8

Differential Evolution Via Exploiting Opposite Populations

Shahryar Rahnamayan[1] and H.R. Tizhoosh[2]

[1] Faculty of Engineering and Applied Science, University of Ontario Institute of Technology (UOIT), Canada
Shahryar.Rahnamayan@uoit.ca
[2] Department of Systems Design Engineering, University of Waterloo, Canada
tizhoosh@uwaterloo.ca

Summary. The concept of opposition can contribute to improve the performance of population-based algorithms. This chapter presents an overview of a novel opposition-based scheme to accelerate an evolutionary algorithm, differential evolution (DE). The proposed opposition-based DE (ODE) employs opposition-based computation (OBC) for population initialization and also for generation jumping. Opposite numbers, representing anti-chromosomes, have been utilized to improve the convergence rate of the classical DE. A test suite with 15 well-known benchmark functions is employed for experimental verification. Descriptions for the DE and ODE algorithms, and a comparison strategy are provided. Results are promising and confirm that the ODE outperforms its parent algorithm DE. This work can be regarded as an initial study to exploit oppositional concepts to expedite the optimization process for any population-based approach.

8.1 Introduction

Evolutionary algorithms (EAs) are well-established techniques to approach problems with mixed-type variables, many local optima, and with undifferentiable or non-analytical functions [1]. Among various kinds of evolutionary algorithms, differential evolution (DE) is well known for its effectiveness and robustness. Many comparative studies confirm that the DE outperforms many other optimizers [5]. Finding more accurate solution(s) in a shorter period of time for complex black-box problems is still a crucial target of research on evolutionary algorithms.

In this chapter, opposition-based schemes including opposition-based population initialization and generation jumping, will be described. The differential evolution (DE) is selected as a parent algorithm to verify the acceleration effect of the proposed schemes. A set of well-known complex benchmark functions is employed to experimentally compare and analyze the algorithms. Results confirm that Opposition-Based Differential Evolution (ODE) performs better than DE in terms of convergence speed and solution accuracy.

The main purpose of this and previous works has been to introduce a new notion into nonlinear continuous optimization via innovative metaheuristics, namely *the notion of opposition*. Although, all conducted experiments utilize DE as a parent algorithm, the proposed schemes are defined at the population level and, hence, have an inherent potential to be utilized for acceleration of other population-based algorithms.

H.R. Tizhoosh, M. Ventresca (Eds.): Oppos. Concepts in Comp. Intel., SCI 155, pp. 143–160, 2008.
springerlink.com

The organization of this chapter is as follows: A short review of differential evolution is given in section 8.2. The main reasons to select DE as a parent algorithm are explained in section 8.3. Opposition-based differential evolution is described in section 8.4. Experimental verifications are elaborated in section 8.5. Finally, the chapter is concluded in section 8.6.

8.2 Differential Evolution (DE)

Differential evolution (DE) is a population-based optimization algorithm based on the idea of genetic annealing which was used to solve the Chebyshev polynomial fitting problem [1]. In order to solve the Chebyshev problem in continuous space, a modified genetic annealing algorithm from bit-string to floating-point encoding and a consequent switch from logical operators to arithmetic ones were proposed [2, 3, 4]. During these experiments, the differential mutation operator to perturb the population of vectors was discovered. Additionally, by using differential mutation, discrete recombination, and pair-wise selection, it was recognized that an annealing mechanism is not needed; it was removed completely and DE was born.

Let us assume that $X_{i,G}(i = 1, 2, ..., N_p)$ are candidate solution vectors in generation G (N_p : population size). Like other evolutionary algorithms, DE starts with an initial population, which is usually generated in a random manner. A typical vector of the initial population can be generated as follows [5]:

$$X_{i,j} = l_j + \text{RAND}_j(0,1) \times (l_j - u_j) \quad \text{with} \quad j = 1, 2, ..., D, \tag{8.1}$$

where D is the problem dimensionality; l_j and u_j are the lower and the upper boundaries of the j^{th} variable, respectively, and $\text{RAND}(0,1)$ is a uniformly generated random number in $[0,1]$.

Successive populations are generated by adding the weighted difference of two randomly selected vectors to a third randomly selected vector. For classical DE (see Algorithm 1), the mutation, crossover, and selection operators are straightforwardly defined.

8.2.1 Mutation

For each vector $X_{i,G}$ in generation G a mutant vector $V_{i,G}$ (see line 9 of Algorithm 1) is defined by

$$V_{i,G} = X_{a,G} + F(X_{c,G} - X_{b,G}), \tag{8.2}$$

where $i = \{1, 2, ..., N_p\}$, and a, b, and c are mutually different random integer indices selected from $\{1, 2, ..., N_p\}$. Further, the variables i, a, b, and c are different so that $N_p \geq 4$ is necessary. The factor $F \in [0, 2]$ is a real constant which determines the amplification of the added differential variation of $(X_{c,G} - X_{b,G})$ [5]. Larger values for F result in higher diversity in the generated population and lower values cause faster convergence.

8.2.2 Crossover

DE utilizes the crossover operation to generate new solutions by shuffling competing vectors and also to increase the population diversity. For the classical DE (lines $10-16$ of Algorithm 1), the binary crossover is utilized. It defines the following trial vector:

$$U_{i,G} = (U_{1i,G}, U_{2i,G}, ..., U_{Di,G}), \tag{8.3}$$

where

$$U_{ji,G} = \begin{cases} V_{ji,G} & \text{if RAND}_j(0,1) \leq C_r \vee j = k, \\ X_{ji,G} & \text{otherwise.} \end{cases} \tag{8.4}$$

$C_r \in (0,1)$ is the predefined crossover rate, and $\text{RAND}_j(0,1)$ is the j^{th} evaluation of a uniform random number generator. The parameter $k \in \{1,2,...,D\}$ is a random index chosen once for each i to make sure that at least one parameter is always selected from the mutated vector $V_{ji,G}$. The most common values for C_r are in the range of $(0.4,1)$ [17]. Figure 8.1 illustrates a pictorial example for the binary crossover.

