DE/ current-to-*p*best mutation scheme that comes along with JADE owe a lot to the *lbest* PSO model described in Chapter 5. In ODE, opposition numbers have been as an add-on feature to improve any existing DE scheme.

DE has been successfully hybridized with many other global optimization algorithms like PSO, BFOA, SA etc. as well as local search heuristics. Modifications of DE have been proposed to make it suitable for tackling discrete and binary optimization problems. In particular, the angle modulated DE can operate through binary problem spaces without getting much deviated from the basic working principles of DE. However, this area needs substantial research effort in order to solve various complex discrete and combinatorial problems with DE.

Unlike the huge theoretical studies that exist for GA, ES etc., the theory of DE is still much in its preliminary stage. While analyzing the population variance and explorative power of DE in absence of selection pressure, Zaharie has shown that the explorative power of DE could be more than some simple ES algorithms. Some bounds for the control parameter F and its effects on the performance on DE were also analyzed. The population dynamics of DE in uni-dimensional search space have also been analyzed under some simplifying assumptions. It has been demonstrated that under many simplifying assumptions, DE has a striking similarity with the classical gradient descent type algorithms, although the former does not use any analytical expression for the function gradient. The role of crossover

This article attempted to provide an overall picture of the state-of-the-art research on and with DE. Starting with a comprehensive introduction to the basic steps of the DE algorithm, it discussed the different schemes of parameter control and adaptation for DE and then overviewed several promising variants of the conventional DE. Role of the exponential and binomial crossovers in DE have also been analyzed. Preliminary studies on the runtime complexity of DE and effects of different stopping criteria on runtime have been Undertaken. The content of the present chapter indicates the fact that DE will continue to remain a vibrant and active field of multi-disciplinary research in the years to come.

Review Problems

1. Given $Cr = 0.4$, consider the dimension of DE-vectors $D = 4$. Remove the restriction on binomial crossover that at least one component of the trial vector must be inherited from the donor vector.

   (a) Now generate a random number in [0, 1] for each component of the new trial vector. What is the probability that the trial vector is same as the target vector? [Ans. 0.0256]

   (b) What is the probability that the two components of the target vector will be added to the trial vector? [Ans. 0.16]

   (c) What is the probability that at least two components of the target vector will appear in the trial vector? [Ans. 0.48]

2. Consider a sphere function in 4 dimensions given by $f(\vec{X}) = \sum_{i=1}^{4} x_i^2$ for real x_i. What is the probability that the trial vector is fitter than the target vector? [Ans. 0.0625]

   [Hint: For minimization problems, the smaller the value of each field of the sphere function, the better is the fitness value of the field. Now the probability that one field say the field x_1 takes the smaller value from the donor and the target is 0.5. So, the probability that all the fields will pick smaller values from the respective fields of the donor and the target is $(0.5)^4$]

3 Let p be the probability that a field of the trial vector is picked up from the respective field of the target and q is the probability that the same field of the trial vector is picked up from the donor vector. What is the probability that r no. of fields of the trial are picked up from the target and $D - r$ fields from the donor? Given $Cr = 0.6$, $D = 10$, $r = 4$.
[Hint: The required probability will be ${}^{D}C_r p^r q^{D-r}$].

4. Show that the number of possible distinct trial vectors that can be created with DE's uniform (binomial) cross-over is [34]:

$$n_{trial} = 2^D.NP.(NP^3 - 3.NP^2 + 2NP)$$

for $0 < Cr < 1$.

5. The exponential crossover was proposed in the first version of DE [20] and has been illustrated in Section 2.3 of the current chapter. It is similar with the two-point crossover where the first cut point is randomly selected from {1, 2,..., D} and the second point is determined such that L consecutive components (counted in a circular manner) are taken from the mutant vector. In their original paper [20], Storn and Price suggested to choose L from {1,2,…,D} such that $Prob(L = h) = Cr^h$. It is easy to check that this is not a probability distribution on {1,2,…,D} but just a relationship which suggest that the probability of mutating h $1 \le h \le D$ components increases with the parameter Cr and decreases with the value of h. Following the pseudo-code given in Section 2.3, compute the expectation value of the random variable L.

