Opposition-Based Differential Evolution with Protective Generation Jumping

Ali Esmailzadeh, *IEEE Member*
Faculty of Engineering and Applied Science
University of Ontario Institute of Technology (UOIT)
Oshawa, Ontario L1H 7K4
Email: ali.esmailzadeh@uoit.ca

Shahryar Rahnamayan, *IEEE Member*
Faculty of Engineering and Applied Science
University of Ontario Institute of Technology (UOIT)
Oshawa, Ontario L1H 7K4
Phone: (905) 721-8668 ext.3843
Fax: (905)-721-3178
Email: shahryar.rahnamayan@uoit.ca

***Abstract*—The Opposition-based Differential Evolution (ODE) algorithm has shown to be superior to its parent, Differential Evolution (DE) algorithm in solving many real-world problems and benchmark functions efficiently. An acceleration component of ODE, called *generation jumping*, is involved with creating *opposite* population and competing with current population, and from the union of those populations, selecting the N_p fittest individuals. The *jumping* is triggered based on a constant percentage (i.e., *jumping rate*) during search process. There are optimization problems in which *generation jumping* is not useful and only wastes computation time and resources. In this paper, we focus on those certain benchmark functions which ODE performs poorly because of the useless generation jumping, and we introduce *Opposition-Based Differential Evolution with Protective Generation Jumping* (ODEPGJ), in which it makes the ODE algorithm more *adaptive* in term of generation jumping. In fact, we stop generation jumping when it seems to be unhelpful in acceleration process. The experimental verifications are provided to show the improvement caused due to the mentioned protective generation jumping.**

I. Introduction

The p-metaheuristic algorithms have at least one thing in common, they all initialize and evolve populations of candidate-solutions to solve a problem. By having more than one candidate, specially for problems with difficult landscapes, it is beneficial to use population-based metaheuristic so that the search process does not get trapped easily in local optima. There have been different schemes and algorithms introduced over the years that increase the diversity of initial population. However, with any attempts to increase the diversity, there are overhead costs involved. As in most cases with computational algorithms, the added overhead is in terms of increased computational time and storage. Even with great advances in computer hardware, computational time is still main concern for designed algorithms. For solving large-scale problems, even with a modern computer equipped with latest hardware, a single simulation could take a long time to terminate.

This is the motivation for researchers to make existing algorithms faster and more efficient. One of the ways to achieve this is to make an algorithm *adaptive*. For example, the algorithm will execute the steps defined in the algorithm until it realizes that certain steps are no longer useful. This ability of an algorithm to adapt to its current search process situation has the potential to save on processing time and storage. This is an attractive direction for making algorithms smarter and more efficient.

In the recent years, a great amount of research has been conducted in improving population-based metaheuristic algorithms in term of convergence speed. The Differential Evolution (DE) algorithm is a population-based metaheuristic evolutionary algorithm. DE is an efficient algorithm in solving global optimization problems, but it has problems with, as well, it is prone to *curse of dimensionality*. There has been improvements done to improve the performance and efficiency of DE. Das et al. have proposed a change to the mutation step of DE using Neighborhood-Based concept [2]. In this work, the authors attempt to balance the exploration and exploitation of the algorithm by using a self-adaptive neighborhood-based mutation.

In research work by Zhang and Sanderson in 2009, the *current-to-pbest* mutation scheme is used, along with adaptive control of crossover and mutation parameters [3]. In this algorithm, called *JADE*, a historical data about past candidate-solutions is kept and utilized to guide the search. JADE has outperformed other evolutionary algorithms, particularly large-scale problems [3].

Another improvement done to the original version of DE is called *Opposition-based Differential Evolution* (ODE), introduced by Rahnamayan et al. [10], which applies the concept of Opposition-Based Learning to accelerate the DE algorithm. In ODE, the algorithm includes *generation jumping* which creates opposite population to compete with the current population. In ODE, the frequency of generation jumping is set with a jumping rate parameter, J_r. The J_r parameter is set to a constant value for the entire search process. Generation jumping, utilized in ODE, is not an exception in regards to computational overhead. It increases diversity of the current population but needs extra function calls.

The opposition concept has been applied to other p-metaheuristic algorithms, such as, Particle Swarm Optimization (PSO), called Opposition-Based Particle Swarm Optimization (OPSO), by Han et al. [1]. In OPSO the opposite of the velocity and position of a particle are taken into consideration. Similar to ODE, in OPSO the generation jumping is used, and J_r value is set to a constant value for the entire search.

The opposition concept was also utilized in Biogeography-Based Optimization (BBO) algorithm, entitled *Oppositional biogeography-based optimization*, by Ergezer et al. [4]. In this research work, population *jumping*, J_r, is used to promote generation of opposite population.

Rahnamayan et al. introduced a modification to ODE in 2007 by having *variable* jumping rate [5]. In that paper, the jumping rate value is linearly changed based on the number of function evaluations. The higher jumping rate value was used at the beginning of the search during exploration.