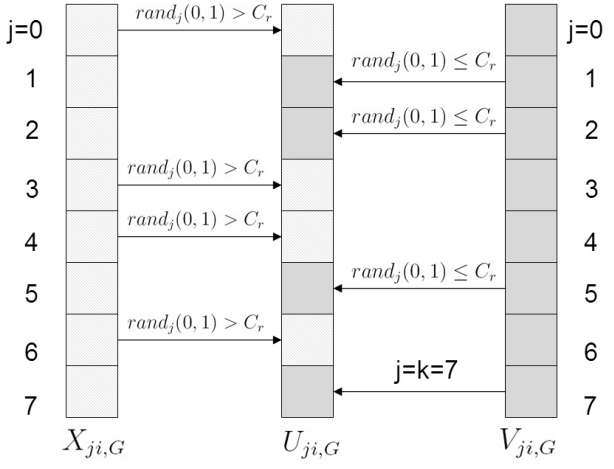
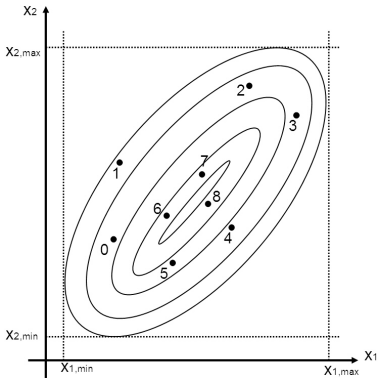


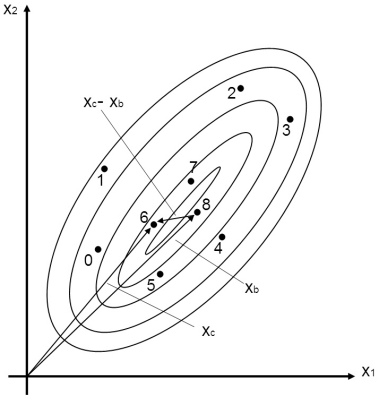
Fig. 8.1. A pictorial example for the binary crossover in DE ($k = 7$) [5]. When $\text{RAND}_j(0,1) \leq C_r \vee j = k$, then the variable is copied from $V_{ji,G}$, otherwise copied from $X_{ji,G}$ to $U_{ji,G}$.

8.2.3 Selection

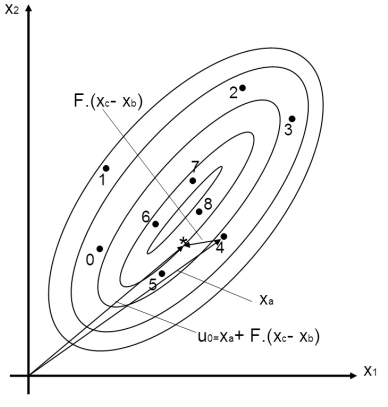
Selection is a mechanism to decide which vector ($U_{i,G}$ or $X_{i,G}$) should be a member of next (new) generation $G+1$. For a minimization problem, the vector with the lower objective function value is chosen (greedy selection). If $f(U_i) \leq f(X_i)$, then U_i is selected; otherwise X_i will be chosen (lines $17-23$ of Algorithm 1).



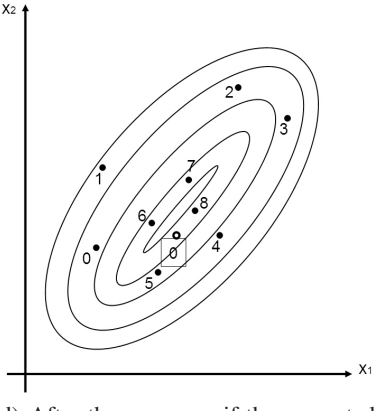
(a) Population initialization for DE ($N_p = 9$). Contour lines for $f(x_1, x_2)$ are shown by ellipses.



(b) Generating difference vector $X_c - X_b$. b and c are the randomly selected indices.



(c) Generating $X_{a,G} + F(X_{c,G} - X_{b,G})$. a is the third randomly selected index.



(d) After the crossover if the generated vector has lower objective value, then it will be replaced with the vector 0.

Fig. 8.2. Illustration of one generate-and-test cycle for DE (starting from vector 0) [5]

This evolutionary cycle (i.e., mutation, crossover, and selection) is repeated N_p times to generate new populations. These successive generations are produced until the termination conditions are satisfied. One generate-and-test cycle for DE is presented in Figure 8.2.

The starting point for the mutation, crossover and selection is indicated by the comments in the algorithm. The algorithm terminates (line 5) when the best achieved fitness value (BFV) is smaller than the value-to-reach (VTR), or the number of function calls (NFC) exceeds the predefined maximum number of function calls (MAX_{NFC}). The termination strategy can be defined differently based on the application or the purpose of the experiment. The number of generations, the execution time, or some population

Algorithm 1. Differential Evolution (DE). P_0: Initial population, N_p: Population size, V: Noise vector, U: Trial vector, D: Problem dimension, BFV: Best achieved fitness value, VTR: Value-to-reach, NFC: Number of function calls, MAX_{NFC}: Maximum number of function calls, F: Mutation constant, $\text{RAND}(0,1)$: Uniformly generated random number, C_r: Crossover, $f(\cdot)$: Objective function, P': Population of the next generation.

Generate uniformly distributed random population P_0
NFC $\leftarrow 0$
Evaluate individuals of P_0
NFC $\leftarrow$ NFC $+ N_p$
while (BFV $>$ VTR **and** NFC $<$ MAX_{NFC}) **do**
  {Generate-and-Test-Loop}
  for $i = 0$ to N_p **do**
    Select three parents X_a, X_b, and X_c randomly from current population where $i \neq a \neq b \neq c$
    {Mutation}
    $V_i \leftarrow X_a + F \times (X_c - X_b)$
    {Crossover}
    for $j = 0$ to D **do**
      if $\text{RAND}(0,1) < C_r$ **then**
        $U_{i,j} \leftarrow V_{i,j}$
      else
        $U_{i,j} \leftarrow X_{i,j}$
      end if
    end for
    {Selection}
    Evaluate U_i
    NFC $\leftarrow$ NFC $+ 1$
    if $(f(U_i) \leq f(X_i))$ **then**
      $X'_i \leftarrow U_i$
    else
      $X'_i \leftarrow X_i$
    end if
  end for
  $X \leftarrow X'$
end while

statistics (e.g., diversity or the improvement rate) are some commonly used termination criteria.

