# Quasi-Oppositional Differential Evolution

Shahryar Rahnamayan[1], Hamid R. Tizhoosh[1], Magdy M.A. Salama[2]
Faculty of Engineering, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada
[1]Pattern Analysis and Machine Intelligence (PAMI) Research Group
[1,2]Medical Instrument Analysis and Machine Intelligence (MIAMI) Research Group
shahryar@pami.uwaterloo.ca, tizhoosh@uwaterloo.ca, m.salama@ece.uwaterloo.ca

*Abstract*— **In this paper, an enhanced version of the Opposition-Based Differential Evolution (ODE) is proposed. ODE utilizes opposite numbers in the population initialization and generation jumping to accelerate Differential Evolution (DE). Instead of opposite numbers, in this work, quasi opposite points are used. So, we call the new extension Quasi- Oppositional DE (QODE). The proposed mathematical proof shows that in a black-box optimization problem quasi- opposite points have a higher chance to be closer to the solution than opposite points. A test suite with 15 benchmark functions has been employed to compare performance of DE, ODE, and QODE experimentally. Results confirm that QODE performs better than ODE and DE in overall. Details for the proposed approach and the conducted experiments are provided.**

## I. INTRODUCTION

Differential Evolution (DE) was proposed by Price and Storn in 1995 [16]. It is an effective, robust, and simple global optimization algorithm [8] which has only a few control parameters. According to frequently reported comprehensive studies [8], [22], DE outperforms many other optimization methods in terms of convergence speed and robustness over common benchmark functions and real-world problems. Generally speaking, all population-based optimization algorithms, no exception for DE, suffer from long computational times because of their evolutionary nature. This crucial drawback sometimes limits their application to off-line problems with little or no real time constraints.

The concept of *opposition-based learning* (OBL) was introduced by Tizhoosh [18] and has thus far been applied to accelerate reinforcement learning [15], [19], [20], backpropagation learning [21], and differential evolution [10]–[12], [14]. The main idea behind OBL is the simultaneous consideration of an estimate and its corresponding opposite estimate (i.e. guess and opposite guess) in order to achieve a better approximation of the current candidate solution. Opposition-based deferential evolution (ODE) [9], [10], [14] uses opposite numbers during population initialization and also for generating new populations during the evolutionary process.

In this paper, an OBL has been utilized to accelerate ODE. In fact, instead of opposite numbers, quasi opposite points are used to accelerate ODE. For this reason, we call the new method Quasi-Oppositional DE (QODE) which employs exactly the same schemes of ODE for population initialization and generation jumping.

Purely random sampling or selection of solutions from a given population has the chance of visiting or even revisiting unproductive regions of the search space. A mathematical proof has been provided to show that, in general, opposite numbers are more likely to be closer to the optimal solution than purely random ones [13]. In this paper, we prove the quasi-opposite points have higher chance to be closer to solution than opposite points. Our experimental results confirm that QODE outperforms DE and ODE.

The organization of this paper is as follows: Differential Evolution, the parent algorithm, is briefly reviewed in section II. In section III, the concept of opposition-based learning is explained. The proposed approach is presented in section IV. Experimental verifications are given in section V. Finally, the work is concluded in section VI.

## II. DIFFERENTIAL EVOLUTION

Differential Evolution (DE) is a population-based and directed search method [6], [7]. Like other evolutionary algorithms, it starts with an initial population vector, which is randomly generated when no preliminary knowledge about the solution space is available.

Let us assume that $X_{i,G}(i = 1, 2, ..., N_p)$ are solution vectors in generation $G$ ($N_p$ =population size). Successive populations are generated by adding the weighted difference of two randomly selected vectors to a third randomly selected vector.

For classical DE ($DE/rand/1/bin$), the mutation, crossover, and selection operators are straightforwardly defined as follows:

**Mutation -** For each vector $X_{i,G}$ in generation $G$ a mutant vector $V_{i,G}$ is defined by

$$V_{i,G} = X_{a,G} + F(X_{b,G} - X_{c,G}), \tag{1}$$

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

**Crossover -** DE utilizes the crossover operation to generate new solutions by shuffling competing vectors and also to increase the diversity of the population. For the classical version of the DE ($DE/rand/1/bin$), the binary crossover (shown by 'bin' in the notation) is utilized. It defines the following trial vector:

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

where $j = 1, 2, ..., D$ ($D =$ problem dimension) and

$$U_{ji,G} = \begin{cases} V_{ji,G} & \text{if } rand_j(0,1) \leq C_r \vee j = k, \\ X_{ji,G} & \text{otherwise.} \end{cases} \tag{3}$$

$C_r \in (0, 1)$ is the predefined crossover rate constant, and $rand_j(0, 1)$ is the $j^{th}$ evaluation of a uniform random number generator. $k \in \{1, 2, ..., D\}$ is a random parameter index, chosen once for each $i$ to make sure that at least one parameter is always selected from the mutated vector, $V_{ji,G}$. Most popular values for $C_r$ are in the range of $(0.4, 1)$ [3].