[Ans: $E(L)=\frac{1-Cr^n}{1-Cr}$]

6. Starting from the assumptions listed in Section 7.2, derive the expression for the expected value of the velocity of the centroid (mean vector) of a one-dimensional DE population as given in equation (56)

[Hint: Now, $x_{av}=\frac{1}{NP}\sum_{i=1}^{NP}x_i=\frac{1}{NP}\sum_{m=1}^{NP}x_m$

$$\Rightarrow \frac{dx_{av}}{dt}=\frac{d}{dt}(\frac{1}{NP}\sum_{m=1}^{NP}x_m)=\frac{1}{NP}\sum_{m=1}^{NP}\frac{dx_m}{dt}$$

$$\Rightarrow E(\frac{dx_{av}}{dt})=E(\frac{1}{NP}\sum_{m=1}^{NP}\frac{dx_m}{dt})=\frac{1}{NP}\sum_{m=1}^{NP}E(\frac{dx_m}{dt})$$

$$E(\frac{dx_{av}}{dt})=\frac{1}{NP}(-\frac{k}{8}CR\sum_{m=1}^{NP}\{(2F^2+1)Var(x)+(x_{av}-x_m)^2\}$$

$$f^{'}(x_m)+\frac{1}{2}CR\sum_{m=1}^{NP}(x_{av}-x_m))$$

Now, $\sum_{m=1}^{NP}(x_{av}-x_m)=0$

$$\therefore E(\frac{dx_{av}}{dt})=\frac{1}{NP}(-\frac{k}{8}CR\sum_{m=1}^{NP}\{(2F^2+1)Var(x)+(x_{av}-x_m)^2\}f^{'}(x_m)$$ Let us denote $\varepsilon_m=x_{av}-x_m=$ deviation of individual from average.

$$\therefore E(\frac{dx_{av}}{dt})=-\frac{k}{8}CR(2F^2+1)Var(x)$$

$$(\frac{1}{NP}\sum_{m=1}^{NP}f^{'}(x_m))-\frac{k}{8}CR(\frac{1}{NP}\sum_{m=1}^{NP}\varepsilon_m{}^2f^{'}(x_m))$$

$$\Rightarrow E(\frac{dx_{av}}{dt})=-\frac{k}{8}CR(2F^2+1)Var(x)$$

$$(\frac{1}{NP}\sum_{m=1}^{NP}f^{'}(x_m))-\frac{k}{8}CR(\frac{1}{NP}\sum_{m=1}^{NP}\varepsilon_m{}^2f^{'}(x_{av}+\varepsilon_m))$$

$$\therefore E(\frac{dx_{av}}{dt})=-\frac{k}{8}CR(2F^2+1)Var(x)f_{av}{}^{'}-\frac{k}{8}CR(\frac{1}{N}\sum_{m=1}^{NP}\varepsilon_m{}^2f^{'}(x_{av}+\varepsilon_m))$$

where, $f^{'}{}_{av}=\frac{1}{NP}\sum_{m=1}^{NP}f^{'}(x_m)=$ average of gradients for trial solution points on fitness landscape.]

Computer Project:

7. Frequency-modulated (FM) sound synthesis plays an important role in several modern music-systems. The goal is to introduce a system that can automatically generate sounds similar to the target sounds [84, 85]. It consists of an FM synthesizer, a DE optimizer, and a feature extractor. The system architecture is shown in Figure 12. The target sound is a *.wav* file. The DE algorithm initializes a set of parameters and the FM synthesizer generates the corresponding sounds. In the feature extraction step, the dissimilarities of features between the target sound and synthesized sound are used to compute the fitness value. The process continues until synthesized sounds become very similar to the target.