8.2.4 DE in Optimization Field

A summary classification of optimization methods can be seen in Figure 8.3. According to the proposed classification scheme for optimization methods, DE is a population-based, nonlinear, continuous and global optimization algorithm [1].

Studies have been conducted to enhance the performance of the classical DE algorithm by adaptive determination of DE control parameters. For instance, the fuzzy adaptive differential evolution algorithm (FADE) was introduced by Liu and Lampinen

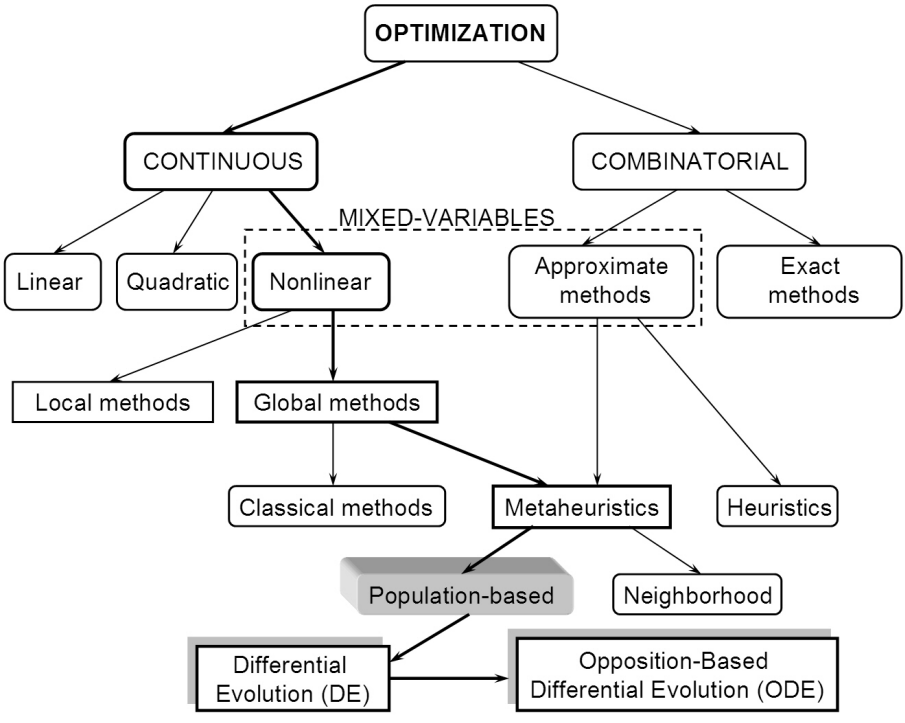


Fig. 8.3. A simple classification scheme of optimization methods [1]

[24]. They employed a fuzzy logic controller to set the mutation and crossover rates. In the same direction, Brest et al. [20] proposed self-adaptive DE. Teo [30] proposed a dynamic population sizing strategy based on self-adaptation, and Ali and Törn [25] introduced auxiliary population and automatic calculating of the amplification factor F for the difference vector.

Other researchers have experimented with multi-population ideas. Tasoulis et al. [31] proposed parallel DE where they assign each subpopulation to a different processor node. Shi et al. [32] partitioned high-dimensional search spaces into smaller spaces and used multiple cooperating subpopulations to find the solution. They called this method cooperative co-evolutionary differential evolution.

Hybridization with different algorithms is another direction for improvement of DE. Sun et al. [26] proposed a new hybrid algorithm based on a combination of DE and estimation of distribution algorithm. This technique uses a probability model to determine promising regions in order to focus the search process on those areas. Noman and Iba [35] incorporated local search into the classical DE. They employed fittest-individual refinement which is a crossover-based local search. Fan and Lampinen [33] introduced a new local search operation, trigonometric mutation, in order to obtain a better trade-off between convergence speed and robustness. Kaelo and Ali [34] employed reinforcement learning and some other schemes for generating fitter trial points.

All mentioned approaches are proposed to increase the convergence rate and/or the accuracy of DE. For more details about DE extensions, the reader is referred to literature [1, 5, 27].

8.3 Why Differential Evolution?

Differential evolution is a simple and compact metaheuristic which directly operates on continuous variables (arithmetic operators instead of logical operators). Unlike many binary versions of the genetic algorithms, DE works with the floating-point numbers. This removes encoding and decoding of the variables which is a source of inaccuracy. Consequently, this feature makes DE scalable for high-dimensional problems and also time and memory efficient.

Another reason for choosing DE is that it does not need a probability density function to adapt the control parameters (unlike most evolutionary strategies) or any probability distribution pattern for the mutation (unlike genetic algorithms or evolutionary programming). DE's different mutation and crossover schemes distinguish it from other evolutionary algorithms [5].

Additionally, handling mixed integers, discrete and continuous variables makes DE more applicable for a wider range of real-world applications. The main advantage of DE while working with integer variables is that it internally works on a continuous space and only switches to the integer space during the evaluation of the objective function. This characteristic supports higher accuracy compared to some other well-known algorithms (e.g., GAs) which perform in the reverse manner [27]. Extensions of classical DE are capable of handling boundary constraints and also nonlinear function constraints which both are commonly required in the real-world problems [5].

Many comparative studies report higher robustness, convergence speed, and solution quality of the DE when compared to other evolutionary algorithms for both benchmark functions and real-world applications. A comprehensive performance study is provided in [5]. The authors first compare DE to 16 other optimizers against five well-known thirty-dimensional test functions (namely, Rosenbrock, Ackley, Griewangk, Rastrigin, and Schwefel). Consequently, they explored eight function-based comparative studies (e.g., unconstrained optimization, multi-constraints nonlinear optimization, and multi-objective optimization) and also eleven application-oriented performance comparison studies (e.g., multi-sensor fusion, earthquake relocation, image registration, and optimization of neural networks). Finally, they conclude [5] "[...] DE may not always be the fastest method, it is usually the one that produces the best results, although the number of cases in which it is also the faster is significant. DE also proves itself to be robust, both in how control parameters are chosen and in the regularity with which it finds the true optimum. [...] As these researchers have found, DE is a good first choice when approaching a new and difficult global optimization problem is defined with continuous and/or discrete parameters."