**Selection -** The approach that must decide which vector ($U_{i,G}$ or $X_{i,G}$) should be a member of next (new) generation, $G + 1$. For a maximization problem, the vector with the higher fitness value is chosen. There are other variants based on different mutation and crossover strategies [16].

## III. Opposition-Based Learning

Generally speaking, evolutionary optimization methods start with some initial 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 some *random guesses*. 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 guesses*. By doing this, the fitter one (guess or opposite guess) can be chosen as an initial solution. In fact, according to probability theory, the likelihood that a guess is further from the solution than its opposite guess is 50%. So, starting with the fitter of the two, guess or opposite guess, 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.

Before concentrating on quasi-oppositional version of DE, we need to define the concept of opposite numbers [18]:

**Definition (Opposite Number) -** Let $x \in [a, b]$ be a real number. The opposite number $\breve{x}$ is defined by

$$\breve{x} = a + b - x. \tag{4}$$

Similarly, this definition can be extended to higher dimensions as follows [18]:

**Definition (Opposite Point) -** Let $P(x_1, x_2, ..., x_n)$ be a point in $n$-dimensional space, where $x_1, x_2, ..., x_n \in R$ and $x_i \in [a_i, b_i]$ $\forall i \in \{1, 2, ..., n\}$. The opposite point $\breve{P}(\breve{x}_1, \breve{x}_2, ..., \breve{x}_n)$ is completely defined by its components

$$\breve{x}_i = a_i + b_i - x_i. \tag{5}$$

As we mentioned before, opposition-based differential evolution (ODE) employs opposite points in population initialization and generation jumping to accelerate the classical DE. In this paper, in order to enhance the ODE, instead of opposite points a quasi opposite points are utilized. Figure 1 and figure 2 show the interval and region which are used to generate these points in one-dimensional and two-dimensional spaces, respectively.

Fig. 1. Illustration of $x$, its opposite $\breve{x}$, and the interval $[M, \breve{x}]$ (showed by dotted arrow) which the quasi opposite point, $\breve{x}^q$, is generated in this interval.

Fig. 2. For a two-dimensional space, the point $P$, its opposite $\breve{P}$, and the region (illustrated by shadow) which the quasi opposite point, $\breve{P}^q$, is generated in that area.

Mathematically, we can prove that for a black-box optimization problem (which means solution can appear anywhere over the search space), that the quasi opposite point $\breve{x}^q$ has a higher chance than opposite point $\breve{x}$ to be closer to the solution. This proof can be as follows:

**Theorem -** Given a guess $x$, its opposite $\breve{x}$ and quasi-opposite $\breve{x}^q$, and given the distance from the solution $d(.)$ and probability function $P_r(.)$, we have

$$P_r\left[d(\breve{x}^q) < d(\breve{x})\right] > 1/2 \tag{6}$$

**Proof -** Assume, the solution is in one of these intervals: $[a, x]$, $[x, M]$, $[M, \breve{x}]$, $[\breve{x}, b]$

($[a, x] \cup [x, M] \cup [M, \breve{x}] \cup [\breve{x}, b] = [a, b]$). We investigate all cases:

- $[a, x]$,$[\breve{x}, b]$ - According to the definition of opposite point, intervals $[a, x]$ and $[\breve{x}, b]$ have the same length, so the probability of that the solution be in interval $[a, x]$ or $[\breve{x}, b]$ is equal ($\frac{x-a}{b-a} = \frac{b-\breve{x}}{b-a}$). Now, if the solution is in interval $[a, x]$, definitely, it is closer to $\breve{x}^q$ and in the same manner if it is in interval $[\breve{x}, b]$ it would be closer to $\breve{x}$. So, until now, $\breve{x}^q$ and $\breve{x}$ have the equal chance to be closer to the solution.
- $[M, \breve{x}]$ - For this case, according to probability theory, $\breve{x}^q$ and $\breve{x}$ have the equal chance to be closer to the solution.
- $[x, M]$ - For this case, obviously, $\breve{x}^q$ is closer to the solution than $\breve{x}$.

