Proceedings of the 2007 IEEE Symposium on
Foundations of Computational Intelligence (FOCI 2007)

Opposition-Based Differential Evolution (ODE) with Variable Jumping Rate

S. Rahnamayan[1], H.R. Tizhoosh[1], M.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, a time varying jumping rate (TVJR) model for Opposition-Based Differential Evolution (ODE) has been proposed. According to this model, the jumping rate changes linearly during the evolution based on the number of function evaluations. A test suite with 15 well-known benchmark functions has been employed to compare performance of the DE and ODE with variable jumping rate settings. Results show that a higher jumping rate is more desirable during the exploration than during the exploitation. Details for the proposed approach and the conducted experiments are provided.**

I. INTRODUCTION

Generally speaking, parameter control in Evolutionary Algorithms (EAs) can be performed by following three ways [1]: 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 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 idea proposed in this paper is similar to Das *et al.* work [2]. 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.

The concept of *opposition-based learning* (OBL) was introduced by Tizhoosh [3] and has thus far been applied to accelerate reinforcement learning [4]–[6], backpropagation learning [7], and differential evolution [8]–[10]. 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 differential evolution (ODE) [8] uses opposite numbers during population initialization and also for generating new populations during the evolutionary process. ODE introduces a new parameter, called jumping rate. In this paper, a time varying policy to set this parameter in order to achieve higher convergence velocity will be proposed.

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 opposition-based DE (ODE) is reviewed and also the proposed jumping rate setting policies are discussed 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 [11], [12]. Like many other evolutionary algorithms, it starts with an initial population vector, which is randomly generated when no a priori knowledge about the solution space is available.

Let us assume that $X_{i,G}(i = 1, 2, ..., N_p)$ are candidate solution vectors in the 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 higher diversity in the generated population and lower values cause faster convergence.

Crossover - DE utilizes the crossover operation to increase the diversity of the population. It defines the following trial vector:

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

where D is the problem dimension and

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

$C_r \in (0, 1)$ is the predefined crossover rate constant, and $rand_j(0, 1)$ is the j^{th} evaluation of a uniform random number generator. Most popular values for C_r are in the range of $(0.4, 1)$ [2].

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 strategies [13].

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 any a priori information about the solution, we usually start with *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 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 further from the solution than its opposite. So, starting with the closer of the two guesses (as judged by their 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 learning, we need to define the concept of opposite numbers [3]:

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 [3]:

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

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

Now, by employing the opposite point definition, the opposition-based optimization can be defined as follows:

Opposition-Based Optimization - Let $P = (x_1, x_2, ..., x_n)$ be a point in an n-dimensional space (i.e. a candidate solution). Assume $f(\cdot)$ is a fitness function which is used to measure the candidate's fitness. According to the definition of the opposite point, $\breve{P} = (\breve{x}_1, \breve{x}_2, ..., \breve{x}_n)$ is the opposite of $P = (x_1, x_2, ..., x_n)$. 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 in order to continue with the fitter one.

IV. Revisiting ODE and proposing variable jumping rates

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 learning scheme. The original DE is chosen as a parent algorithm and the proposed opposition-based ideas are embedded in DE to accelerate its convergence velocity. Corresponding pseudo-code for the ODE is given in Table I. Newly added/extended code segments will be explained in the following subsections.

A. Opposition-Based Population Initialization

Random number generation, in absence of a priori knowledge, is a widely used choice to create an initial population. But as mentioned in section III, by utilizing opposition-based learning we can obtain fitter starting candidate solutions even when there is no knowledge about the solution(s). Steps 1-5 from Table I show the implementation of opposition-based initialization for the ODE. Following steps show that procedure:

1) Initialize the population P(N_P) randomly,
2) Calculate opposite population by

$$\text{OP}_{i,j} = \text{a}_j + \text{b}_j - \text{P}_{i,j}, \tag{6}$$

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

where $\text{P}_{i,j}$ and $\text{OP}_{i,j}$ denote j^{th} variable of the i^{th} vector of the population and the opposite-population, respectively.