8.4 Opposition-Based Differential Evolution

In his primary paper on opposition-based learning (OBL), Tizhoosh proposed to use *anti-chromosmes* for GAs [12]. For every selected chromosome a corresponding

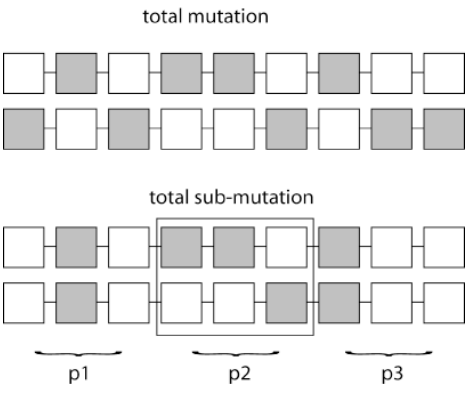


Fig. 8.4. Generation of anti-chromosomes [12]

anti-chromosome can be generated. The initial chromosomes are generally generated randomly meaning that they can possess high or low fitness. However, in a complex problem it is usually very likely that the initial populations do not contain optimal solutions. Hence, in lack of any a-priori knowledge, it is reasonable to look at anti-chromosomes simultaneously. Considering the search direction and its opposite at the same time will bear more likelihood to reach the best population in a shorter time (for more motivation on OBL see also Chapters 1-2). Tizhoosh also suggested to use total or sub-total-mutation to generate opposite candidate solutions (Figure 8.4).

Similar to all population-based optimization algorithms, two main steps are distinguishable for DE, namely population initialization and producing new generations by evolutionary operations such as mutation, crossover, and selection. The opposition-based differential evolution (ODE) [6, 9, 13] will enhance these two steps by considering opposite solutions. For black-box optimization – which is a general assumption for optimization methods – there is no information about the shape of the problem landscape such that type II opposition can only be approximated via type I opposition (see Definitions 2 and 6 in Chapter 2). The pseudo-code of ODE is presented in Algorithm 2 [13].

8.4.1 Opposition-Based Population Initialization

In absence of domain knowledge, uniform random number generation is generally the only choice to create an initial population. But as mentioned before, by utilizing type-I opposition it is possible to obtain fitter starting candidates. Block (1) from Figure 8.5 shows the implementation of opposition-based population initialization (lines $5-12$ of Algorithm 2). The following steps explain that procedure [6]:

Step 1. Initialize the population $\mathrm{P}(N_p)$ randomly,
Step 2. Create the opposite population OP by

$$\mathrm{OP}_{i,j} = \mathrm{a}_j + \mathrm{b}_j - \mathrm{P}_{i,j}, \quad \text{with} \quad i = 1, 2, ..., N_p; j = 1, 2, ..., D, \tag{8.5}$$

Algorithm 2. Pseudo-code of Opposition-Based Differential Evolution (ODE) in order to solve a minimization problem [Adopted from [13]]. P_0: Initial population, OP_0: Opposite of initial population, P: Current population, OP: Opposite of current population, D: Problem dimension, $[l_j, u_j]$: Range of the j-th variable, J_r: Jumping rate, C_r: Crossover rate, $\min_j^p$/$\max_j^p$: Minimum/maximum value of the j-th variable in the current population. Lines **1-12** and **33-42** are implementations of opposition-based population initialization and opposition-based generation jumping, respectively.

1: Generate uniformly distributed random population P_0
2: NFC $\leftarrow 0$
3: Evaluate individuals of P_0
4: NFC $\leftarrow$ NFC $+ N_p$
{**Begin of Opposition-Based Population Initialization**}
5: **for** $i = 0$ to N_p **do**
6: **for** $j = 0$ to D **do**
7: $OP_{0i,j} \leftarrow l_j + u_j - P_{0i,j}$
8: **end for**
9: **end for**
10: Evaluate individuals of OP_0
11: NFC $\leftarrow$ NFC $+ N_p$
12: Select N_p fittest (best) individuals from P_0 and OP_0 as initial population P_0
{Begin of DE's Evolution Steps}
13: **while** (BFV $>$ VTR **and** NFC $<$ MAX_{NFC}) **do**
14: **for** $i = 0$ to N_p **do**
15: Select three parents P_{i_1}, P_{i_2}, and P_{i_3} randomly from current population where $i \neq i_1 \neq i_2 \neq i_3$
16: $V_i \leftarrow P_{i_1} + F \times (P_{i_2} - P_{i_3})$
17: **for** $j = 0$ to D **do**
18: **if** RAND$(0, 1) < C_r$ **then**
19: $U_{i,j} \leftarrow V_{i,j}$
20: **else**
21: $U_{i,j} \leftarrow P_{i,j}$
22: **end if**
23: **end for**
24: Evaluate U_i
25: NFC $\leftarrow$ NFC $+ 1$
26: **if** $(f(U_i) \leq f(P_i))$ **then**
27: $P'_i \leftarrow U_i$
28: **else**
29: $P'_i \leftarrow P_i$
30: **end if**
31: **end for**
32: $P \leftarrow P'$
{**Begin of Opposition-Based Generation Jumping**}
33: **if** RAND$(0, 1) < J_r$ **then**
34: **for** $i = 0$ to N_p **do**
35: **for** $j = 0$ to D **do**
36: $OP_{i,j} \leftarrow \text{MIN}_j^p + \text{MAX}_j^p - P_{i,j}$
37: **end for**
38: **end for**
39: Evaluate individuals of OP_0
40: NFC $\leftarrow$ NFC $+ N_p$
41: Select N_p fittest (best) individuals from P and OP as current population P
42: **end if**
43: **end while**