Now, we can conclude that, in overall, $\breve{x}^q$ has a higher chance to be closer to the solution than $\breve{x}$, because for the first two cases they had equal chance and just for last case ($[x, M]$) $\breve{x}^q$ has a higher chance to be closer to the solution. ■

This proof is for a one-dimensional space but the conclusion is the same for the higher dimensions:

$$P_r\left[d(\breve{P}^q) < d(\breve{P})\right] > 1/2 \qquad (7)$$

Because according to the definition of Euclidean distance between two points $Y(y_1, y_2, ..., y_D)$ and $Z(z_1, z_2, ..., z_D)$ in a D-dimensional space

$$d(Y, Z) = \parallel Y, Z \parallel = \sqrt{\sum_{i=1}^{D}(y_i - z_i)^2}, \qquad (8)$$

If in each dimension $\breve{x}^q$ has a higher chance to be closer to the solution than $\breve{x}$, consequently, the point $\breve{P}^q$, in a D-dimensional space, will have a higher chance to be closer to solution than $P$. Now let us define an optimization process which uses a quasi-oppositional scheme.

**Quasi-Oppositional Optimization**

Let $P(x_1, x_2, ..., x_D)$ be a point in an D-dimensional space (i.e. a candidate solution) and $\breve{P}^q(\breve{x}^q_1, \breve{x}^q_2, ..., \breve{x}^q_D)$ be a quasi opposite point (see figure 2). Assume $f(\cdot)$ is a fitness function which is used to measure the candidate's fitness. Now, if $f(\breve{P}^q) \geq f(P)$, then point $P$ can be replaced with $\breve{P}^q$; otherwise we continue with $P$. Hence, we continue with the fitter one.

## IV. Proposed Algorithm

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 selection, crossover, and mutation. Similar to ODE, we will enhance these two steps using the quasi-oppositional scheme. The classical DE is chosen as a parent algorithm and the proposed scheme is embedded in DE to accelerate the convergence speed. Corresponding pseudo-code for the proposed approach (QODE) is given in Table I. Newly added/extended code segments will be explained in the following subsections.

### *A. Quasi-Oppositional Population Initialization*

According to our review of optimization literature, random number generation, in absence of a priori knowledge, is the only choice to create an initial population. By utilizing quasi-oppositional learning we can obtain fitter starting candidate solutions even when there is no a priori knowledge about the solution(s). Steps 1-12 from Table I present the implementation of quasi-oppositional initialization for QODE. Following steps show that procedure:

1) Initialize the population $\mathrm{P}_0(N_P)$ randomly,
2) Calculate quasi-opposite population ($\mathrm{QOP}_0$), steps 5-10 from Table I,
3) Select the $N_p$ fittest individuals from $\{\mathrm{P}_0 \cup \mathrm{QOP}_0\}$ as initial population.

### *B. Quasi-Oppositional Generation Jumping*

By applying a similar approach to the current population, the evolutionary process can be forced to jump to a new solution candidate, which ideally is fitter than the current one. Based on a jumping rate $J_r$ (i.e. jumping probability), after generating new populations by selection, crossover, and mutation, the quasi-opposite population is calculated and the $N_p$ fittest individuals are selected from the union of the current population and the quasi-opposite population. As a difference to quasi-oppositional initialization, it should be noted here that in order to calculate the quasi-opposite population for generation jumping, the opposite of each variable and middle point are calculated dynamically. That is, the maximum and minimum values of each variable in the *current population* ($[\mathrm{MIN}^p_j, \mathrm{MAX}^p_j]$) are used to calculate middle-to-opposite points instead of using variables' predefined interval boundaries ($[a_j, b_j]$).

By staying within variables' interval static boundaries, we would jump outside of the already shrunken search space and the knowledge of the current reduced space (converged population) would be lost. Hence, we calculate new points by using variables' current interval in the population ($[\mathrm{MIN}^p_j, \mathrm{MAX}^p_j]$) which is, as the search does progress, increasingly smaller than the corresponding initial range $[a_j, b_j]$. Steps 33-46 from Table I show the implementation of quasi- oppositional generation jumping for QODE.

TABLE I

Pseudo-code for Quasi-Oppositional Differential Evolution (QODE). $P_0$: Initial population, $OP_0$: Opposite of initial population, $N_p$: Population size, $P$: Current population, $OP$: Opposite of current population, $V$: Noise vector, $U$: Trial vector, $D$: Problem dimension, $[a_j, b_j]$: Range of the $j$-th variable, BFV: Best fitness value so far, VTR: Value to reach, NFC: Number of function calls, $\text{MAX}_{\text{NFC}}$: Maximum number of function calls, F: Mutation constant, $rand(0,1)$: Uniformly generated random number, $C_r$: Crossover rate, $f(\cdot)$: Objective function, $P'$: Population of next generation, $J_r$: Jumping rate, $\text{MIN}_j^p$: Minimum value of the $j$-th variable in the current population, $\text{MAX}_j^p$: Maximum value of the $j$-th variable in the current population, $M_{i,j}$: Middle Point. Steps **1-12** and **33-46** are implementations of quasi-oppositional population initialization and generation jumping, respectively.