3) Select the N_p fittest individuals from $\{\text{P} \cup \text{OP}\}$ as initial population.

B. Opposition-Based 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 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* ($[\text{MIN}_j^p, \text{MAX}_j^p]$) are used to calculate opposite points instead of using variables' predefined interval boundaries ($[a_j, b_j]$):

$$\text{OP}_{i,j} = \text{MIN}_j^p + \text{MAX}_j^p - \text{P}_{i,j}, \qquad (7)$$
$$i = 1, 2, ..., N_p \;;\; j = 1, 2, ..., D.$$

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 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 $[a_j, b_j]$. Steps 26-32 from Table I show the implementation of opposition-based generation jumping for ODE.

C. Proposed jumping rate settings

In the first version of the opposition-based differential evolution (ODE) [8]–[10], a constant value for jumping rate was used ($J_r = 0.3$). In this paper, two types of time varying jumping rate are investigated (linearly increasing and decreasing functions). Three proposed settings for the current investigation are as follows:

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

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 as fair as possible comparison between these three different jumping rate settings, the average jumping rate should be the same for all of them. So, obviously we should have $J_{r_{ave}} = \frac{(J_{r_{max}} + J_{r_{min}})}{2}$. Following values for these parameters are selected: $J_{r_{ave}} = 0.3$ [8]–[10] and $J_{r_{min}} = 0$ (no jumping), so $J_{r_{max}} = 0.6$ is resulted. Figure 1 shows the corresponding diagrams (jumping rate vs. NFCs) for four following settings:

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

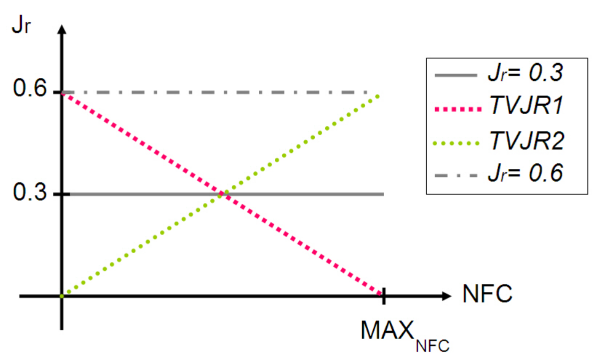

Fig. 1. Jumping rate vs. NFCs for $J_r(\text{ODE}) = 0.3$, $J_r(\text{TVJR1}) = 0.6 \times (\frac{\text{MAX}_{\text{NFC}} - \text{NFC}}{\text{MAX}_{\text{NFC}}})$, $J_r(\text{TVJR2}) = 0.6 - 0.6 \times (\frac{\text{MAX}_{\text{NFC}} - \text{NFC}}{\text{MAX}_{\text{NFC}}})$, and $J_r(\text{ODE}) = 0.6$.

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

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 well-known benchmark functions has been used for performance verification of the proposed approach. The classical differential evolution (DE) and the opposition-based DE (ODE) with different jumping

TABLE I

PSEUDO-CODE FOR OPPOSITION-BASED DIFFERENTIAL EVOLUTION (ODE). 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, MAX_{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, $\text{MIN}_j^p/\text{MAX}_j^p$: MINIMUM/MAXIMUM VALUE OF THE j-TH VARIABLE IN THE CURRENT POPULATION. STEPS **1-5** AND **26-32** ARE IMPLEMENTATIONS OF OPPOSITION-BASED INITIALIZATION AND OPPOSITION-BASED GENERATION JUMPING, RESPECTIVELY.