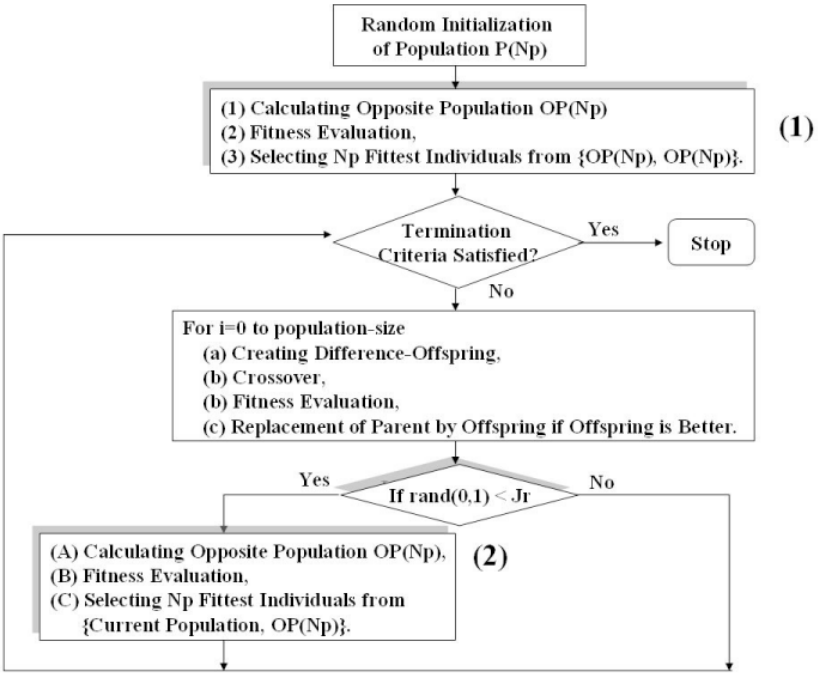


Fig. 8.5. Gray boxes extend DE to ODE. Block (1): Opposition-based initialization, Block (2): Opposition-based generation jumping (J_r: jumping rate, RAND$(0, 1)$: uniformly generated random number, N_p: population size).

where P_i and OP_i denote the i^{th} individual of the current population and its corresponding opposite, respectively, and $[l_j, u_j]$ is the range of the j^{th} variable.

Step 3. Select the N_p fittest (best) individuals from P $\cup$ OP as the initial population.

According to the above procedure, $2N_p$ function evaluations are required instead of N_p for the regular random population initialization. But, by the opposition-based initialization, the parent algorithm can start with the fitter initial individuals instead.

8.4.2 Opposition-Based Generation Jumping

By applying a similar approach mentioned in Sec. 8.4.1 to the current population, which means selecting N_p best individuals from the current and corresponding opposite populations, the evolutionary process can be forced to jump to a fitter generation (the generation with fitter individuals). After generating new populations, the opposite population is calculated and the N_p fittest (best) individuals are selected from the union of the current and opposite population based on a jumping rate $J_r \in (0, 0.4)$ [13, 15]. In order to calculate the opposite population for generation jumping, the opposite of each variable is calculated dynamically; that is, the maximum and minimum values of each variable

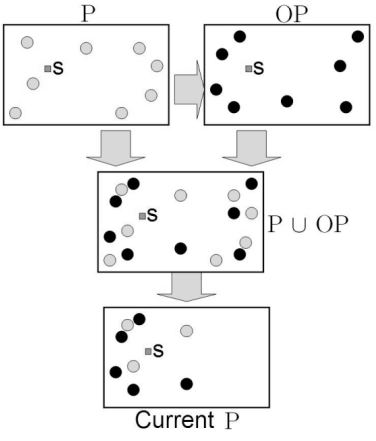


Fig. 8.6. Example to visualize the opposition-based generation jumping in 2D space ($N_p = 8$)

in the current population ($[\text{MIN}_j^p, \text{MAX}_j^p]$) are used to calculate opposite points instead of using variables' predefined interval boundaries ($[l_j, u_j]$):

$$\text{OP}_{i,j} = \text{MIN}_j^p + \text{MAX}_j^p - \text{P}_{i,j}, i = 1, 2, ..., N_p; j = 1, 2, ..., D. \tag{8.6}$$

If the opposites are calculated within variables' static boundaries, it is possible to jump outside of the already shrunken search space and lose the knowledge of the already reduced space. Hence, we calculate opposite points by using variables' current interval in the population ($[\text{MIN}_j^p, \text{MAX}_j^p]$) which is, as the search does progress, increasingly smaller than the corresponding initial range $[l_j, u_j]$. Block (2) from Figure 8.5 illustrates the implementation of opposition-based generation jumping (lines $33 - 42$ of Algorithm 2).

A pictorial example for opposition-based generation jumping procedure in 2D space is illustrated in Figure 8.6. The letter 'S' indicates the location of the optimal solution. Dark and light circles represent the points and the opposite points, respectively.

In [14], we established mathematical proofs and experimental evidence to verify the advantage of opposite points compared to additional random points when dealing with high-dimensional problems (see also Chapter 2 for more discussions on the formalism of opposition). Both experimental and mathematical results confirmed that opposite points are more beneficial than additional independent random points. We can conclude that the opposition-based learning can be utilized to accelerate optimization methods since considering the pair x and its opposite $\breve{x}$ has apparently a higher fitness probability than pure randomness.

8.5 Experimental Verifications

In this section, the convergence metrics are defined and DE and ODE are compared experimentally over well-known benchmark functions (section 8.5.1). Also, the

contribution of opposite points to the achieved acceleration rate is investigated by replacing them with random points (section 8.5.2).

8.5.1 Comparison of DE and ODE

A set of 15 well-known benchmark functions [6, 13, 15], which contains 7 unimodal (f_1,f_2,f_3,f_6,f_{10},f_{11},f_{14}) and 8 multimodal functions (f_4,f_5,f_7,f_8,f_9,f_{12},f_{13},f_{15}), has been selected for performance verification of ODE. The definition of the benchmark functions is given in Table 8.1.