The specific instance of the problem discussed here involves determination of six real parameters: $\vec{X} = \{a_1, \omega_1, a_2, \omega_2, a_3, \omega_3\}$ of the FM sound wave given by equation (60) for approximating it to the sound wave given in (61) where $\theta = 2\pi/100$. The parameters are defined in the range [-6.4, +6.35]. The formula for the estimated sound wave and the target sound wave may be given as:

$$y(t) = a_1.\sin(\omega_1.t.\theta + a_2.\sin(\omega_2.t.\theta + a_3.\sin(\omega_3.t.\theta))) \quad (60)$$

$$y_0(t) = 1.0.\sin(5.0.t.\theta - 1.5.\sin(4.8.t.\theta + 2.0.\sin(4.9.t.\theta))) \quad (61)$$

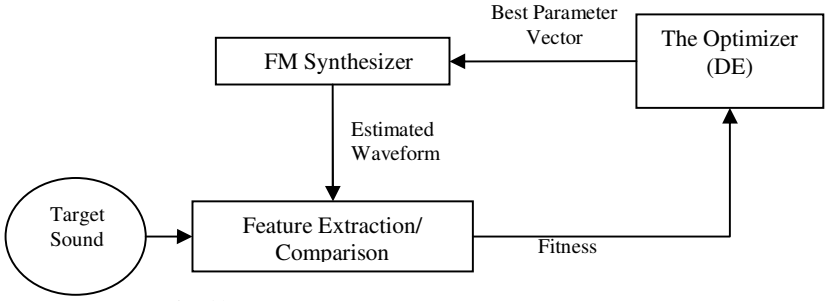


Fig. 12: Architecture of the optimization system.

The goal is to minimize the sum of squared errors between the estimated sound and the target sound, as given by (62). This problem involves a highly complex multi-modal function having strong epistasis (interrelation among the variables), with optimum value 0.0.

$$f(\vec{X}) = \sum_{t=0}^{100} (y(t) - y_0(t))^2. \quad (62)$$

Develop a C or MATLAB program to solve the above engineering optimization problem with DE. Compare the final accuracy of the five classical DE variants given in equations (3), (7) – (10). Let all the DE variants start from the same initial population. Run all the algorithms until the number of fitness function evaluations (FEs) reaches 10^4. Take $NP = 60, Cr = 0.5,$ and $F = 0.8,$ for all DE variants.

References

1. R. Storn and K. V. Price, "Differential Evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces", *Technical Report TR-95-012, ICSI*, http://http.icsi.berkeley.edu/~storn/litera.html, 1995.