/* Opposition-Based 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. $OP_{0i,j} = a_j + b_j - P_{0i,j}$;
5. Select N_p fittest individuals from set the $\{P_0, OP_0\}$ as initial population P_0;

/* End of Opposition-Based Population Initialization */

6. **while** (BFV $>$ VTR **and** NFC $<$ MAX_{NFC})
7. {
8.   **for** $(i = 0\ ;\ i < N_p\ ;\ i++)$
9.   {
10.   Select three parents P_a, P_b, and P_c randomly from current population where $i \neq a \neq b \neq c$;
11. $V_i = P_a + F \times (P_b - P_c)$;
12.   **for** $(j = 0\ ;\ j < D\ ;\ j++)$
13.   {
14.   **if** $(rand(0,1) < C_r)$
15. $U_{i,j} = V_{i,j}$;
16.   **else**
17. $U_{i,j} = P_{i,j}$;
18.   }
19.   Evaluate U_i;
20.   **if** $(f(U_i) \leq f(P_i))$
21. $P'_i = U_i$;
22.   **else**
23. $P'_i = P_i$;
24.   }
25. $P = P'$;

    /* Opposition-Based Generation Jumping */

26.   **if** $(rand(0,1) < J_r)$
27.   {
28.   **for** $(i = 0\ ;\ i < N_p\ ;\ i++)$
29.   **for** $(j = 0\ ;\ j < D\ ;\ j++)$
30. $OP_{i,j} = \text{MIN}_j^p + \text{MAX}_j^p - P_{i,j}$;
31.   Select N_p fittest individuals from set the $\{P, OP\}$ as current population P;
32.   }

    /* End of Opposition-Based Generation Jumping */

33. }

---

rate setting policies (constant values and time varyings) are compared. The definition of the benchmark functions and their global optimum(s) are listed in Appendix A.

B. Comparison Strategies and Metrics

In this study, three metrics, namely, *number of function calls* (NFC), *success rate* (SR), and *success performance* (SP) [14] have been utilized to compare the algorithms. We compare the convergence velocity by measuring the number of function calls which is the most commonly used metric in the literature [8]–[10], [14]. A smaller NFC means higher convergence velocity. 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 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:

$$\text{SR} = \frac{\text{number of times reached VTR}}{\text{total number of trials}}. \tag{8}$$

The average success rate (SR_{ave}) over n test functions are calculated as follows:

$$\text{SR}_{ave} = \frac{1}{n}\sum_{i=1}^{n}\text{SR}_i. \tag{9}$$

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 [14]:

$$\text{SP} = \frac{\text{mean (NFC for successful runs)}}{\text{SR}}. \tag{10}$$

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 the main measure to judge which algorithm performs better.

C. Setting Control Parameters

Parameter settings for all conducted experiments are as follows:

- Population size, $N_p = 100$ [15]–[17]
- Differential amplification factor, $F = 0.5$ [13], [15], [18]–[20]
- Crossover probability constant, $C_r = 0.9$ [13], [15], [18]–[20]
- Maximum number of function calls, MAX_{NFC}, 2×10^5 for $f_1, f_2, f_3, f_5, f_7, f_8, f_{11}$; 5×10^5 for f_4, f_9, f_{10}, f_{13}; 5×10^4 for $f_6, f_{12}, f_{14}, f_{15}$
- Value to reach, VTR= 10^{-8} [14]
- Jumping rates (see Figure 1):
  $J_r(\text{ODE}) = 0.3$ [8]–[10]
  $J_r(\text{TVJR1}) = 0.6 \times (\frac{\text{MAX}_{\text{NFC}}-\text{NFC}}{\text{MAX}_{\text{NFC}}})$
  $J_r(\text{TVJR2}) = 0.6 - 0.6 \times (\frac{\text{MAX}_{\text{NFC}}-\text{NFC}}{\text{MAX}_{\text{NFC}}})$
  $J_r(\text{ODE}) = 0.6$

D. Results

Results of applying DE, ODE ($J_r = 0.3$), ODE ($J_r = 0.6$), ODE (TVJR1), and ODE (TVJR2) to solve 15 test problems are given in Table II. The best success performance for each function is highlighted in boldface. The last rows of the table present the sum (for NFCs and SPs) and the average success rates. We can rank these algorithms as ODE (TVJR1) (best), ODE ($J_r = 0.3$), ODE ($J_r = 0.6$), ODE (TVJR2), and DE with the respect to the total success performance to solve 15 problems. 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 more reliable comparison of the optimization algorithms. ODE ($J_r = 0.6$) presents the lowest average success rate (0.89); while DE and ODE (TVJR2) show the highest one (0.97).