Table 8.1. List of employed benchmark functions (unimodal and multimodal)

Function	*Search Space*
$f_1(X) = \sum_{i=1}^{D} {x_i}^2$	$[-5.12, 5.12]^D$
$f_2(X) = \sum_{i=1}^{D} i{x_i}^2$	$[-5.12, 5.12]^D$
$f_3(X) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j \right)^2$	$[-65, 65]^D$
$f_4(X) = 10D + \sum_{i=1}^{D} \left(x_i^2 - 10\cos(2\pi x_i) \right)$	$[-5.12, 5.12]^D$
$f_5(X) = \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600, 600]^D$
$f_6(X) = \sum_{i=1}^{D} \lvert x_i \rvert^{(i+1)}$	$[-1, 1]^D$
$f_7(X) = -20\exp\left(-0.2\sqrt{\frac{\sum_{i=1}^{D} x_i^2}{D}}\right) - \exp\left(\frac{\sum_{i=1}^{D} \cos(2\pi x_i)}{D}\right) + 20 + e$	$[-32, 32]^D$
$f_8(X) = \sin^2(3\pi x_1) + \sum_{i=1}^{D-1} (x_i - 1)^2(1 + \sin^2(3\pi x_{i+1})) + (x_D - 1)(1 + \sin^2(2\pi x_D))$	$[-10, 10]^D$
$f_9(X) = -\sum_{i=1}^{D} \sin(x_i)(sin(ix_i^2/\pi))^{2m}, (m = 10)$	$[0, \pi]^D$
$f_{10}(X) = \sum_{i=1}^{D} x_i^2 + \left(\sum_{i=1}^{D} 0.5ix_i \right)^2 + \left(\sum_{i=1}^{D} 0.5ix_i \right)^4$	$[-5, 10]^D$
$f_{11}(X) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	$[-10, 10]^D$
$f_{12}(X) = \sum_{i=1}^{D} (\lfloor x_i + 0.5 \rfloor)^2$	$[-100, 100]^D$
$f_{13}(X) = \sum_{i=1}^{D} \lvert x_i \sin(x_i) + 0.1x_i \rvert$	$[-10, 10]^D$
$f_{14}(X) = \exp\left(-0.5 \sum_{i=1}^{D} x_i^2\right)$	$[-1, 1]^D$
$f_{15}(X) = 1 - \cos(2\pi \parallel x \parallel) + 0.1 \parallel x \parallel$, where $\parallel x \parallel = \sqrt{\sum_{i=1}^{D} x_i^2}$	$[-100, 100]^D$

We compare the convergence speed of DE and ODE by measuring the number of function calls (NFC) which is the most commonly used metric in literature [5, 7, 8, 9, 10, 11, 19]; a smaller NFC means higher convergence speed. The termination criterion is to find a value smaller than the value-to-reach (VTR) before reaching the maximum number of function calls MAX_{NFC}. In order to minimize the effect of the stochastic nature of the algorithms on the metric, the reported number of function calls (NFC) for

Table 8.2. Parameter settings for conducted experiments

Parameter name	*Setting*	*Reference*
population size (N_p)	100	[20, 21, 22]
differential amplification factor (F)	0.5	[11, 20, 23, 24, 25]
crossover probability constant (C_r)	0.9	[11, 20, 23, 24, 25]
jumping rate constant (J_r)	0.3	[9, 13, 18]
maximum number of function calls (MAX_{NFC})	10^6	[9, 13, 18]
value to reach (VTR)	10^{-8}	[10]
mutation strategy	$DE/rand/1/bin$	[5, 20, 23, 26, 27]

Table 8.3. Comparison of DE, ODE, and RDE. The best result for each case is highlighted in boldface. Results for RDE have been discussed in section 8.5.2 (corresponding results for replacing the opposite points with random points).

		DE			*ODE*			*RDE*		
F	*D*	*NFC*	*SR*	*SP*	*NFC*	*SR*	*SP*	*NFC*	*SR*	*SP*
f_1	30	87748	1	87748	47716	1	**47716**	115096	1	115096
f_2	30	96488	1	96488	53304	1	**53304**	126780	1	126780
f_3	20	177880	1	177880	168680	1	**168680**	231152	1	231152
f_4	10	328844	1	328844	70389	0.76	**92617**	501875	0.96	522786
f_5	30	113428	1	113428	69342	0.96	**72231**	149744	1	149744
f_6	30	25140	1	25140	8328	1	**8328**	29096	1	29096
f_7	30	169152	1	169152	98296	1	**98296**	222784	1	222784
f_8	30	101460	1	101460	70408	1	**70408**	138308	1	138308
f_9	10	191340	0.76	**251763**	213330	0.56	380946	306900	0.60	511500
f_{10}	30	385192	1	385192	369104	1	**369104**	498200	1	498200
f_{11}	30	187300	1	187300	155636	1	**155636**	244396	1	244396
f_{12}	30	41588	1	41588	23124	1	**23124**	54316	1	54316
f_{13}	30	411164	1	411164	337532	1	**337532**	927230	0.24	3863458
f_{14}	10	19528	1	19528	15704	1	**15704**	23156	1	23156
f_{15}	10	37824	1	37824	24260	1	**24260**	46800	1	46800
	SR_{ave}		0.98			0.95			0.92	

each function is the average over 50 different trials. The number of times, for which the algorithm successfully reaches the VTR for each test function is measured as the success rate SR:

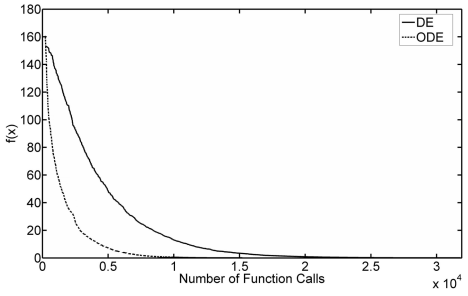
$$\text{SR} = \frac{\text{number of times reached VTR}}{\text{total number of trials}}. \tag{8.7}$$

In order to combine these two measures (NFC and SR), a new measure, called success performance has been introduced as follows [10]:

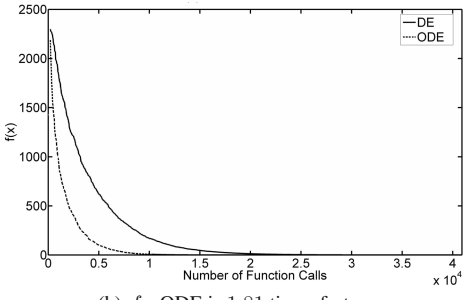
$$\text{SP} = \frac{\text{average of NFC over successful runs)}}{\text{SR}}. \tag{8.8}$$

The parameter setting for all conducted experiments is summarized in Table 8.2.

