Opposition-Based Differential Evolution

Shahryar Rahnamayan, Hamid R. Tizhoosh, and Magdy M.A. Salama

Faculty of Engineering, University of Waterloo, Waterloo, Canada

Summary. Although the concept of the opposition has an old history in other fields and sciences, this is the first time that it contributes to enhance an optimizer. This chapter presents a novel scheme to make the differential evolution (DE) algorithm faster. The proposed opposition-based DE (ODE) employs opposition-based optimization (OBO) for population initialization and also for generation jumping. In this work, opposite numbers have been utilized to improve the convergence rate of the classical DE. A test suite with 15 benchmark functions is employed for experimental verification. The contribution of the opposite numbers is empirically verified. Additionally, two time varying models for control parameter adjustment of ODE are investigated. Details of the ODE algorithm, the test set, and the comparison strategy are provided.

1 Introduction

The footprints of the opposition concept can be observed in many areas around us. This concept has sometimes been called by different names. Opposite particles in physics, antonyms in languages, complement of an event in probability, antithetic variables in simulation, opposite proverbs in culture, absolute or relative complement in set theory, subject and object in philosophy of science, good and evil in animism, opposition parties in politics, theses and antitheses in dialectic, opposition day in parliaments, and dualism in religions and philosophies are just some examples to mention (Table 1 contains more examples and corresponding details).

The Yin-Yang symbol in the ancient Chinese philosophy is probably the oldest opposition concept which was expressed by human kind (Figure 1). Black and white represent yin and yang, respectively. This symbol reflects the twisted duality of all things in nature, namely, receptive vs. creative, feminine vs. masculine, dark vs. bright, and finally passive force vs. active force. Greek's classical elements to explain patterns in nature (Figure 2) also mention and make use of the opposition concept, namely, fire (hot and dry) vs. water (cold and wet), earth (cold and dry) vs. air (hot and wet).

It seems that without using the opposition concept explaining of different entities around us is hard and maybe even impossible. In order to explain an entity or a situation we sometimes explain its opposite instead. In fact, opposition often offers a balance between the entities. For instance, the east, west, south, and north can not be defined

© Springer-Verlag Berlin Heidelberg 2008

alone. The same is valid for cold and hot and many other examples. Extreme opposites constitute our upper and lower boundaries. Imagination of the infinity is vague but when we consider the limited, then it becomes more imaginable because its opposite is definable. We finish the introduction with Rumi's quote:

> *Therefore, the foundation of the creation was (based) upon opposites. Necessarily, we are battling because of loss and gain.* Rumi (1207 – 1273) in "Masnawi"

In the following section, we explain how opposition concept can be utilized in optimization. Firstly, the opposite point in one- and D-dimensional spaces are defined, and secondly, the opposition-based optimization is described.

Table 1. Footprints of opposition in variant fields

Example	**Field**	**Description**
Opposite Particles/Elements	Physics	Such as magnetic poles (N and S), opposite polarities ($+$ and $-$), electron-proton in an atom, action-reaction forces in Newton's third law, and so on.
Antonyms	Language	A word that means the opposite of another word (e.g., hot/cold, fast/slow, top/down, left/right, day/night, on/off).
Antithetic Variables	Simulation	Antithetic (negatively correlated) pair of random variables used for variance reduction.
Opposite Proverbs	Culture	Two proverbs with the opposite advice or meaning (e.g., The pen is mightier than the sword. Actions speak louder than words.); proverb or its opposite pair offers an applicable solution based on specific situation or condition.
Complements	Set theory	a) Relative complement ($B - A = \{x \in B \| x \notin A\}$), b) Absolute complement ($A^c = U - A$, where U is the universal set).
Opposition	Politics	A political party or organized group opposed to the government.
Inverter	Digital design	Output of the inverter gate is one if input is zero and vice versa.
Opposition Day	Legislation	A day in the parliament in which small opposition parties are allowed to propose the subject for debate (e.g., Canada's parliament has 25 opposition days).
Dualism	Philosophy and Religion	Two fundamental principles/concepts, often in opposition to each other; such as "Yin" and "Yang" in Chinese philosophy and Taoist religion (Figure 1), "subject" and "object" in philosophy of science, "good" and "evil" in animism.
Dialectic	Philosophy	An exchange of "theses" and "antitheses" resulting in a "synthesis" (e.g. in Hinduism, these three elements are creation (Brahma), maintenance of order (Vishnu) and destruction or disorder (Shiva)).
Classical Elements	Archetype	A set of archetypal classical elements to explain patterns in nature (e.g., the Greek classical elements are Fire (hot and dry), Earth (cold and dry), Air (hot and wet), and Water (cold and wet), Figure 2).
if-then-else	Algorithm	if *condition* then *statements* [else *elsestatements*], the *else statements* are executed when the opposite of the *condition* happens.
Complement of an Event	Probability	$P(A') = 1 - P(A)$.
Revolution	socio-political	A significant Socio-political change in a short period of time.

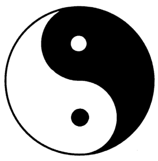


Fig. 1. Yin-Yang symbol or Taijitu in ancient Chinese philosophy. Black and white are representing yin (receptive, feminine, dark, passive force) and yang (creative, masculine, bright, active force), respectively.