**Quasi-Oppositional Differential Evolution (QODE)**

```
      /* Quasi-Oppositional Population Initialization */
1.      Generate uniformly distributed random population $P_0$;
2.      for ($i = 0$ ; $i < N_p$ ; $i++$)
3.        for ($j = 0$ ; $j < D$ ; $j++$)
4.          {
5.            $OP_{0i,j} = a_j + b_j - P_{0i,j}$;
6.            $M_{i,j} = (a_j + b_j)/2$;
7.            if ($P_{0i,j} < M_{i,j}$)
8.              $QOP_{0i,j} = M_{i,j} + (OP_{0i,j} - M_{i,j}) \times rand(0,1)$;
9.            else
10.             $QOP_{0i,j} = OP_{0i,j} + (M_{i,j} - OP_{0i,j}) \times rand(0,1)$;
11.         }
12.     Select $N_p$ fittest individuals from set the $\{P_0, QOP_0\}$ as initial population $P_0$;
      /* End of Quasi-Oppositional Population Initialization */

13.       while ( BFV $>$ VTR and NFC $<$ $\text{MAX}_{\text{NFC}}$ )
14.       {
15.         for ($i = 0$ ; $i < N_p$ ; $i++$ )
16.         {
17.              Select three parents $P_{i_1}$, $P_{i_2}$, and $P_{i_3}$ randomly from current population where $i \neq i_1 \neq i_2 \neq i_3$;
18.              $V_i = P_{i_1} + F \times (P_{i_2} - P_{i_3})$;
19.              for ($j = 0$ ; $j < D$ ; $j++$ )
20.              {
21.                if ($rand(0,1) < C_r \vee j = k$)
22.                  $U_{i,j} = V_{i,j}$;
23.                else
24.                  $U_{i,j} = P_{i,j}$;
25.              }
26.              Evaluate $U_i$;
27.              if ($f(U_i) \leq f(P_i)$)
28.                $P'_i = U_i$;
29.              else
30.                $P'_i = P_i$;
31.         }
32.              $P = P'$;

           /* Quasi-Oppositional Generation Jumping */
33.              if ($rand(0,1) < J_r$)
34.              {
35.                for ($i = 0$ ; $i < N_p$ ; $i++$)
36.                  for ($j = 0$ ; $j < D$ ; $j++$)
37.                    {
38.                      $OP_{i,j} = \text{MIN}_j^p + \text{MAX}_j^p - P_{i,j}$;
39.                      $M_{i,j} = (\text{MIN}_j^p + \text{MAX}_j^p)/2$;
40.                        if ($P_{i,j} < M_{i,j}$)
41.                          $QOP_{i,j} = M_{i,j} + (OP_{i,j} - M_{i,j}) \times rand(0,1)$;
42.                        else
43.                          $QOP_{i,j} = OP_{i,j} + (M_{i,j} - OP_{i,j}) \times rand(0,1)$;
44.                    }
45.                Select $N_p$ fittest individuals from set the $\{P, QOP\}$ as current population $P$;
46.              }
           /* End of Quasi-Oppositional Generation Jumping */

47.       }
```

## V. Experimental Verification

In this section we describe the benchmark functions, comparison strategies, algorithm settings, and present the results.

### A. Benchmark Functions

A set of 15 benchmark functions (7 unimodal and 8 multimodal functions) has been used for performance verification of the proposed approach. Furthermore, test functions with two different dimensions ($D$ and $2 * D$) have been employed in the conducted experiments. By this way, the classical differential evolution (DE), opposition-based DE (ODE), and quasi-oppositional DE (QODE) are compared on 30 minimization problems. The definition of the benchmark functions and their global optimum(s) are listed in Appendix A. The 13 functions (out of 15) have an optimum in the center of searching space, to make it asymmetric the search space for all of these functions are shifted $\frac{a}{2}$ as follows:

If O.P.B.: $-a \leq x_i \leq a$ and $f_{min} = f(0, ..., 0) = 0$ then S.P.B.:$-a + \frac{a}{2} \leq x_i \leq a + \frac{a}{2}$,

where O.P.B. and S.P.B. stand for *original parameter bounds* and *shifted parameter bounds*, respectively.