The generation jumping of ODE does not always have a positive effect, hence, generation jumping (and opposition) has to be used when it is useful. In this paper, we will examine the J_r parameter of the ODE algorithm, which is a constant value throughout the search. We intend to make generation jumping process of ODE *protective*, such that, when generation jumping is no longer useful in the search process, it is stopped (i.e., J_r=0), and hence, saves on computation costs.

The rest of paper is organized as follows: in Section II, the classical Differential Evolution (DE) and Opposition-based Differential Evolution (ODE) are briefly reviewed. In Section III, modification to the scheme of generation jumping in ODE which builds our proposed approach, is explained. The experimental results and analysis are given in Section IV. Finally, the work is concluded in Section V.

II. A Short Review of Classical and Opposition-Based Differential Evolution

The classical Differential Evolution algorithm and the Opposition-based Differential Evolution are briefly reviewed because of their relevance to the proposed algorithm in this paper.

A. 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 generated randomly 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 one.

For classical DE (i.e., $DE/rand/1/bin$), 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 a higher diversity in the generated population and lower values cause a 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 constant crossover rate, 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 $(0.4, 1)$ [13].

Selection - The scheme that must decide which vector ($U_{i,G}$ or $X_{i,G}$) should be a member of the next generation, $G + 1$. For a maximization problem, the vector with a higher fitness value is chosen. There are other variants based on different mutation and crossover strategies [8].

B. Opposition-based Differential Evolution

Before explaining ODE, we need to give a brief introduction to the origin of the *Opposition-Based Learning (OBL)* concept, which is the corner-stone of all the opposition-based computations. OBL was introduced by Tizhoosh in 2005 [9]. The main concept behind OBL is to consider the opposite of an assumption or a guess. That is, by taking into account the opposite of an assumption, and compare it with the original assumption, we can increase our chances to find solution faster. Of course the concept proposed in the paper [9] was a general, high-level concept which can be utilized in specific optimization algorithms.

The concept of OBL is attractive when applied to optimization algorithms, specifically, p-metaheuristic algorithms. Since p-metaheuristic algorithms deal with a population of randomly generated candidate-solutions, creating another population of opposite-candidates increases the diversity of the population. It is interesting to see that the concept of opposition helps solving the problem more efficiently.

The opposition concept was applied and tested on the classical DE algorithm by Rahnamayan et al. [10], called

Opposition-Based Differential Evolution (ODE). The DE algorithm, as previously mentioned is an evolutionary, population-based algorithm, which, similar to other evolutionary algorithms, an initial population of candidate-solutions is generated uniform randomly. The random population generation can play a role in terms of convergence speed of an algorithm since the distance of the candidates from the unknown solution determines how fast an optimal solution can be found. In the ODE algorithm, the concept of opposition has been used to decrease the distance from an unknown solution by comparing the candidate solution with its opposite and continuing with the better one. The formula for calculating the opposite point is as follows [9]:

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

The Fig. 1 illustrates the candidate-solution, x, and its corresponding opposite, $\hat{x}$, in interval [a,b].

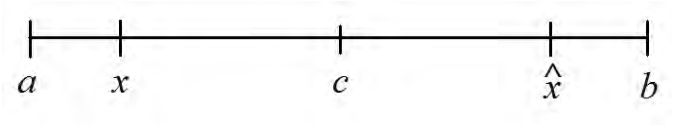


Fig. 1. The visual presentation (in 1D) of uniform-random candidate-solution, x, and the opposite candidate-solution, $\hat{x}$, in the interval [a,b], where c indicates corresponding center of the search space, c=(x+$\hat{x}$)/2.

Furthermore, for higher dimensions, the definition can be extended as follows [9]:

Let $P = (x_1, x_2, ..., x_D)$ be a point in D-dimensional space, where $x_1, x_2, ..., x_D \in R$ and $x_i \in [a_i, b_i]$ $\forall i \in \{1, 2, ..., D\}$. The opposite point $\hat{P} = (\hat{x}_1, \hat{x}_2, ..., \hat{x}_D)$ is defined by its components as follow:

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

In steps of DE where a uniform-random population is generated, the opposite of each individual (i.e., candidate-solution) is calculated and populated. Then, from union of the current population and the opposite one the fittest individuals are selected. Therefore, the current population includes uniform-random and opposite individuals with the better fitness values. Another additional modification to DE, by ODE is the generation jumping based on a new checking variable called generation jumping rate, J_r, which is constant value in the entire run, and determines opposition-based generation jumping rate. The flowchart of ODE is presented in Fig. 2.

The simulation results of different test cases on well-known benchmark functions done in [10], show that in overall, ODE outperforms DE; the ODE algorithm is about 60% faster than DE.