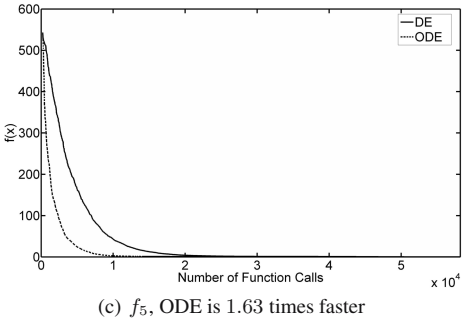
In order to maintain a reliable and fair comparison, the parameter settings are kept the same for all conducted experiments, unless we mention new settings. Besides, for all experiments, the reported values are the average of the results for 50 independent runs. In addition, and most importantly, extra fitness evaluations required for the



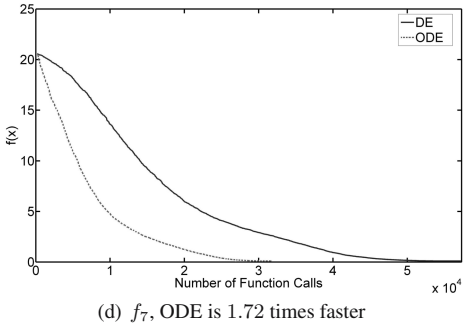
(a) f_1, ODE is 1.83 times faster



(b) f_2, ODE is 1.81 times faster



(c) f_5, ODE is 1.63 times faster



(d) f_7, ODE is 1.72 times faster

Fig. 8.7. Sample convergence graphs (best solution vs. number of function calls). As seen, ODE (dotted curve) shows better convergence speed than DE (solid curve) because it needs small amount of function calls to find the solution. is calculated by $\frac{\text{NFC}_{\text{DE}}}{\text{NFC}_{\text{ODE}}}$.

opposite points (both in population initialization and also generation jumping phases) are counted as well to accurately measure the benefit in spite of the additional overhead for computing the opposites.

The results for DE and ODE to solve the test problems are given in Table 8.3 (the results in the last column will be discussed in section 8.5.2). ODE outperforms DE on 14 benchmark functions with respect to the success performance. Some sample performance comparison graphs are presented in Figure 8.7. ODE (dotted curve) shows better convergence speed than DE (solid curve) because it needs smaller number of function calls to find the solution. With the same parameter settings for both algorithms and fixing the jumping rate for the ODE ($J_r = 0.3$), their success rates are comparable while ODE shows better convergence speed than DE. DE has a better success rate (SR) than ODE on 3 functions (f_4,f_5, and f_9). The jumping rate is an important control parameter which, if optimally set, can achieve even better results. Detailed discussions about this parameter can be found in [13].

On 7 multimodal functions (out of 8), ODE performs better than DE. This means that the opposition-based extension performs well even when the function contains many optima.

8.5.2 Contribution of Opposite Points

In this section, we verify whether the achieved acceleration rate for DE is really due to utilizing opposite points. For this purpose, all parts of the proposed algorithm remain unchanged and instead of using opposite points for the population initialization and the generation jumping, uniformly generated random points will be employed. In order to have a fair competition for this case, exactly like what we did for opposite points, the current interval (dynamic interval, $[\text{MIN}_j^p, \text{MAX}_j^p]$) of the variables are used to generate new random points in the generation jumping phase. So, line 4 in Algorithm 2 should be changed to:

$$RP_{0i,j} \longleftarrow l_j + (u_j - l_j) \times \text{RAND}(0,1),$$

where $\text{RAND}(0,1)$ generates a uniformly distributed random number on the interval $(0,1)$. In fact, instead of generating N_p, $2N_p$ random individuals are generated. In the same manner, line 30 in Algorithm 2 should be replaced with

$$RP_{i,j} \longleftarrow \text{MIN}_j^p + (\text{MAX}_j^p - \text{MIN}_j^p) \times \text{RAND}(0,1).$$

As mentioned before, the current boundaries of the variables ($[\text{MAX}_j^p, \text{MIN}_j^p]$) are used to generate random numbers for generation jumping. And finally, in order to have the same selection method, lines 7 and 33 in Algorithm 2 are substituted with

Select N_p fittest (best) individuals from P and RP as current population P;

After these modifications, the random version of ODE (called RDE) is established. Results for the current algorithm are presented in Table 8.3. As apparent, RDE can

not outperform DE or ODE on any of benchmark function with respect to the success performance. This clearly demonstrates that the achieved improvements are due to usage of opposite points, and that the same level of improvement cannot be achieved via additional random sampling [14, 15].

8.6 Conclusions and Future Work

In this chapter, we briefly reviewed how opposition-based optimization can be employed to accelerate convergence speed of differential evolution by embedding opposition-based population initialization and opposition-based generation jumping. The experimental results confirmed that ODE provides a higher performance than the classical DE. However, opposition-based optimization is still in its infancy and future research is required to fully investigate its benefits and drawbacks.

By replacing opposite points with uniformly generated random points in the same variables' range, the resulted algorithm (RDE) performs slower than the parent algorithm (DE). Therefore, the contribution of opposite points to the acceleration process was confirmed and was not reproducible by additional random sampling.

The benefits of opposition-based optimization is most likely not the same for different problems. This is because of using fixed settings for the parameters and/or the different characteristics of each problem (e.g., modality, dimension, surface features, separability of the variables and so on). Similar to all optimization approaches, ODE does not present a consistent behavior over different problems. However, over the employed benchmark test suite, ODE performed better than classical DE.

The proposed opposition-based schemes are general enough to be applied to other population-based algorithms. The opposition-based schemes work at the population level and leave the evolutionary part of the algorithms untouched. This generality gives higher flexibility to these schemes to be embedded inside other population-based algorithms.

Opposition-based optimization opens new perspectives to accelerate optimization processes. For most practical applications, we are faced with constrained functions and also with multi-objective problems. So far, there are many approaches for handling constraints in DE and also for multi-objective optimization using DE. All of these proposals can be borrowed and investigated to generalize ODE to solve multi-objective constrained problems.