### B. Comparison Strategies and Metrics

In this study, three metrics, namely, *number of function calls* (NFC), *success rate* (SR), and *success performance* (SP) [17] have been utilized to compare the algorithms. We compare the convergence speed by measuring the number of function calls which is the most commonly used metric in literature [10]–[12], [14], [17]. 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 $\text{MAX}_{\text{NFC}}$. In order to minimize the effect of the stochastic nature of the algorithms on the metric, the reported number of function calls for each function is the average over 50 trials.

The number of times, for which the algorithm succeeds to reach the VTR for each test function is measured as the success rate SR:

$$\text{SR} = \frac{\text{number of times reached VTR}}{\text{total number of trials}}. \tag{9}$$

The average success rate ($\text{SR}_{ave}$) over $n$ test functions are calculated as follows:

$$\text{SR}_{ave} = \frac{1}{n} \sum_{i=1}^{n} \text{SR}_i. \tag{10}$$

Both of NFC and SR are important measures in an optimization process. So, two individual objectives should be considered simultaneously to compare competitors. In order to combine these two metrics, a new measure, called success performance (SP), has been introduced as follows [17]:

$$\text{SP} = \frac{\text{mean (NFC for successful runs)}}{\text{SR}}. \tag{11}$$

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 is our main measure to judge which algorithm performs better than others.

### C. Setting Control Parameters

Parameter settings for all conducted experiments are as follows:

- Population size, $N_p = 100$ [2], [4], [23]
- Differential amplification factor, $F = 0.5$ [1], [2], [5], [16], [22]
- Crossover probability constant, $C_r = 0.9$ [1], [2], [5], [16], [22]
- Jumping rate constant for ODE, $J_{rODE} = 0.3$ [10]–[12], [14]
- Jumping rate constant for QODE, $J_{rQODE} = 0.05$
- Maximum number of function calls, $\text{MAX}_{\text{NFC}} = 10^6$
- Value to reach, VTR$= 10^{-8}$ [17]

The jumping rate for QODE is set to a smaller value ($J_{rQODE} = \frac{1}{6} J_{rODE}$) because our trials showed that the higher jumping rates can reduce diversity of the population very fast and cause a premature convergence. This was predictable for QODE because instead of opposite point a random point between middle point and the opposite point is generated and so variable's search interval is prone to be shrunk very fast. A complementary study is required to determine an optimal value/interval for QODE's jumping rate.

### D. Results

Results of applying DE, ODE, and QODE to solve 30 test problems (15 test problems with two different dimensions) are given in Table II. The best NFC and the success performance for each case are highlighted in boldface. As seen, QODE outperforms DE and ODE on 22 functions, ODE on 6 functions, and DE just on one function. DE performs marginally better than ODE and QODE in terms of average success rate (0.90, 0.88, and 0.86, respectively). ODE surpasses DE on 26 functions.

As we mentioned before, the success performance is a measure which considers the number of function calls and the success rate simultaneously and so it can be utilized for a reasonable comparison of optimization algorithms.

## VI. Conclusion

In this paper, the quasi-oppositional DE (QODE), an enhanced version of the opposition-based differential evolution (ODE), is proposed. Both algorithms (ODE and QODE) use the same schemes for population initialization and generation jumping. But, QODE uses quasi-opposite points instead of opposite points. The presented mathematical proof confirms that this point has a higher chance than opposite point to be closer to the solution. Experimental results, conducted on 30 test problems, clearly show that QODE outperforms ODE and DE. Number of function calls, success rate, and success performance are three metrics which were employed to compare DE, ODE, and QODE in this study.

According to our studies on the opposition-based learning, this field presents the promising potentials but still requires many deep theoretical and empirical investigations. Control parameters study (jumping rate in specific), adaptive setting of the jumping rate, and investigating of QODE on a more comprehensive test set are our directions for the future study.

## References

[1] M. Ali and A. Törn. Population set-based global optimization algorithms: Some modifications and numerical studies. *Journal of Computers and Operations Research*, 31(10):1703–1725, 2004.

[2] 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. *Journal of IEEE Transactions on Evolutionary Computation*, 10(6):646–657, 2006.

[3] S. Das, A. Konar, and U. Chakraborty. Improved differential evolution algorithms for handling noisy optimization problems. In *Proceedings of IEEE Congress on Evolutionary Computation Conference*, pages 1691–1698, Napier University, Edinburgh, UK, September 2005.

[4] C. Y. Lee and X. Yao. Evolutionary programming using mutations based on the lévy probability distribution. *Journal of IEEE Transactions on Evolutionary Computation*, 8(1):1–13, 2004.

[5] J. Liu and J. Lampinen. A fuzzy adaptive differential evolution algorithm. *Journal of Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 9(6):448–462, 2005.

[6] G. C. Onwubolu and B. Babu. *New Optimization Techniques in Engineering*. Springer, Berlin, New York, 2004.