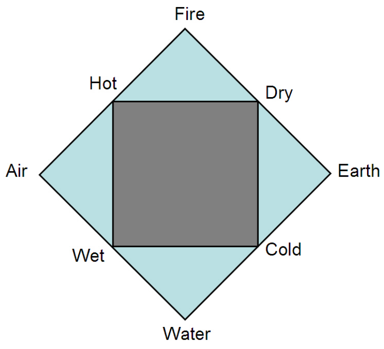


Fig. 2. Greek classical elements to explain patterns in nature are Fire (hot and dry), Earth (cold and dry), Air (hot and wet), and Water (cold and wet).

2 Opposition-Based Optimization

Generally speaking, evolutionary optimization methods start with some candidate solutions (initial population) and try to improve them toward some optimal solution(s). The process of searching terminates when some predefined criteria are satisfied. In the absence of a priori information about the solution, we usually start with a *random guesse*. The computation time, among others, is related to the distance of these initial guesses from the optimal solution. We can improve our chance of starting with a closer (fitter) solution by simultaneously checking *the opposite solution*. By doing this, the fitter one (guess or opposite guess) can be chosen as an initial solution. In fact, according to probability theory, 50% of the time a guess is farther from the solution than its opposite. So, starting with the closer of the two guesses (as judged by its fitness) has the potential to accelerate convergence. The same approach can be applied not only to initial solutions but also continuously to each solution in the current population. However, before concentrating on opposition-based optimization, we need to define the concept of opposite numbers [1]:

Definition (opposite number) - Let x be a real number in an interval $[a, b]$ ($x \in [a, b]$); the opposite of x, denoted by $\breve{x}$, is defined by

$$\breve{x} = a + b - x. \tag{1}$$

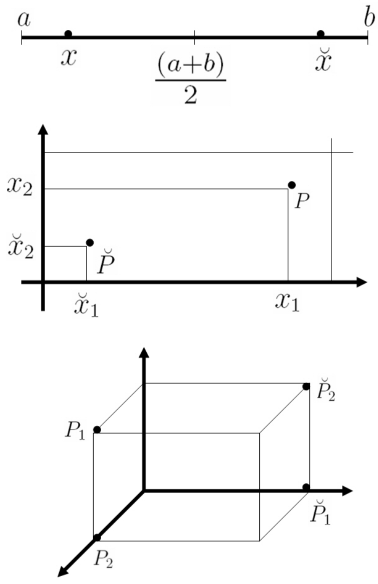


Fig. 3. Illustration of a point and its corresponding opposite

Figure 3 (top) illustrates x and its opposite $\breve{x}$ in interval $[a, b]$. As seen, x and $\breve{x}$ are located in equal distance from the interval center ($|(a+b)/2 - x| = |\breve{x} - (a+b)/2|$) and from the interval boundaries ($|x - a| = |b - \breve{x}|$) as well.

This definition can be extended to higher dimensions [1].

Definition (opposite point) - Let $P(x_1, x_2, ..., x_D)$ be a point in D-dimensional space, where $x_1, x_2, ..., x_D$ are real numbers and $x_i \in [a_i, b_i]$, $i = 1, 2, ..., D$. The opposite point of P is defined by $\breve{P}(\breve{x}_1, \breve{x}_2, ..., \breve{x}_D)$ where

$$\breve{x}_i = a_i + b_i - x_i. \tag{2}$$

Figure 3 illustrates a sample point and its corresponding opposite point in one, two, and three dimensional spaces.

Now, after the definition of the opposite points we are ready to define *Opposition-Based Optimization (OBO).*

Opposition-Based Optimization (OBO) - Let $P(x_1, x_2, ..., x_D)$, a point in a D-dimensional space with $x_i \in [a_i, b_i]$ ($i = 1, 2, 3, ..., D$), be a candidate solution. Assume $f(x)$ is a fitness function which is used to measure candidate's optimality. According to the opposite point definition, $\breve{P}(\breve{x}_1, \breve{x}_2, ..., \breve{x}_D)$ is the opposite of $P(x_1, x_2, ..., x_D)$. Now, if $f(\breve{P}) \geq f(P)$, then point P can be replaced with $\breve{P}$; otherwise we continue with P. Hence, the point and its opposite point are evaluated simultaneously to continue with the fitter one [1].

3 Opposition-Based Differential Evolution

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. We will enhance these two steps using the opposition-based optimization concept. The classical DE is chosen as a parent algorithm and the proposed opposition-based schemes are embedded in DE to accelerate its convergence speed. Corresponding pseudo-code and flowchart for the proposed approach (ODE) are given in Algorithm 1 and Figure 4, respectively. Newly added/extended code segments (which are shown by grey blocks in Figure 4) will be explained in the following subsections.