Pair comparison of these algorithms are presented in Table III. Given number in each cell shows on how many functions the algorithm in the table's row outperforms the corresponding algorithm in the table's column. The last column of the table shows the total numbers (number of cases which the algorithm can outperform other competitors); by comparing these numbers we obtain the following ranking result: ODE (TVJR1) (best), ODE ($J_r = 0.6$), ODE ($J_r = 0.3$), ODE (TVJR2), and DE. ODE ($J_r = 0.6$) and ODE ($J_r = 0.3$) have changed their positions compared to the previous ranking. But, ODE (TVJR1) still keeps the first position in both rankings. Results for $J_r = 0.3$ and $J_r = 0.6$ confirm that the constant higher jumping rate reduces the overall success rate.

VI. Conclusion

In this paper, the time varying jumping rate for opposition-based differential evolution was proposed and two behaviorally reverse versions of them (linearly decreasing and increasing functions) were compared with the constant settings ($J_{r_{ave}}$ and $J_{r_{max}}$). The results

TABLE II

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. THE LAST ROWS OF THE TABLE PRESENT THE SUM (FOR NFCS AND SPS) AND THE AVERAGE SUCCESS RATES. THE BEST SUCCESS PERFORMANCE FOR EACH CASE IS HIGHLIGHTED IN **boldface**.

		DE			*ODE* ($J_r = 0.3$)			*ODE (TVJR1)*			*ODE (TVJR2)*			*ODE* ($J_r = 0.6$)		
F	*D*	*NFC*	*SR*	*SP*	*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	41260	1	**41260**
f_2	30	96488	1	96488	53304	1	53304	45720	1	**45720**	72990	1	72990	46396	0.96	48329
f_3	20	177880	1	177880	168680	1	168680	159775	1	**159775**	175460	1	175460	160720	1	160720
f_4	10	328844	1	328844	65056	0.64	101650	59063	0.80	**73829**	136070	1	136070	121920	0.60	203200
f_5	30	113428	1	113428	64920	0.75	86560	63594	0.90	70660	86235	1	86235	62828	0.90	**69808**
f_6	30	25140	1	25140	8328	1	8328	6080	1	6080	14175	1	14175	5715	1	**5715**
f_7	30	169152	1	169152	98296	1	98296	88355	1	**88355**	117095	1	117095	87617	0.90	97352
f_8	30	101460	1	101460	70408	1	70408	65247	0.95	68681	82245	1	82245	62785	1	**62785**
f_9	10	215260	0.56	384393	168470	0.76	**221671**	188440	0.65	289908	379660	0.60	632767	207690	0.35	593400
f_{10}	30	385192	1	385192	369104	1	369104	389955	1	389955	360595	1	**360595**	395115	1	395115
f_{11}	30	187300	1	187300	155636	1	155636	146795	1	146795	167685	1	167685	136180	1	**136180**
f_{12}	30	41588	1	41588	23124	1	23124	20290	1	**20290**	29165	1	29165	20460	1	20460
f_{13}	30	411164	1	411164	337532	1	337532	326350	1	**326350**	377425	1	377425	347010	0.75	462680
f_{14}	10	19528	1	19528	15704	1	15704	14270	1	14270	17735	1	17735	13600	1	**13600**
f_{15}	10	37824	1	37824	24260	1	24260	21400	1	21400	28710	1	28710	19735	1	**19735**
	$\sum$	2397996		2567129	1670538		1781973	**1637634**		**1764368**	2111550		2364657	1729031		2330339
	SR_{ave}		0.97			0.94			0.95			0.97			0.89	

TABLE III

PAIR COMPARISON OF DE, ODE ($J_r = 0.3$), ODE (TVJR1), ODE (TVJR2), AND ODE ($J_r = 0.6$). THE LAST COLUMN SHOWS THE TOTAL NUMBERS (NUMBER OF CASES WHICH THE ALGORITHM CAN OUTPERFORM OTHER COMPETITORS).