2. R. Storn and K. V. Price, “Minimizing the real functions of the ICEC 1996 contest by differential evolution”, *Proceedings of the 1996 IEEE international conference on evolutionary computation*, Nagoya, Japan, pp. 842–844. IEEE Press, New York, 1996.
3. R. Storn, “On the usage of differential evolution for function optimization”, In: M. H. Smith, M. A. Lee, J. Keller, and J. Yesn (Eds) *Proc. of the North American Fuzzy Information Processing Society*, IEEE Press, New York, pp. 519–523, 1996.
4. K. V. Price, **“**Differential evolution vs. the functions of the 2nd ICEO”, *IEEE International Conference on Evolutionary Computation*, Page(s):153 – 157, 13-16 Apr 1997.
5. K. V. Price and R. Storn, “Differential evolution: a simple evolution strategy for fast optimization”, *Dr. Dobb’s Journal,* Vol. 22, 18–24, 1997.
6. R. Storn and K. Price, “Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces”, *Journal of Global Optimization*, 11(4) 341–359, 1997.
7. J.A. Nelder and *R.* Mead, "A simplex method for function minimization," *Computer Journal*, Vol 7, pp 308-313, 1965.
8. W. L. Price, “Global optimization by controlled random search”, *Computer Journal,* 20(4): 367-370, 1977.
9. J. Lampinen, “A bibliography of differential evolution algorithm,” *Technical Report*. Lappeenranta University of Technology, Department of Information Technology, Laboratory of Information Processing, 1999. Available via Internet http://www.lut.fi/~jlampine/debiblio.htm
10. A.E. Eiben and J.E. Smith, *Introduction to Evolutionary Computing*, Springer, 2003.
11. K. Price, R. Storn, and J. Lampinen, *Differential Evolution - A Practical Approach to Global Optimization*, Springer, Berlin, 2005.
12. K. V. Price, “An introduction to differential evolution”, in D. Corne, M. Dorigo, and V. Glover (Eds.) *New Ideas in Optimization*, pp. 79–108. Mc Graw-Hill, UK, 1999.
13. K.V. Price and J.I. Rönkkönen, “Comparing the uni-modal scaling performance of global and local selection in a mutation-only differential evolution algorithm”, in: *IEEE Congress on Evolutionary Computation (CEC 2006)*, pp. 2034–2041, July 16-21, 2006.
14. D. Wolpert and W. G. Macready, “No free lunch theorems for optimization”, *IEEE Transactions on Evolutionary Computation*, 1(1), pp. 67-82, April, 1997.
15. E. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello, “A comparative study of differential evolution variants for global optimization,” in *Genetic and Evolutionary Computation Conference (GECCO 2006)*, pp. 485–492, 2006.
16. R. Gamperle, S. D. Muller, and A. Koumoutsakos, “Parameter study for differential evolution”, *WSEAS NNA-FSFS-EC 2002*, Interlaken, Switzerland, Feb. 11-15, 2002.
17. J. Ronkkonen, S. Kukkonen, and K. V. Price, “Real parameter optimization with differential evolution*”, In The 2005 IEEE Congress on Evolutionary Computation (CEC2005)*, vol. 1, pages 506 – 513. IEEE Press, 2005.
18. J. Liu and J. Lampinen, “On setting the control parameters of the differential evolution method”,. in: R. Matoušek and P. Ošmera, (eds.) *Proc. of Mendel* 2002, 8-th *International Conference on Soft Computing*, pp. 11–18, 2002.
19. J. Liu and J. Lampinen, “Adaptive parameter control of differential evolution”, : R. Matoušek and P. Ošmera, (eds.) *Proc. of Mendel* 2002, 8-th *International Conference on Soft Computing*, pp. 19–26, 2002.
20. J. Liu and J. Lampinen, “A Fuzzy adaptive differential evolution algorithm”*, Soft computing- A Fusion of Foundations*, *Methodologies and Applications*, Vol. 9, No. 6, pp. 448-462, 2005.
21. J. Rönkkönen and J. Lampinen, “On using normally distributed mutation step length for the differential evolution algorithm”, 9th *Int. Conf. Soft Computing* (*MENDEL* 2003), Brno, Czech Republic, June 5-7, 2002, pp. 11–18, 2003.
22. A. K. Qin, V. L. Huang, and P. N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization", *IEEE Transactions on Evolutionary Computation,* Vol. 13, Issue 2,, pp. 398-417, April, 2009.
23. M. M. Ali and A. Törn, “Population set based global optimization algorithms: some modifications and numerical studies,” *Computers and Operations Research, Elsevier*, no. 31, pp. 1703–1725, 2004.
24. J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer, “Self-adapting Control parameters in differential evolution: a comparative study on numerical benchmark problems,” *IEEE Transactions on Evolutionary Computation*, Vol. 10, Issue 6, pp. 646 – 657, 2006.