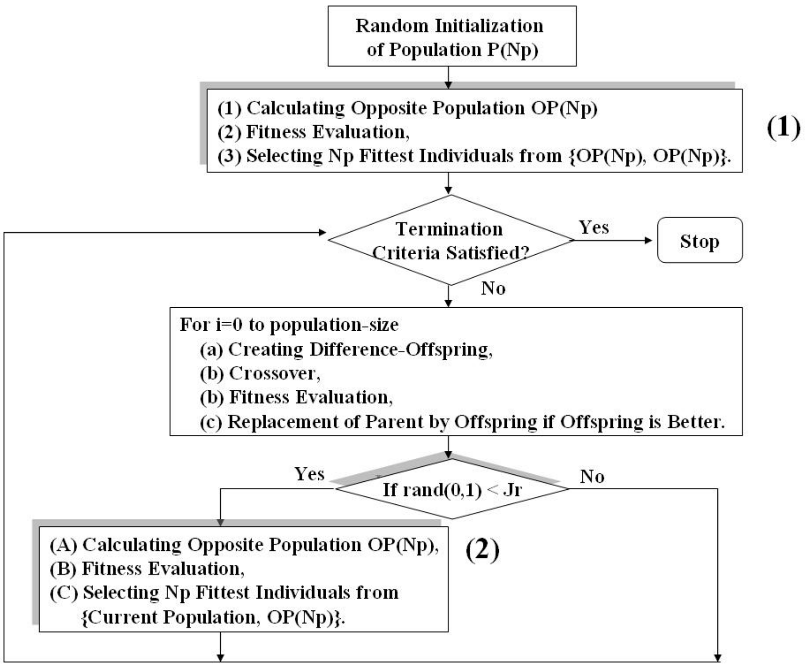

Random Initialization of Population P(Np)
(1) Calculating Opposite Population OP(Np)
(2) Fitness Evaluation,
(3) Selecting Np Fittest Individuals from {OP(Np), OP(Np)}.
(1)
Termination Criteria Satisfied?
Yes
Stop
No
For i=0 to population-size
(a) Creating Difference-Offspring,
(b) Crossover,
(b) Fitness Evaluation,
(c) Replacement of Parent by Offspring if Offspring is Better.
Yes
If rand(0,1) < Jr
No
(A) Calculating Opposite Population OP(Np),
(B) Fitness Evaluation,
(C) Selecting Np Fittest Individuals from {Current Population, OP(Np)}.
(2)

Fig. 4. New blocks are illustrated by gray boxes. 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)

3.1 Opposition-Based Population Initialization

In absence of a priori knowledge, random number generation is generally the only choice to create an initial population. But as mentioned before, by utilizing OBO we can obtain fitter starting candidates even when there is no a priori knowledge about the solution(s). Block (1) from Figure 4 shows the implementation of opposition-based population initialization. Following steps explain that procedure:

Algorithm 1. Pseudo-code for Opposition-Based Differential Evolution (ODE). P_0: Initial population, OP_0: Opposite of initial population, P: Current population, OP: Opposite of current population, D: Problem dimension, $[a_j, b_j]$: Range of the j-th variable, J_r: Jumping rate, $\min_j^p/\max_j^p$: Minimum/maximum value of the j-th variable in the current population. Steps **2-7** and **27-34** are implementations of opposition-based population initialization and opposition-based generation jumping, respectively.

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

step 1. Initialize the population $\mathrm{P}(N_p)$ randomly,
step 2. Calculate opposite population by

$$\mathrm{OP}_{i,j} = \mathrm{a}_j + \mathrm{b}_j - \mathrm{P}_{i,j}, \tag{3}$$

$$i = 1, 2, ..., N_p \; ; j = 1, 2, ..., D.$$

where $\mathrm{P}_{i,j}$ and $\mathrm{OP}_{i,j}$ denote the j^{th} variable of the i^{th} population and the opposite-population vector, respectively.
step 3. Select the N_p fittest individuals from the set $\{\mathrm{P} \cup \mathrm{OP}\}$ 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.

3.2 Opposition-Based Generation Jumping

By applying a similar approach to the current population, the evolutionary process can be forced to jump to a fitter generation. Based on a jumping rate J_r (i.e. jumping probability), after generating new populations by mutation, crossover, and selection, the opposite population is calculated and the N_p fittest individuals are selected from the union of the current population and the opposite population. As a difference to opposition-based initialization, it should be noted here that 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 in the *current population* ($[\mathrm{MIN}_j^p, \mathrm{MAX}_j^p]$) are used to calculate opposite points instead of using variables' predefined interval boundaries ($[a_j, b_j]$):

$$\mathrm{OP}_{i,j} = \mathrm{MIN}_j^p + \mathrm{MAX}_j^p - \mathrm{P}_{i,j}, i = 1, 2, ..., N_p; j = 1, 2, ..., D. \tag{4}$$

By staying within variables' static boundaries, it is possible to jump outside of the already shrunken search space and loose the knowledge of the current reduced space (converged population). Hence, we calculate opposite points by using variables' current interval in the population ($[\mathrm{MIN}_j^p, \mathrm{MAX}_j^p]$) which is, as the search does progress, increasingly smaller than the corresponding initial range $[a_j, b_j]$. Block (2) from Figure 4 indicates the implementation of opposition-based generation jumping.

A pictorial example is presented in Figure 5 to exhibit opposition-based generation jumping procedure in 2D space. The 'S' indicates location of the solution. Dark and light circles present the points and the opposite points, respectively. As seen, in the resulted population (shown by the current P), the average distance of the selected candidates (which contains some original points and the opposite of some others) from the solution is smaller than it was for population (P) and opposite population (OP), individually.