References

1. Feoktistov, V.: Differential Evolution. In: Search of Solutions. Springer, USA (2006)
2. Storn, R., Price, K.: Differential Evolution - a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces, Technical Report in ICSI, TR-95-012 (1995)
3. Price, K., Storn, R.: Differential Evolution: Numerical Optimization Made Easy. Dr. Dobb's Journal 220, 18–24 (1997)
4. Storn, R., Price, K.: Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11(6), 341–359 (1997)
5. Price, K., Storn, R.M., Lampinen, J.A.: Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series), 1st edn. Springer, Heidelberg (2005)

6. Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.A.: A Novel Population Initialization Method for Accelerating Evolutionary Algorithms. Elsevier Journal on Computers and Mathematics with Applications 53(10), 1605–1614 (2007)
7. Andre, J., Siarry, P., Dognon, T.: An Improvement of the Standard Genetic Algorithm Fighting Premature Convergence in Continuous Optimization. Advance in Engineering Software 32, 49–60 (2001)
8. Hrstka, O., Kučerová, A.: Improvement of Real Coded Genetic Algorithm Based on Differential Operators Preventing Premature Convergence. Advance in Engineering Software 35, 237–246 (2004)
9. Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.A.: Opposition-Based Differential Evolution Algorithms. In: IEEE Congress on Evolutionary Computation (CEC 2006), IEEE World Congress on Computational Intelligence, Vancouver, Canada, pp. 7363–7370 (2006)
10. Suganthan, P.N., Hansen, N., Liang, J.J., Deb, K., Chen, Y.-P., Auger, A., Tiwari, S.: Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization, Technical Report, Nanyang Technological University, Singapore And KanGAL Report Number 2005005 (Kanpur Genetic Algorithms Laboratory, IIT Kanpur) (2005)
11. Vesterstroem, J., Thomsen, R.: A Comparative Study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on Numerical Benchmark Problems. In: Proceedings of the Congress on Evolutionary Computation (CEC 2004), vol. 2, pp. 1980–1987. IEEE Publications, Los Alamitos (2004)
12. Tizhoosh, H.R.: Opposition-Based Learning: A New Scheme for Machine Intelligence. In: Proceedings of International Conference on Computational Intelligence for Modelling Control and Automation - CIMCA 2005, Vienna - Austria, vol. I, pp. 695–701 (2005)
13. Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.A.: Opposition-Based Differential Evolution (ODE). Journal of IEEE Transactions on Evolutionary Computation 12(1), 64–79 (2008)
14. Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.A.: Opposition versus Randomness in Soft Computing Techniques. Elsevier Journal on Applied Soft Computing 8, 906–918 (2008)
15. Rahnamayan, S.: Opposition-Based Differential Evolution, PhD Thesis, Department of Systems Design Engineering, University of Waterloo, Waterloo, Canada (2007)
16. Eiben, A.E., Hinterding, R.: Paramater Control in Evolutionary Algorithms. IEEE Transactions on Evolutionary Computation 3(2), 124–141 (1999)
17. Das, S., Konar, A., Chakraborty, U.K.: Two Improved Differential Evolution Schemes for Faster Global Search. In: Proceedings of the 2005 conference on Genetic and evolutionary computation, Washington, USA, pp. 991–998 (2005)
18. Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.A.: Opposition-Based Differential Evolution (ODE) With Variable Jumping Rate. In: Proc. of IEEE Symposium on Foundations of Computational Intelligence, Honolulu, Hawaii, USA, pp. 81–88 (2007)
19. Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.A.: Opposition-Based Differential Evolution for Optimization of Noisy Problems. In: IEEE Congress on Evolutionary Computation (CEC 2006), IEEE World Congress on Computational Intelligence, Vancouver, Canada, pp. 6756–6763 (2006)
20. Brest, J., Greiner, S., Bošković, B., Mernik, M., Žumer, V.: Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. Journal of IEEE Transactions on Evolutionary Computation 10(6), 646–657 (2006)
21. Lee, C.Y., Yao, X.: Evolutionary programming using mutations based on the Lvy probability distribution. IEEE Transactions on Evolutionary Computation 8(1), 1–13 (2004)
22. Yao, X., Liu, Y., Lin, G.: Evolutionary programming made faster. IEEE Transactions on Evolutionary Computation 3(2), 82–102 (1999)
23. Storn, R., Price, K.: Differential Evolution- A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization 11, 341–359 (1997)

24. Liu, J., Lampinen, J.: A fuzzy adaptive differential evolution algorithm. Soft Computing-A Fusion of Foundations, Methodologies and Applications 9(6), 448–462 (2005)
25. Ali, M.M., Trn, A.: Population set-based global optimization algorithms: Some modifications and numerical studies. Comput. Oper. Res. 31(10), 1703–1725 (2004)
26. Sun, J., Zhang, Q., Tsang, E.P.K.: DE/EDA: A new evolutionary algorithm for global optimization. Information Sciences 169, 249–262 (2005)
27. Onwubolu, G.C., Babu, B.V.: New Optimization Techniques in Engineering. Springer, Berlin (2004)
28. Brest, J., Greiner, S., Bošković, B., Mernik, M., Žumer, V.: Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. Journal of IEEE Transactions on Evolutionary Computation 10(6), 646–657 (2006)
29. Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.A.: Quasi-Oppositional Differential Evolution. In: IEEE Congress on Evolutionary Computation (CEC 2007), Singapore, pp. 2229–2236 (September 2007)
30. Teo, J.: Exploring dynamic self-adaptive populations in differential evolution. Soft Computing - A Fusion of Foundations, Methodologies and Applications 10(8) (2006)
31. Tasoulis, D.K., Pavlidis, N.G., Plagianakos, V.P., Vrahatis, M.N.: Parallel Differential Evolution. In: Proceedings of the Congress on Evolutionary Computation (CEC 2004), vol. 2, pp. 2023–2029. IEEE Publications, Los Alamitos (2004)
32. Shi, Y.-J., Teng, H.-F., Li, Z.-Q.: Cooperative Co-evolutionary Differential Evolution for Function Optimization. In: Proceedings of First International Conference in Advances in Natural Computation (ICNC 2005), Changsha, China, pp. 1080–1088 (2005)
33. Fan, H.-Y., Lampinen, J.: A Trigonometric Mutation Operation to Differential Evolution. Global Optimization 27(1), 105–129 (2003)
34. Kaelo, P., Ali, M.M.: Probabilistic adaptation of point generation schemes in some global optimization algorithms. Optimization Methods and Software 27(3), 343–357 (2006)
35. Noman, N., Iba, H.: Enhancing differential evolution performance with local search for high dimensional function optimization. In: Proceedings of the 2005 conference on Genetic and evolutionary computation (GECCO 2005), Washington DC, USA, pp. 967–974 (2005)
36. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Longman Publishing Co., USA (2005)
37. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks, Piscataway, NJ, pp. 1942–1948 (1995)