25. D. Zaharie, “Control of population diversity and adaptation in differential evolution algorithms”, In D. Matousek, P. Osmera (eds.), *Proc. of MENDEL* 2003,*In 9th International Conference on Soft Computing, Brno*, Czech Republic, pp. 41-46, June 2003.
26. D. Zaharie and D. Petcu, “Adaptive Pareto differential evolution and its parallelization”, *Proc. of 5th International Conference on Parallel Processing and Applied Mathematics*, Czestochowa, Poland, Sept. 2003.
27. H. Abbass, “The self-adaptive Pareto differential evolution algorithm”, *In: Proceedings of the 2002 Congress on Evolutionary Computation*, Vol. 1, pp. 831-836, Piscataway, New Jersey, My 2002.
28. D. Zaharie, “Critical Values for the Control Parameters of Differential Evolution Algorithms”, In R. Matousek, P. Osmera (Eds.), *Proc. of Mendel 2002, 8th International Conference on Soft Computing*, Brno, Czech Republic, pp. 62-67, 2002.
29. M. Omran, A. Salman, and A. P. Engelbrecht, “Self-adaptive differential evolution”, *Computational Intelligence and Security* (*CIS* 2005), *Proceedings Lecture Notes In Artificial Intelligence*(*LNAI*) Vol. 3801, pp. 192-199, 2005.
30. J. Teo, “Exploring dynamic self-adaptive populations in differential evolution,” *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 2006.
31. R. Mallipeddi, P. N. Suganthan, “Empirical study on the effect of population size on Differential evolution Algorithm”, *IEEE Congress on Evolutionary Computation* (*CEC* 2008), Page(s): 3663-3670, Hong Kong, 1-6 June, 2008.
32. P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, “Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization,” *Technical Report, Nanyang Technological University*, Singapore, May 2005 and *KanGAL Report* #2005005, IIT Kanpur, India.
33. J. Brest and M. Sepesy Maučec, “Population size reduction for the differential evolution algorithm”, *Applied Intelligence*, Vol. 29, No. 3, Page(s) 228-247, Dec. 2008.
34. R. Mallipeddi and P. N. Suganthan, “Differential evolution algorithm with ensemble of populations for global numerical optimization”, *OPSEARCH*, Springer India, vol. 46, no. 2, pp. 184-213, June 2009.
35. S. Das, A. Konar, and U. K. Chakraborty, “Two improved differential evolution schemes for faster global search”, *ACM-SIGEVO Proceedings of GECCO*, Washington D.C., pp. 991-998, June 2005.
36. L.S. Coelho and V.C. Mariani, Combining of chaotic differential evolution and quadratic programming for economic dispatch optimization with valve-point effect, *IEEE Transactions on Power Systems*, Vol. 21, Issue 2, pp. 989–996, 2006.
37. L. S. Coelho, “Reliability–redundancy optimization by means of a chaotic differential evolution approach”, *Chaos, Solitons & Fractals*, Vol. 41, Issue 2, Page(s) 594-602, July 2009.
38. K. T. Alligood, *Chaos: an introduction to dynamical systems*. Springer-Verlag New York, LLC, 1997.
39. H.-Y. Fan and J. Lampinen, “A trigonometric mutation operation to differential evolution”, *International Journal of Global Optimization, 27(1),* 105-129, 2003.
40. H. R. Tizhoosh, “Opposition-Based Learning: A New Scheme for Machine Intelligence”, *Int. Conf. on Computational Intelligence for Modeling Control and Automation - CIMCA’2005*, Vol. I, pp. 695-701, Vienna, Austria, 2005.
41. H. R. Tizhoosh, “Reinforcement learning based on actions and opposite actions*”. ICGST International Conference on Artificial Intelligence and Machine Learning (AIML-05),* Cairo, Egypt, 2005.
42. H.R. Tizhoosh, “Opposition-based reinforcement learning”, *Journal of Advanced Computational Intelligence and Intelligent Informatics*, Vol. 10, No. 3, 2006.
43. S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama, “Opposition-based differential evolution”, *IEEE Transactions on Evolutionary Computation*, Volume 12, Issue 1, pp. 64-79, Feb. 2008.
44. S. Rahnamayan, H.R. Tizhoosh, and M. M. A. Salama, “Opposition-based differential evolution for optimization of noisy problems”, *Proc. 2006 IEEE Congress on Evolutionary Computation (CEC-2006), Vancouver,* July 2006, pp. 1865-1872.
45. J. Zhang, and A. C. Sanderson, “JADE: adaptive differential evolution with optional external archive”, *IEEE Transactions on Evolutionary Computation*, Vol. 13, Issue 5, Page(s): 945-958, Oct. 2009.