[7] K. Price. *An Introduction to Differential Evolution*. McGraw-Hill, London (UK), 1999. ISBN: 007-709506-5.

[8] K. Price, R. Storn, and J. Lampinen. *Differential Evolution : A Practical Approach to Global Optimization*. Springer-Verlag, Berlin/Heidelberg/ Germany, 1st edition edition, 2005. ISBN: 3540209506.

[9] S. Rahnamayan. *Opposition-Based Differential Evolution*. Phd thesis, Deptartement of Systems Design Engineering, University of Waterloo, Waterloo, Canada, April 2007.

[10] S. Rahnamayan, H. Tizhoosh, and M. Salama. Opposition-based differential evolution algorithms. In *Proceedings of the 2006 IEEE World Congress on Computational Intelligence (CEC-2006)*, pages 2010–2017, Vancouver, BC, Canada, July 2006.

[11] S. Rahnamayan, H. Tizhoosh, and M. Salama. Opposition-based differential evolution for optimization of noisy problems. In *Proceedings of the 2006 IEEE World Congress on Computational Intelligence (CEC-2006)*, pages 1865–1872, Vancouver, BC, Canada, July 2006.

[12] S. Rahnamayan, H. Tizhoosh, and M. Salama. Opposition-based differential evolution with variable jumping rate. In *IEEE Symposium on Foundations of Computational Intelligence*, Honolulu, Hawaii, USA, April 2007.

[13] S. Rahnamayan, H. Tizhoosh, and M. Salama. Opposition versus randomness in soft computing techniques. *submitted to the Elsevier Journal on Applied Soft Computing*, Aug. 2006.

[14] S. Rahnamayan, H. Tizhoosh, and M. Salama. Opposition-based differential evolution. *accepted at the Journal of IEEE Transactions on Evolutionary Computation*, Dec. 2006.

[15] M. Shokri, H. R. Tizhoosh, and M. Kamel. Opposition-based q($\lambda$) algorithm. In *Proceedings of the 2006 IEEE World Congress on Computational Intelligence (IJCNN-2006)*, pages 649–653, Vancouver, BC, Canada, July 2006.

[16] R. Storn and K. Price. Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[17] 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 2005005, Kanpur Genetic Algorithms Laboratory, IIT Kanpur, Nanyang Technological University, Singapore And KanGAL, May 2005.

[18] H. Tizhoosh. Opposition-based learning: A new scheme for machine intelligence. In *Proceedings of the International Conference on Computational Intelligence for Modelling Control and Automation (CIMCA-2005)*, pages 695–701, Vienna, Austria, 2005.

[19] H. Tizhoosh. Reinforcement learning based on actions and opposite actions. In *Proceedings of the International Conference on Artificial Intelligence and Machine Learning (AIML-2005)*, Cairo, Egypt, 2005.

[20] H. Tizhoosh. Opposition-based reinforcement learning. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 10(3), 2006.

[21] M. Ventresca and H. Tizhoosh. Improving the convergence of backpropagation by opposite transfer functions. In *Proceedings of the 2006 IEEE World Congress on Computational Intelligence (IJCNN-2006)*, pages 9527–9534, Vancouver, BC, Canada, July 2006.

[22] J. Vesterstroem and R. Thomsen. 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), IEEE Publications*, volume 2, pages 1980–1987, San Diego, California, USA, July 2004.

[23] X. Yao, Y. Liu, and G. Lin. Evolutionary programming made faster. *Journal of IEEE Transactions on Evolutionary Computation*, 3(2):82, 1999.

## Appendix A. List of benchmark functions

***O.P.B.* and *S.P.B.* stand for the original parameter bounds and the shifted parameter bounds, respectively. All the conducted experiments are based on S.P.B.**

- $1^{st}$ *De Jong*

$$f_1(X) = \sum_{i=1}^{n} {x_i}^2,$$
$$\text{O.P.B.} -5.12 \le x_i \le 5.12,$$
$$\text{S.P.B.} -2.56 \le x_i \le 7.68,$$
$$min(f_1) = f_1(0,...,0) = 0.$$

TABLE II

COMPARISON OF DE, ODE, AND QODE. D: DIMENSION, NFC: NUMBER OF FUNCTION CALLS (AVERAGE OVER 50 TRIALS), SR: SUCCESS RATE, SP: SUCCESS PERFORMANCE. THE LAST ROW OF THE TABLE PRESENTS THE AVERAGE SUCCESS RATES. THE BEST NFC AND THE SUCCESS PERFORMANCE FOR EACH CASE ARE HIGHLIGHTED IN **boldface**. DE, ODE, AND QODE ARE UNABLE TO SOLVE $f_{10}$ ($D = 60$).