Generally speaking, in order to utilize the advantages of the opposition-based optimization to accelerate population-based algorithms, many schemes can be suggested and investigated. But, it seems that considering the following features during the design of these schemes are crucial:

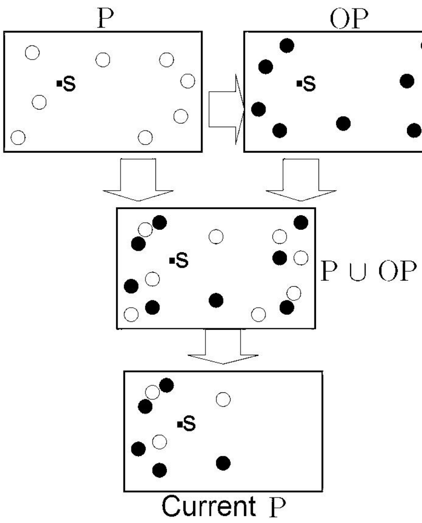


Fig. 5. A pictorial example to show the opposition-based generation jumping in 2D space ($N_p = 8$)

Generality - Proposing general schemes makes it easy to use OBO on a wider range of population-based optimization methods. Tailored schemes would be obviously more rigid for generalization. Manipulating the internal operators of the optimizer leads to lower generality, although, the customized schemes can result a higher performance.

Simplicity - This feature is always desirable in all science and engineering design problems. Simplicity supports a higher understandability and makes any design easy to implement and modify. Also, in practical environments, the simple schemes are widely appreciated.

Problem Independency - Proposed schemes have to be universal and capable to solve a wider range of optimization problems. By equipping the parent optimizer with the opposition-based schemes, it should not be specialized to solve a group of specific problems (e.g., unimodal). This case is experimentally verifiable by applying the algorithm to solve various global optimization problems. In other words, the proposed schemes should not reduce the universality of the parent optimizer to solve different problems.

Effectiveness - Considering opposite points needs more function calls and should be controlled smartly to prevent loosing the benefits through extra computations. Overall, the extra function calls should be reasonable and bring a benefit to the optimization process. The benefit can be faster convergence, higher robustness, or higher solution quality. Furthermore, improving one of these features should not affect the other benefits. During the experimental verification of the proposed algorithm, different measures are employed to investigate each criterion individually.

4 Experimental Verifications

4.1 Comparison of DE and ODE

A set of 15 well-known benchmark functions, which contains 7 unimodal and 8 multimodal functions, has been selected for performance verification of the ODE. The definition of the benchmark functions is given in Table 2.

Table 2. List of employed benchmark functions. S denotes the search space.

Name	*Definition*	*S*
1^{st} De Jong	$f_1(X) = \sum_{i=1}^{D} {x_i}^2$	$[-5.12, 5.12]^D$
Axis Parallel Hyper-Ellipsoid	$f_2(X) = \sum_{i=1}^{D} i{x_i}^2$	$[-5.12, 5.12]^D$
Schwefel's Problem 1.2	$f_3(X) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j \right)^2$	$[-65, 65]^D$
Rastrigin's Function	$f_4(X) = 10D + \sum_{i=1}^{D} \left(x_i^2 - 10\cos(2\pi x_i) \right)$	$[-5.12, 5.12]^D$
Griewangk's Function	$f_5(X) = \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}} \right) + 1$	$[-600, 600]^D$
Sum of Different Power	$f_6(X) = \sum_{i=1}^{D} \lvert x_i \rvert^{(i+1)}$	$[-1, 1]^D$
Ackley's Problem	$f_7(X) = -20\exp\left(-0.2\sqrt{\frac{\sum_{i=1}^{D} x_i^2}{D}} \right) - \exp\left(\frac{\sum_{i=1}^{D} \cos(2\pi x_i)}{D} \right) + 20 + e$	$[-32, 32]^D$
Levy Function	$f_8(X) = \sin^2(3\pi x_1) + \sum_{i=1}^{D-1} (x_i - 1)^2 (1 + \sin^2(3\pi x_{i+1})) + (x_D - 1)(1 + \sin^2(2\pi x_D))$	$[-10, 10]^D$
Michalewicz Function	$f_9(X) = -\sum_{i=1}^{D} \sin(x_i)(sin(ix_i^2/\pi))^{2m}, (m = 10)$	$[0, \pi]^D$
Zakharov Function	$f_{10}(X) = \sum_{i=1}^{D} x_i^2 + \left(\sum_{i=1}^{D} 0.5ix_i \right)^2 + \left(\sum_{i=1}^{D} 0.5ix_i \right)^4$	$[-5, 10]^D$
Schwefel's Problem 2.22	$f_{11}(X) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	$[-10, 10]^D$
Step Function	$f_{12}(X) = \sum_{i=1}^{D} (\lfloor x_i + 0.5 \rfloor)^2$	$[-100, 100]^D$
Alpine Function	$f_{13}(X) = \sum_{i=1}^{D} \lvert x_i \sin(x_i) + 0.1x_i \rvert$	$[-10, 10]^D$
Exponential Problem	$f_{14}(X) = \exp\left(-0.5 \sum_{i=1}^{D} x_i^2 \right)$	$[-1, 1]^D$
Salomon Problem	$f_{15}(X) = 1 - \cos(2\pi \parallel x \parallel) + 0.1 \parallel x \parallel$, 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 [16, 11, 2, 3, 4, 5, 6]. 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 each function is the average over 50 trials. The number of times, for which the algorithm successfully reaches the VTR for each test function is measured as the *success rate* SR:

$$\text{SR} = \frac{\text{number of times reached VTR}}{\text{total number of trials}}. \tag{5}$$