46. S. Das, A. Abraham, U. K. Chakraborty, and A. Konar, "Differential evolution using a neighborhood based mutation operator", *IEEE Transactions on Evolutionary Computation*, Vol. 13, Issue 3, Page(s): 526-553, June, 2009.
47. T. Hendtlass, "A combined swarm differential evolution algorithm for optimization problems," *Lecture Notes in Computer Science*, vol.2070, pp.11-18, 2001.
48. W.-J. Zhang and X.-F. Xie, "DEPSO: Hybrid particle swarm with differential evolution operator," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 2003, pp. 3816–3821.
49. S. Das, A. Konar, and U. K. Chakraborty, "Improving particle swarm optimization with differentially perturbed velocity," in *ACM-SIGEVO Proceedings of Genetic Evol. Comput. Conf. (GECCO)*, Washington DC, Jun. 2005, pp. 177–184.
50. P. W. Moore and G. K. Venayagamoorthy, "Evolving digital circuit using hybrid particle swarm optimization and differential evolution," *Int.J. Neural Syst.*, vol.16, no.3, pp.163-177, 2006.
51. B. Liu, L. Wang, Y. H. Jin, and D. X. Huang, "Designing neural networks using hybrid particle swarm optimization," *Lecture Notes in Computer Science*, vol.3469, pp.391-397, 2005.
52. S. Kannan, S. M. R. Slochanal, P. Subbaraj, and N. P. Padhy, "Application of particle swarm optimization technique and its variants to generation expansion planning," *Electric Power Syst. Res.*, vol.70, no.3, pp.203-210, 2004.
53. Z. F. Hao, G. H. Guo, and H. Huang, "A particle swarm optimization algorithm with differential evolution," in *Proc.* 6th *Int. Conf. Mach.Learn. Cybern.*, 2007, vol.2, pp.1031-1035.
54. M. G. H. Omran, A. P. Engelbrecht, and A. Salman, "Bare bones differential evolution", in *European Journal of Operational Research* 196, Page(s) 128–139, 2009.
55. J. Kennedy, "Bare bones particle swarms," in *Proc. IEEE Swarm Intell. Symp.*(SIS2003), pp.80-87, 2003.
56. X. Xu, Y. Li, S. Fang, Y. Wu, and F. Wang, "A novel differential evolution scheme combined with particle swarm intelligence," *IEEE Congress in Evolutionary Computation.* (CEC08), pp.1057-1062, 2008.
57. S. Das, A. Konar, and U. K. Chakraborty, "Annealed Differential Evolution", *IEEE Congress in Evolutionary Computation, CEC 2007*, IEEE press, USA, ISBN 1-4244-1340-0, 2007.
58. S. Kirkpatrik, C. Gelatt, and M. Vecchi, "Optimization by Simulated Annealing", *Science*, 220: 671–680, 1983.
59. A. Biswas, S. Dasgupta, S. Das, and A. Abraham, "A Synergy of differential evolution and bacterial foraging algorithm for global optimization", *Neural Network World*, Volume 17, No. 6, pp. 607-626, 2007.
60. N. Noman and H. Iba, "Accelerating Differential Evolution Using an Adaptive Local Search", *IEEE Transactions on Evolutionary Computation*, Vol. 12, Issue 1, pp. 107 – 125, 2008.
61. S. Tsutsui, M. Yamamura, and T. Higuchi, "Multi-parent recombination with simplex crossover in real coded genetic algorithms," in *Proc.Genetic Evol. Comput. Conf. (GECCO'99)*, pp. 657–664, Jul. 1999.
62. Y.-S. Ong, and A. J. Keane, "Meta-lamarckian learning in memetic algorithms," *IEEE Transactions on Evolutionary Computation*, Vol. 8, No. 2, pp. 99–110, 2004.
63. Z. Yang, J. He, and X. Yao, *Making a Difference to Differential Evolution*, in *Advances in Metaheuristics for Hard Optimization*, Z. Michalewicz and P. Siarry (eds.), pp 415-432, Springer, 2007.
64. Z. Michalewicz and D. B. Fogel, *How to Solve It: Modern Heuristics*, Springer, Berlin, 1999.
65. Z. Yang, K. Tang and X. Yao, "Large Scale Evolutionary Optimization Using Cooperative Coevolution," *Information Sciences*, Vol. 178, Issue 15, pp. 2985-2999, 2008.
66. Z. Yang, K. Tang and X. Yao, "Self-adaptive differential evolution with neighborhood search", in *Proc. IEEE Congress on Evolutionary Computation* (CEC-2008), Hong Kong, 1110-1116, 2008.
67. R. Mendes and J. Kennedy, "The fully informed particle swarm: simpler, maybe better," *IEEE Transactions of Evolutionary Computation*, Vol. 8, No. 3, 2004.
68. F. Neri and V. Tirronen, "Scale factor local search in differential evolution," *Memetic Computing Journal*, Vol. 1, No. 2 , Page(s) 153-171 June, 2009.
69. V. Tirronen, F. Neri, T. Kärkkäinen, K. Majava, and T. Rossi, "An enhanced memetic differential evolution, in filter design for defect detection in paper production," *Evolutionary Computation*, vol. 16, pp. 529–555, Dec. 2008.
70. A. Caponio, F. Neri, and V. Tirronen, "Super-fit control adaptation in memetic differential evolution frameworks," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 13, no. 8, pp. 811–831, 2009.