| | | *DE* | | | *ODE* | | | *QODE* | | |
|---|---|---|---|---|---|---|---|---|---|---|
| *F* | *D* | *NFC* | *SR* | *SP* | *NFC* | *SR* | *SP* | *NFC* | *SR* | *SP* |
| $f_1$ | 30 | 86072 | 1 | 86072 | 50844 | 1 | 50844 | **42896** | 1 | **42896** |
| | 60 | 154864 | 1 | 154864 | 101832 | 1 | 101832 | **94016** | 1 | **94016** |
| $f_2$ | 30 | 95080 | 1 | 95080 | 56944 | 1 | 56944 | **47072** | 1 | **47072** |
| | 60 | 176344 | 1 | 176344 | 117756 | 1 | 117756 | **105992** | 1 | **105992** |
| $f_3$ | 20 | 174580 | 1 | 174580 | 177300 | 1 | 177300 | **116192** | 1 | **116192** |
| | 40 | 816092 | 1 | 816092 | 834668 | 1 | 834668 | **539608** | 1 | **539608** |
| $f_4$ | 10 | 323770 | 0.96 | 337260 | **75278** | 0.92 | **81823** | 181100 | 1 | 181100 |
| | 20 | 811370 | 0.08 | 10142125 | **421300** | 0.16 | **2633125** | 615280 | 0.16 | 3845500 |
| $f_5$ | 30 | 111440 | 0.96 | 116083 | **74717** | 0.92 | **81214** | 100540 | 0.80 | 125675 |
| | 60 | 193960 | 1 | 193960 | 128340 | 0.68 | 188735 | **115280** | 0.68 | **169529** |
| $f_6$ | 30 | 18760 | 1 | 18760 | 10152 | 1 | 10152 | **9452** | 1 | **9452** |
| | 60 | 33128 | 1 | 33128 | **11452** | 1 | **11452** | 14667 | 0.84 | 17461 |
| $f_7$ | 30 | 168372 | 1 | 168372 | 100280 | 1 | 100280 | **82448** | 1 | **82448** |
| | 60 | 294500 | 1 | 294500 | **202010** | 0.96 | **210427** | 221850 | 0.72 | 308125 |
| $f_8$ | 30 | 101460 | 1 | 101460 | 70408 | 1 | 70408 | **50576** | 1 | **50576** |
| | 60 | 180260 | 0.84 | 215000 | 121750 | 0.60 | **202900** | **98300** | 0.40 | 245800 |
| $f_9$ | 10 | **191340** | 0.76 | **252000** | 213330 | 0.56 | 380900 | 247640 | 0.48 | 515900 |
| | 20 | 288300 | 0.35 | 824000 | 253910 | 0.55 | 461700 | **193330** | 0.68 | **284300** |
| $f_{10}$ | 30 | 385192 | 1 | 385192 | 369104 | 1 | 369104 | **239832** | 1 | **239832** |
| | 60 | – | 0 | – | – | 0 | – | – | 0 | – |
| $f_{11}$ | 30 | 183408 | 1 | 183408 | 167580 | 1 | 167580 | **108852** | 1 | **108852** |
| | 60 | 318112 | 1 | 318112 | 274716 | 1 | 274716 | **183132** | 1 | **183132** |
| $f_{12}$ | 30 | 40240 | 1 | 40240 | 26400 | 1 | 26400 | **21076** | 1 | **21076** |
| | 60 | 73616 | 1 | 73616 | 64780 | 1 | 64780 | **64205** | 1 | **64205** |
| $f_{13}$ | 30 | 386920 | 1 | 386920 | 361884 | 1 | 361884 | **291448** | 1 | **291448** |
| | 60 | 432516 | 1 | 432516 | 425700 | 0.96 | 443438 | **295084** | 1 | **295084** |
| $f_{14}$ | 10 | 19324 | 1 | 19324 | 16112 | 1 | 16112 | **13972** | 1 | **13972** |
| | 20 | 45788 | 1 | 45788 | 31720 | 1 | 31720 | **23776** | 1 | **23776** |
| $f_{15}$ | 10 | 37260 | 1 | 37260 | 26108 | 1 | 26108 | **18944** | 1 | **18944** |
| | 20 | 176872 | 1 | 176872 | 57888 | 1 | 57888 | **40312** | 1 | **40312** |
| | SR$_{ave}$ | | **0.90** | | | 0.88 | | | 0.86 | |