	DE	*ODE* ($J_r = 0.3$)	*ODE (TVJR1)*	*ODE (TVJR2)*	*ODE* ($J_r = 0.6$)	Total
DE	-	0	1	1	3	**5**
ODE ($J_r = 0.3$)	15	-	2	12	4	**33**
ODE (TVJR1)	14	13	-	14	8	**49**
ODE (TVJR2)	14	3	1	-	3	**21**
ODE ($J_r = 0.6$)	12	11	7	12	-	**42**

confirm that the linearly decreasing jumping rate performs better than constant settings 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. We know that there is no exact border between exploration and exploitation time. Hence, the gradual behavior for the decreasing and increasing functions are proposed.

The proposed jumping rate function utilizes the maximum number of function calls (MAX_{NFC}) which may not be exactly known for some black-box optimization problem; this can be regarded as a disadvantage for the proposed method. Adaptive setting of the jumping rate can be a desirable solution which will be a focus of our research in future.

REFERENCES

[1] A.E. Eiben, R. Hinterding, *Paramater Control in Evolutionary Algorithms*, IEEE Transactions on Evolutionary Computation, Vol. 3, no. 2, pp. 124-141, 1999.

[2] S. Das, A. Konar, U.K. Chakraborty, *Two Improved Differential Evolution Schemes for Faster Global Search*, Proceedings of the 2005 conference on Genetic and evolutionary computation, pp. 991-998, Washington, USA, 2005.

[3] H.R. Tizhoosh, *Opposition-Based Learning: A New Scheme for Machine Intelligence*, Int. Conf. on Computational Intelligence for Modelling Control and Automation (CIMCA-2005), Vol. I, pp. 695-701, Vienna, Austria, 2005.

[4] H.R. Tizhoosh, *Reinforcement Learning Based on Actions and Opposite Actions*, Int. Conf. on Artificial Intelligence and Machine Learning (AIML-2005), Cairo, Egypt, 2005.

[5] H.R. Tizhoosh, *Opposition-Based Reinforcement Learning*, Journal of Advanced Computational Intelligence and Intelligent Informatics, Vol. 10, No. 3, 2006.

[6] M. Shokri, H. R. Tizhoosh, M. Kamel, *Opposition-Based Q(λ) Algorithm*, 2006 IEEE World Congress on Computational Intelligence (IJCNN-2006), Vancouver, BC, Canada, pp. 646-653, 2006.

[7] M. Ventresca and H.R. Tizhoosh, *Improving the Convergence of Backpropagation by Opposite Transfer Functions*, 2006 IEEE World Congress on Computational Intelligence (IJCNN-2006), Vancouver, BC, Canada, pp. 9527-9534, 2006.

[8] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, *Opposition-Based Differential Evolution Algorithms*, 2006 IEEE World Congress on Computational Intelligence (CEC-2006), Vancouver, BC, Canada, pp. 7363-7370, 2006.

[9] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, *Opposition-Based Differential Evolution for Optimization of Noisy Problems*, 2006 IEEE World Congress on Computational Intelligence (CEC-2006), Vancouver, BC, Canada, pp. 6756-6763, 2006.

[10] S. Rahnamayan, H.R. Tizhoosh, M.M.A Salama, *Opposition-Based Differential Evolution*, accepted at the Journal of IEEE Transactions on Evolutionary Computation, Dec. 2006 .

[11] K. Price, *An Introduction to Differential Evolution*, In: D. Corne, M. Dorigo, F. Glover (eds) New Ideas in Optimization, McGraw-Hill, London (UK), pp. 79-108, 1999, ISBN:007-709506-5.

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

[13] R. Storn and K. Price, *Differential Evolution- A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces*, Journal of Global Optimization 11, pp. 341-359, 1997.

[14] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. -P. Chen, A. Auger, S. Tiwari, *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.