71. J. Lampinen and I Zelinka, "Mixed integer-discrete-continuous optimization with differential evolution", 5-th International Mendel Conference on Soft Computing, *MENDEL*'99, Brno, Czech Republic, 9 – 12 June, 1999.
72. M. F. Tasgetiren, Q. K. Pan P. N. Suganthan and Y. C. Liang, "A Discrete Differential Evolution Algorithm for the No-Wait Flowshop Scheduling Problem with Total Flow time Criterion", *P. IEEE Symposium on Computational Intelligence in Scheduling*, Hawaii, pp. 251-258, 1st-5th April 2007.
73. M. F.Tasgetiren, Q. Pan, and Y. Liang, "Discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times", *Computers & Operations Research*, Vol. 36, Issue 6, Pages 1900-1915, June 2009.
74. G. Pampara, A.P. Engelbrecht, and N. Franken, "Binary differential evolution", *IEEE Congress on Evolutionary Computation* (CEC), 2006, Page(s): 1873-1879, Vancouver, BC, July 16 – 21, 2006.
75. D. Zaharie, "On the explorative power of differential evolution", 3rd *International Workshop on Symbolic and Numerical Algorithms on Scientific Computing*, SYNASC-2001, Timişoara, Romania, October 2 – 4, 2001
76. D. Zaharie, "Parameter Adaptation in Differential Evolution by Controlling the Population Diversity", in D.Petcu *et al*. (eds), *Proc. of 4th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing*, Timisoara, Romania (2002) 385–397.
77. H.-G. Beyer, "On the dynamics of EAs without selection", *Proceedings of Foundations of Genetic Algorithms 5 (FOGA-5),* Pages 5–26, 1999.
78. S. Dasgupta, S. Das, A. Biswas, A. Abraham, "The population dynamics of Differential Evolution: A mathematical model", *IEEE Congress on Evolutionary Computation* (*CEC* 2008), Page(s): 1439 - 1446, Hong Kong, 1-6 June, 2008.
79. S. Dasgupta, S. Das, A. Biswas, A. Abraham, "On stability and convergence of the population-dynamics in differential evolution", *AI Communications,* Vol. 22, Issue 1*,* Page(s) 1-20, 2009.
80. J. Vesterstrøm and R. A. Thomson, "Comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems", in *Proc. Sixth Congress on Evolutionary Computation* (*CEC*-2004), IEEE Press, 2004.
81. D. Zaharie, "Statistical properties of differential evolution and related random search algorithms", in P. Brito (Ed.) *Proceedings of International Conference on Computational Statistics,* Porto, Portugal, August 24-29, Physica-Verlag HD, Page(s) 473-485, 2008.
82. D. Zaharie, "Influence of crossover on the behavior of differential evolution algorithms", *Applied Soft Computing*, Vol. 9, Issue 3, Page(s) 1126 – 1138, June 2009.
83.. K. Zielinski, D. Peters, and R. Laur, "Run time analysis regarding stopping criteria for differential evolution and particle swarm optimization," in *Proc. of the 1st International Conference on Experiments/Process/System Modelling/Simulation/Optimization*, Athens, Greece, 2005.
84. A. Horner, J. Beauchamp, and L. Haken, "Genetic algorithms and their application to FM matching synthesis," *Comput. Music J*., vol. 17, pp. 17-29, 1993.
85. F. Herrera and M. Lozano, "Gradual distributed real-coded genetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 1, pp. 43–62, (2000).