- *Axis Parallel Hyper-Ellipsoid*

$$f_2(X) = \sum_{i=1}^{n} i{x_i}^2,$$
$$\text{O.P.B.} -5.12 \leq x_i \leq 5.12,$$
$$\text{S.P.B.} -2.56 \leq x_i \leq 7.68,$$
$$min(f_2) = f_2(0, ..., 0) = 0.$$

- *Schwefel's Problem 1.2*

$$f_3(X) = \sum_{i=1}^{n}(\sum_{j=1}^{i} x_j)^2,$$
$$\text{O.P.B.} -65 \leq x_i \leq 65,$$
$$\text{S.P.B.} -32.5 \leq x_i \leq 97.5,$$
$$min(f_3) = f_3(0, ..., 0) = 0.$$

- *Rastrigin's Function*

$$f_4(X) = 10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i)),$$
$$\text{O.P.B.} -5.12 \leq x_i \leq 5.12,$$
$$\text{S.P.B.} -2.56 \leq x_i \leq 7.68,$$
$$min(f_4) = f_4(0, ..., 0) = 0.$$

- *Griewangk's Function*

$$f_5(X) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1,$$
$$\text{O.P.B.} -600 \leq x_i \leq 600,$$
$$\text{S.P.B.} -300 \leq x_i \leq 900,$$
$$min(f_5) = f_5(0, ..., 0) = 0.$$

- *Sum of Different Power*

$$f_6(X) = \sum_{i=1}^{n} |x_i|^{(i+1)},$$
$$\text{O.P.B.} -1 \leq x_i \leq 1,$$
$$\text{S.P.B.} -0.5 \leq x_i \leq 1.5,$$
$$min(f_6) = f_6(0, ..., 0) = 0.$$

- *Ackley's Problem*

$f_7(X) = -20\exp(-0.2\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}) - \exp(\frac{\sum_{i=1}^{n}\cos(2\pi x_i)}{n}) + 20 + e,$

O.P.B. $-32 \leq x_i \leq 32,$

S.P.B. $-16 \leq x_i \leq 48,$

$min(f_7) = f_7(0, ..., 0) = 0.$

- *Levy Function*

$$f_8(X) = \sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2$$
$$(1 + \sin^2(3\pi x_{i+1})) + (x_n - 1)(1 + \sin^2(2\pi x_n)),$$
with $-10 \leq x_i \leq 10$,
$min(f_8) = f_8(1, ..., 1) = 0.$

- *Michalewicz Function*

$$f_9(X) = -\sum_{i=1}^{n} \sin(x_i)(sin(ix_i^2/\pi))^{2m},$$
with $0 \leq x_i \leq \pi, m = 10,$
$min(f_{9_{(n=10)}}) = -9.66015.$

- *Zakharov Function*

$$f_{10}(X) = \sum_{i=1}^{n} x_i^2 + (\sum_{i=1}^{n} 0.5ix_i)^2 + (\sum_{i=1}^{n} 0.5ix_i)^4,$$
with $-5 \leq x_i \leq 10,$
$min(f_{10}) = f_{10}(0, ..., 0) = 0.$

- *Schwefel's Problem 2.22*

$$f_{11}(X) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|,$$
O.P.B. $-10 \leq x_i \leq 10,$
S.P.B. $-5 \leq x_i \leq 15,$
$min(f_{11}) = f_{11}(0, ..., 0) = 0.$

- *Step Function*

$$f_{12}(X) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2,$$
O.P.B. $-100 \leq x_i \leq 100,$
S.P.B. $-50 \leq x_i \leq 150,$
$min(f_{12}) = f_{12}(-0.5 \leq x_i < 0.5) = 0.$

- *Alpine Function*

$$f_{13}(X) = \sum_{i=1}^{n} |x_i \sin(x_i) + 0.1x_i|,$$
O.P.B. $-10 \leq x_i \leq 10,$
S.P.B. $-5 \leq x_i \leq 15,$
$min(f_{13}) = f_{13}(0, ..., 0) = 0.$

- *Exponential Problem*

$$f_{14}(X) = \exp(-0.5 \sum_{i=1}^{n} x_i^2),$$
O.P.B. $-1 \leq x_i \leq 1,$
S.P.B. $-0.5 \leq x_i \leq 1.5,$
$min(f_{14}) = f_{14}(0, ..., 0) = 1.$

- *Salomon Problem*

$$f_{15}(X) = 1 - \cos(2\pi \parallel x \parallel) + 0.1 \parallel x \parallel,$$
where $\parallel x \parallel = \sqrt{\sum_{i=1}^{n} x_i^2},$
O.P.B. $-100 \leq x_i \leq 100,$
S.P.B. $-50 \leq x_i \leq 150,$
$min(f_{15}) = f_{15}(0, 0, ..., 0) = 0.$