In order to combine these two measures (NFC and SR), a new measure, called *success performance*, has been introduced as follows [6]:

$$\text{SP} = \frac{\text{mean (NFC for successful runs)}}{\text{SR}}. \tag{6}$$

By this definition, the two following algorithms have equal performances (SP=100):

Algorithm A: mean (NFC for successful runs)=50 *and SR*=0.5,
Algorithm B: mean (NFC for successful runs)=100 *and SR*=1.

SP gives an equal importance weight for NFC and SR, but dependent to the different applications each of them can be more important than other one. Some times success rate is more crucial factor than convergence speed and vice versa. For our experiments, gathering results for unsuccessful case is more time consuming because the algorithm should meet the maximum number of function calls (MAX_{NFC}) for termination. Parameter settings for all conducted experiments are presented in Table 3.

Table 3. Parameter settings

Parameter name	*Setting*	*Reference*
population size (N_p)	100	[7,9,8]
differential amplification factor (F)	0.5	[10,11,13,12,7]
crossover probability constant (C_r)	0.9	[10,11,13,12,7]
jumping rate constant (J_r)	0.3	[4,14,**?**,15]
maximum number of function calls (MAX_{NFC})	10^6	[4,14,15]
value to reach (VTR)	10^{-8}	[6]
mutation strategy	$DE/rand/1/bin$	[10,16,18,7,17]

In order to maintain a reliable and fair comparison (a) the parameter settings are kept the same for all conducted experiments, unless we mention new settings, (b) for all experiments, the reported values are the average of the results for 50 independent runs, and most importantly (c) extra fitness evaluations required for the opposite points (both in population initialization and also generation jumping phases) are counted as well.

Results for DE and ODE to solve test problems are given in Table 4 (the results in the last column will be discussed in section 4.2). As seen, ODE outperforms DE on 14 benchmark functions with respect to the success performance. Some sample performance comparison graphs are presented in Figure 6. With the same control 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. The jumping rate is an important control parameter which, if optimally set, can achieve even better results. Detailed discussion about this parameter is given in [14, ?].

4.2 Contribution of Opposite Points

In this section, we want to verify that the achieved acceleration rate is really due to utilizing opposite points. For this purpose, all parts of the proposed algorithm are kept untouched and instead of using opposite points for the population initialization and the generation jumping, uniformly generated random points are employed. In order to have a fair competition for this case, exactly like what we did for opposite points, the current

Table 4. Comparison of DE, ODE, and RDE. The best result for each case is highlighted in boldface. Results for RDE has been explained in section 4.2 (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	

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 from Algorithm 1 should be changed to:

$$RP_{0i,j} \longleftarrow a_j + (b_j - a_j) \times rand(0,1),$$

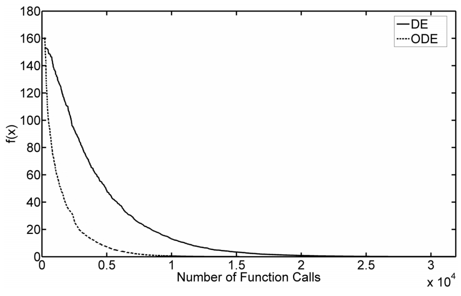
where $rand(0,1)$ generates a uniformly distributed random number on the interval $(0,1)$. This change is for the initialization part, so the predefined boundaries of variables ($[a_j, b_j]$) have been used to generate new random numbers. In fact, instead of generating N_p, $2N_p$ random individuals are generated. In the same manner, line 30 should be replaced with

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

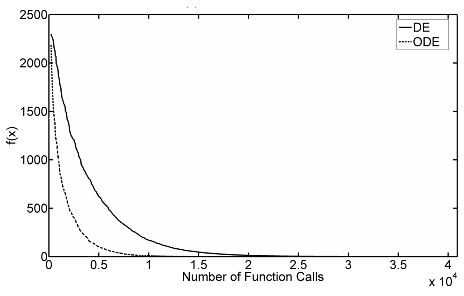
As mentioned before, for generation jumping, the current boundaries of the variables ($[\text{MAX}_j^p, \text{MIN}_j^p]$) are used to generate random numbers. And finally, in order to have the same selection method, lines 7 and 33 in Algorithm 1 are substituted with

Select N_p fittest individuals from set the $\{P, RP\}$ as current population P;

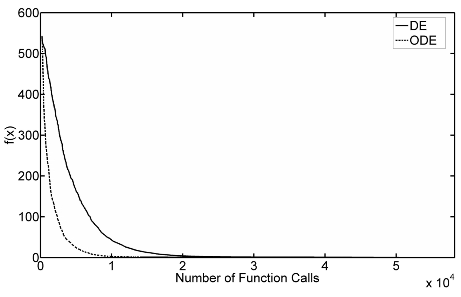
After these modifications, the random version of ODE (called RDE) is introduced. Now, we are ready to apply this algorithm to solve our test problems. All control parameters are kept the same to have a fair comparison. Results for the current algorithm are presented in Table 4. As seen, RDE can not outperform DE or ODE on any of benchmark function with respect to the success performance (even, its average success rate is less than others). 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.