[15] J. Brest, S. Greiner, B. Bošković, Marjan Mernik, Viljem Žumer, *Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems,* accepted in IEEE Transactions on Evolutionary Computation.

[16] X. Yao, Y. Liu, and G. Lin, *Evolutionary programming made faster,* IEEE Transactions on Evolutionary Computation, Vol. 3, No. 2, p.82, Jul. 1999.

[17] C. Y. Lee and X. Yao, *Evolutionary programming using mutations based on the Lévy probability distribution,* IEEE Transactions on Evolutionary Computation, Vol. 8, No. 1, pp. 1-13, Feb. 2004.

[18] J. Vesterstroem and R. Thomsen, *A Comparative Study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on Numerical Benchmark Problems.* Proceedings of the Congress on Evolutionary Computation (CEC-2004), IEEE Publications, Vol. 2, pp. 1980-1987, 2004.

[19] M.M. Ali and A. Törn, *Population set-based global optimization algorithms: Some modifications and numerical studies,* Comput. Oper. Res., Vol. 31, No. 10, pp. 1703-1725, 2004.

[20] J. Liu and J. Lampinen, *A fuzzy adaptive differential evolution algorithm,* Soft Computing-A Fusion of Foundations, Methodologies and Applications, Vol. 9, No. 6, pp. 448-462, 2005.

APPENDIX A. LIST OF BENCHMARK FUNCTIONS

- 1^{st} *De Jong*

$$f_1(X) = \sum_{i=1}^{n} {x_i}^2,$$
$$\text{with} -5.12 \leq x_i \leq 5.12,$$
$$min(f_1) = f_1(0, ..., 0) = 0.$$

- *Axis Parallel Hyper-Ellipsoid*

$$f_2(X) = \sum_{i=1}^{n} i{x_i}^2,$$
$$\text{with} - 5.12 \leq x_i \leq 5.12,$$
$$min(f_2) = f_2(0, ..., 0) = 0.$$

- *Schwefel's Problem 1.2*

$$f_3(X) = \sum_{i=1}^{n}(\sum_{j=1}^{i} x_j)^2,$$
$$\text{with} - 65 \leq x_i \leq 65,$$
$$min(f_3) = f_3(0, ..., 0) = 0.$$

- *Rastrigin's Function*

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

- *Griewangk's Function*

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

- *Sum of Different Power*

$$f_6(X) = \sum_{i=1}^{n} |x_i|^{(i+1)},$$
$$\text{with} - 1 \leq x_i \leq 1,$$
$$min(f_6) = f_6(0, ..., 0) = 0.$$

- *Ackley's Problem*

$$f_7(X) = -20\exp(-0.2\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}) - \exp(\frac{\sum_{i=1}^{n} \cos(2\pi x_i)}{n}) +$$
$$20 + e, \text{with} - 32 \leq x_i \leq 32,$$
$$min(f_7) = f_7(0, ..., 0) = 0.$$

- *Levy Function*

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

- *Michalewicz Function*

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

- *Zakharov Function*

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

- *Schwefel's Problem 2.22*

$$f_{11}(X) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|,$$
$$\text{with } - 10 \leq x_i \leq 10,$$
$$min(f_{11}) = f_{11}(0, ..., 0) = 0.$$

- *Step Function*

$$f_{12}(X) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2,$$
$$\text{with } - 100 \leq x_i \leq 100,$$
$$min(f_{12}) = f_{12}(-0.5 \leq x_i < 0.5) = 0.$$

- *Alpine Function*

$$f_{13}(X) = \sum_{i=1}^{n} |x_i \sin(x_i) + 0.1x_i|,$$
$$\text{with} - 10 \leq x_i \leq 10,$$
$$min(f_{13}) = f_{13}(0, ..., 0) = 0.$$

- *Exponential Problem*

$$f_{14}(X) = \exp(-0.5\sum_{i=1}^{n} x_i^2),$$
$$\text{with} - 1 \leq x_i \leq 1,$$
$$min(f_{14}) = f_{14}(0, ..., 0) = 1.$$

- *Salomon Problem*

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