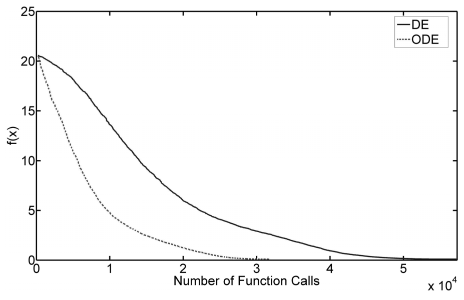
(a) f_1, ODE is 1.83 times faster



(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. 6. Sample graphs (best solution vs. number of function calls)

5 ODE with Variable Jumping Rate

In this section, a time varying jumping rate (TVJR) model for opposition-based differential evolution (ODE) has been investigated. According to this model, the jumping rate changes during the evolution based on the number of function evaluations. The same test suite has been employed to compare performance of DE and ODE with variable jumping rate settings.

Generally speaking, parameter control in evolutionary algorithms (EAs) can be performed in following three ways [19]: deterministic, adaptive, and self-adaptive. The first one uses a predefined rule to modify the parameter value without gaining any feedback from the evolution process while the second one changes the parameter value based on the information which receives from the search process. The third one utilizes the same evolutionary approach not only to solve the problem but also to optimize own control parameters by encoding some strategic parameters inside the population.

The proposed idea in this section is similar to Das *et al.* work [20]. They utilized time varying approach for setting of the scale factor F in differential evolution

(DE), which can be considered as a deterministic approach according to the mentioned categorization.

5.1 Investigated Jumping Rate Models

For ODE a constant value for jumping rates was utilized. Here, two types of varying jumping rate are investigated (linearly increasing and decreasing functions). Three proposed settings for J_r are as follows:

- J_r (constant)$= J_{r_{ave}}$,
- $J_r(\text{TVJR1}) = (J_{r_{max}} - J_{r_{min}}) \times \left(\frac{\text{MAX}_{\text{NFC}} - \text{NFC}}{\text{MAX}_{\text{NFC}}}\right)$,
- $J_r(\text{TVJR2}) = (J_{r_{max}} - J_{r_{min}}) - (J_{r_{max}} - J_{r_{min}}) \times \left(\frac{\text{MAX}_{\text{NFC}} - \text{NFC}}{\text{MAX}_{\text{NFC}}}\right)$,

where $J_{r_{ave}}$, $J_{r_{max}}$, and $J_{r_{min}}$ are the average, maximum, and minimum jumping rates, respectively. MAX_{NFC} and NFC are the maximum number of function calls and the current number of function calls, respectively.

In order to support a fair comparison between these three different jumping rate settings, the average jumping rate should be the same for all of them. Obviously we should have $J_{r_{ave}} = \frac{(J_{r_{max}} + J_{r_{min}})}{2}$. Following values for these parameters are selected: $J_{r_{ave}} = 0.3$ and $J_{r_{min}} = 0$ (no jumping), so $J_{r_{max}} = 0.6$. Figure 7 shows the corresponding diagrams (jumping rate vs. NFCs) for three following settings:

- $J_r(\text{constant}) = 0.3$,
- $J_r(\text{TVJR1}) = 0.6 \times \left(\frac{\text{MAX}_{\text{NFC}} - \text{NFC}}{\text{MAX}_{\text{NFC}}}\right)$,
- $J_r(\text{TVJR2}) = 0.6 - 0.6 \times \left(\frac{\text{MAX}_{\text{NFC}} - \text{NFC}}{\text{MAX}_{\text{NFC}}}\right)$.

$J_r(\text{TVJR1})$ represents higher jumping rate during exploration and lower jumping rate during exploitation (fine-tuning); $J_r(\text{TVJR2})$ performs exactly in reverse manner. By these settings, we can investigate effects of generation jumping during optimization process.

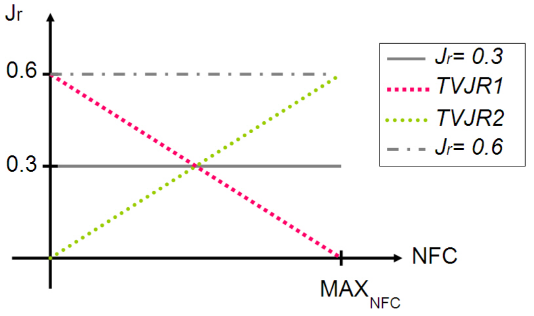


Fig. 7. Jumping rate vs. NFCs for $J_r(\text{ODE}) = 0.3$, $J_r(\text{TVJR1}) = 0.6 \times \left(\frac{\text{MAX}_{\text{NFC}} - \text{NFC}}{\text{MAX}_{\text{NFC}}}\right)$, and $J_r(\text{TVJR2}) = 0.6 - 0.6 \times \left(\frac{\text{MAX}_{\text{NFC}} - \text{NFC}}{\text{MAX}_{\text{NFC}}}\right)$

5.2 Empirical Results

The benchmark test set and all parameter settings are the same as before. The only difference is the maximum number of function calls, which is 2×10^5 for $f_1, f_2, f_3, f_6, f_8, f_{15}, f_{21}$; 5×10^5 for $f_5, f_{18}, f_{19}, f_{31}$; and 5×10^4 for $f_7, f_{23}, f_{41}, f_{56}$. Results of applying DE, ODE ($J_r = 0.3$), ODE (TVJR1), and ODE (TVJR2) to solve the test problems are given in Table 5. As seen, ODE (TVJR1) delivers best success performance (SP) for 13 benchmark functions, while this number for DE, ODE ($J_r = 0.3$), and ODE (TVJR2) is 0, 1, and 1, respectively.

Table 5. Comparison of DE, ODE ($J_r = 0.3$), ODE (TVJR1), and ODE (TVJR2). D: Dimension, NFC: Number of function calls (average over 50 trials), SR: Success rate, SP: Success performance.

		DE			*ODE ($J_r = 0.3$)*			*ODE (TVJR1)*			*ODE (TVJR2)*		
F	*D*	*NFC*	*SR*	*SP*	*NFC*	*SR*	*SP*	*NFC*	*SR*	*SP*	*NFC*	*SR*	*SP*
f_1	30	87748	1	87748	47716	1	47716	42300	1	**42300**	66305	1	66305
f_2	30	96488	1	96488	53304	1	53304	45720	1	**45720**	72990	1	72990
f_3	20	177880	1	177880	168680	1	168680	159775	1	**159775**	175460	1	175460
f_4	10	328844	1	328844	65056	0.64	101650	59063	0.80	**73829**	136070	1	136070
f_5	30	113428	1	113428	64920	0.75	86560	63594	0.90	**70660**	86235	1	86235
f_6	30	25140	1	25140	8328	1	8328	6080	1	**6080**	14175	1	14175
f_7	30	169152	1	169152	98296	1	98296	88355	1	**88355**	117095	1	117095
f_8	30	101460	1	101460	70408	1	70408	65247	0.95	**68681**	82245	1	82245
f_9	10	215260	0.56	384393	168470	0.76	**221671**	188440	0.65	289908	379660	0.60	632767
f_{10}	30	385192	1	385192	369104	1	369104	389955	1	389955	360595	1	**360595**
f_{11}	30	187300	1	187300	155636	1	155636	146795	1	**146795**	167685	1	167685
f_{12}	30	41588	1	41588	23124	1	23124	20290	1	**20290**	29165	1	29165
f_{13}	30	411164	1	411164	337532	1	337532	326350	1	**326350**	377425	1	377425
f_{14}	10	19528	1	19528	15704	1	15704	14270	1	**14270**	17735	1	17735
f_{15}	10	37824	1	37824	24260	1	24260	21400	1	**21400**	28710	1	28710
	SR$_{ave}$		0.97			0.94			0.95			0.97	

Pairwise comparison of these algorithms is presented in Table 6. The given number in each cell indicates on how many functions the algorithm in each row outperforms the corresponding algorithm in each column. The last column of the table shows the total numbers (number of cases which the algorithm outperforms other competitors); by comparing these numbers the following ranking is obtained: ODE (TVJR1) (best), ODE ($J_r = 0.3$), ODE (TVJR2), and DE.

Table 6. Pairwise comparison of DE, ODE ($J_r = 0.3$), ODE (TVJR1), and ODE (TVJR2). Given number in each cell shows how many functions the algorithm in each row outperforms the corresponding algorithm in each column. The last column shows the total numbers (number of cases which the algorithm outperforms other competitors).

	DE	*ODE ($J_r = 0.3$)*	*ODE (TVJR1)*	*ODE (TVJR2)*	Total
DE	-	0	1	1	**2**
ODE ($J_r = 0.3$)	15	-	2	12	**29**
ODE (TVJR1)	14	13	-	14	**41**
ODE (TVJR2)	14	3	1	-	**18**

The average success rate (shown in the last row of the Table 5) for DE and ODE (TVJR2) is marginally better than other two competitors.

6 Conclusion

In this chapter, the concept of opposition-based optimization (OBO) has been employed to accelerate differential evolution. The OBO was utilized to introduce opposition-based population initialization and opposition-based generation jumping. By embedding these two steps within DE, opposition-based differential evolution (ODE) was proposed. The experimental results clearly confirmed that ODE is faster than the classical DE. Our conclusion can be summarized as follows:

- 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.
- According to our comprehensive experiments (not included in this chapter), the range $[0.1, 0.4]$ is recommended for an unknown optimization problem. Most of the functions presented a reliable acceleration improvement and almost a smooth behavior in this interval. Although, the optimal jumping rate can be somewhere out of this range, higher jumping rates are generally not recommended.
- As an advantageous of an opposite versus random point, purely random resampling or selection of solutions from a given population has the higher chance of visiting or even revisiting unproductive regions of the search space compared to the opposite points [25] .
- The benefits of opposition-based optimization is not the same for different problems. This is because of using fix settings for the parameters instead of the optimal ones 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, in overall and over the employed benchmark test suite, ODE performed better than classical DE.
- The proposed opposition-based schemes are general enough to be applied on 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 for further investigation.
- The optimal control parameters are problem-oriented such that developing a self-adaptive/ adaptive algorithm is a valuable attempt. Many studies confirm that for population-based algorithms the optimal parameters are problem-oriented. Running limited trials to determine desirable parameters is a common approach (if not a practical way for time consuming objective functions). For this reason, the self-adaptive/adaptive control parameter setting would be a valuable improvement for ODE.
- The time varying jumping rate for opposition-based differential evolution was proposed and two behaviorally reverse versions (linearly decreasing and increasing

functions) were compared with the constant setting [15]. The results show that the linearly decreasing jumping rate performs better than constant setting and also than linearly increasing policy. This means generation jumping in the exploration time is more desirable than during exploitation. Because during the fine-tuning, we are faced with shrunken search space and the jumping of the individuals may not be advantageous (because the point and the opposite-point are very close together). There is no exact border between exploration and exploitation stage. Hence, the gradual behavior for the decreasing and increasing functions are proposed.

- The proposed time varying jumping rate functions utilize the maximum number of function calls ($\mathrm{MAX}_{\mathrm{NFC}}$) which may not be exactly known for the black-box optimization problems; this can be regarded as a disadvantage for this method. Adaptive setting of the jumping rate can be a desirable solution.
- Results are promising, however, the opposition-based optimization is still in its infancy. Results confirm that the opposition concept has the potential to play desire and positive role in optimization. But, it is important to mention that the current study constitutes the first step of this newly opened direction. Like many other new concepts, opposition-based optimization needs further studies to disclose its exact benefits, weaknesses, and limitations. In fact, the main claim is not defeating DE or any of its numerous versions but to introduce a new notion into optimization via metaheuristics; this is the notion of opposition.

The opposition-based optimization is simple to implement and open to be used in many different ways for different purposes. This study is a starting point in this direction to confirm the potentials and motivate other researchers in optimization and machine learning fields to engage with the opposition concept.

References

1. Tizhoosh, H.R.: Opposition-Based Learning: A New Scheme for Machine Intelligence. In: Int. Conf. on Computational Intelligence for Modelling Control and Automation (CIMCA 2005), Vienna, Austria, vol. I, pp. 695–701 (2005)
2. 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)
3. 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)
4. 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)
5. 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)
6. 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) (May 2005)

7. 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)
8. Lee, C.Y., Yao, X.: Evolutionary programming using mutations based on the Lévy probability distribution. IEEE Transactions on Evolutionary Computation 8(1), 1–13 (2004)
9. Yao, X., Liu, Y., Lin, G.: Evolutionary programming made faster. IEEE Transactions on Evolutionary Computation 3(2), 82–102 (1999)
10. 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)
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. Liu, J., Lampinen, J.: A fuzzy adaptive differential evolution algorithm. Soft Computing - A Fusion of Foundations, Methodologies and Applications 9(6), 448–462 (2005)
13. Ali, M.M., Törn, A.: Population set-based global optimization algorithms: Some modifications and numerical studies. Comput. Oper. Res. 31(10), 1703–1725 (2004)
14. 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)
15. Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.A.: Opposition-Based Differential Evolution (ODE) With Variable Jumping Rate. In: IEEE Symposium on Foundations of Computational Intelligence, Honolulu, Hawaii, USA, April 2007, pp. 81–88 (2007)
16. Price, K., Storn, R.M., Lampinen, J.A.: Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series), 1st edn. Springer, Heidelberg (2005)
17. Sun, J., Zhang, Q., Tsang, E.P.K.: DE/EDA: A new evolutionary algorithm for global optimization. Information Sciences 169, 249–262 (2005)
18. Onwubolu, G.C., Babu, B.V.: New Optimization Techniques in Engineering. Springer, Berlin (2004)
19. Eiben, A.E., Hinterding, R.: Paramater Control in Evolutionary Algorithms. IEEE Transactions on Evolutionary Computation 3(2), 124–141 (1999)
20. 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)
21. Tizhoosh, H.R.: Reinforcement Learning Based on Actions and Opposite Actions. In: ICGST International Conference on Artificial Intelligence and Machine Learning (AIML 2005), Cairo, Egypt (2005)
22. Tizhoosh, H.R.: Opposition-Based Reinforcement Learning. Journal of Advanced Computational Intelligence and Intelligent Informatics 10(3) (2006)
23. Shokri, M., Tizhoosh, H.R., Kamel, M.: Opposition-Based Q(λ) Algorithm. In: 2006 IEEE World Congress on Computational Intelligence (IJCNN 2006), Vancouver, Canada, pp. 646–653 (2006)
24. Ventresca, M., Tizhoosh, H.R.: Improving the Convergence of Backpropagation by Opposite Transfer Functions. In: 2006 IEEE World Congress on Computational Intelligence (IJCNN 2006), Vancouver, Canada, pp. 9527–9534 (2006)
25. 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)
26. Brest, J., Greiner, S., Bošković, B., Mernik, M., Žumer, V.: Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. IEEE Transactions on Evolutionary Computation 10(6), 646–657 (2006)
27. Rahnamayan, S.: Opposition-Based Differential Evolution, PhD Thesis, Departement of Systems Design Engineering, University of Waterloo, Waterloo, Canada (May 2007)