III. Proposed Algorithm: ODE With Protective Generation Jumping

As mentioned previously, ODE uses a constant amount for Jumping Rate (J_r). This component of ODE creates more

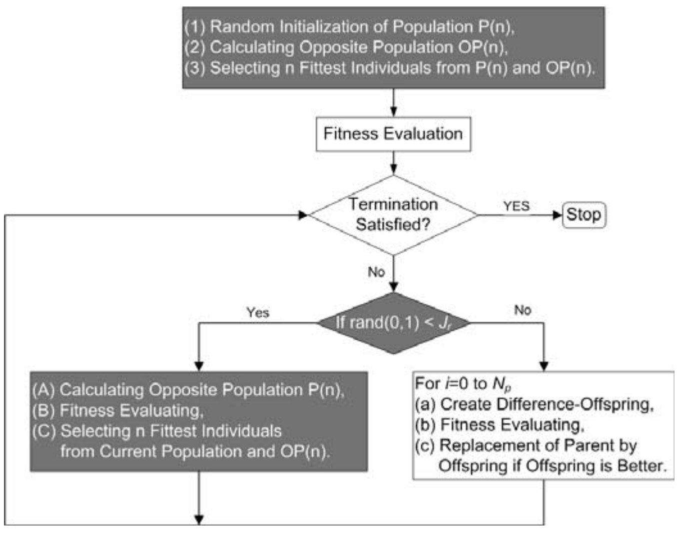

(1) Random Initialization of Population P(n),
(2) Calculating Opposite Population OP(n),
(3) Selecting n Fittest Individuals from P(n) and OP(n).
Fitness Evaluation
Termination Satisfied?
YES
Stop
No
If rand(0,1) < J_r
Yes
No
(A) Calculating Opposite Population P(n),
(B) Fitness Evaluating,
(C) Selecting n Fittest Individuals from Current Population and OP(n).
For i=0 to N_p
(a) Create Difference-Offspring,
(b) Fitness Evaluating,
(c) Replacement of Parent by Offspring if Offspring is Better.

Fig. 2. The flowchart presentation of ODE; the gray boxes represent the components which are added or modified in the classical DE algorithm [12].

opportunities for increasing diversity and continuing with fitter individuals by allowing the current population to be compared with the opposite population and selecting the fittest individuals.

In the original scheme of ODE, the J_r value is set to a constant value, and is chosen based on experimental efforts. The opposition-based jumping is beneficial when the generated opposite-points are partially better than the current points. Otherwise, there is a wasted computational cost because of the cost involved in calculating opposite individuals and their evaluations (extra function calls).

Even though the purpose of generation jumping is to accelerate convergence rate, there are optimization problems which ODE does not converge before meeting of maximum number of function calls (Max_NFC) because of the useless jumpings. In [10], authors have compared ODE with DE in terms of Success Rate (SR) and Number of Function Calls (NFC). They showed that for some benchmark functions ODE presents a poor success rate. Even though, for the most of 58 test functions [10], ODE outperforms DE in terms of SR and NFC, the success rate is lower than that of DE for some functions, such as f_4, f_5, f_6, f_{17}, f_{18}, f_{22}, f_{25}, f_{49}, f_{50}, f_{51}, f_{55}, and f_{57}.

In the proposed scheme, we intend to manipulate the generation jumping from a constant rate (as in original ODE) to constant rate but adaptive one. As the candidate-solutions converge to the solution, the opposite candidates do not make a significant difference in the search process; therefore, the process of generating opposite-population is an unnecessary step that does not help improving the convergence rate and diversity. As a result, it wastes processing resources and time.

In this paper, we allow the J_r value to be a constant value throughout the search process just as in ODE; but, we want to stop the generation jumping process when it is not helping the convergence. The idea is to keep track of when generation jumping (i.e., generating opposite-population) is no longer helping the current population. We call this modified version of ODE as *Opposition-based Differential Evolution with Protective Generation Jumping (ODEPGJ)*. In ODEPGJ, we define two parameters in order to determine when to stop the generation jumping of ODE, when it is deemed useless. We need to consider that, during jumping, what percentage of opposite-population has been selected in the current population (from set of P $\cup$ OP, N_p fitter individuals are selected). When the opposite-population is less than a certain percentage of the current population, we consider the generation jumping ineffective and it is better to follow the original DE steps and stop opposition-based jumping. We call this parameter *Cut-off Threshold (CT)*. Since ODE algorithm is stochastic, even in one generation jumping where CT is less than the set amount, does not necessarily indicate that generation jumping is useless in the rest of the search. Therefore, we need to consider consecutive unsuccessful generation jumpings with CT of less than the set amount, in order to consider more than one sample. That requires us to define another parameter for ODEPGJ to consider how many consecutive CT values of less than set amount is allowed before jumping is stopped. We call this parameter *Cut-off Threshold Frequency (CTF)*. The value set for CTF will restrict the *consecutive* instances for which CT value can be less than a desired amount. By setting CT and CTF parameters, if the generation jumping meets the CT value for number of consecutive times set by CTF parameter, then the generation jumping is stopped for the rest of the search (i.e., J_r=0). By this way, the regular DE steps are performed for each generation without any instance of creating opposite-population; hence, it will save on computation cost and time for the rest of the search. In fact, the proposed scheme turns off the generation jumping after *CTF* consecutive unsuccessful jumps. The ODEPGJ algorithm is presented in Fig. 3. From comparing ODE algorithm (Fig. 2) and ODEPGJ algorithm, the decision step is added to the generation jumping step, in which if the percentage of opposite-individuals in current population is less than CT value, the *counter* variable is incremented by 1. At the next generation jumping step, if the *counter* value is equal to the CTF value, then generation jumping is stopped for the rest of the search.

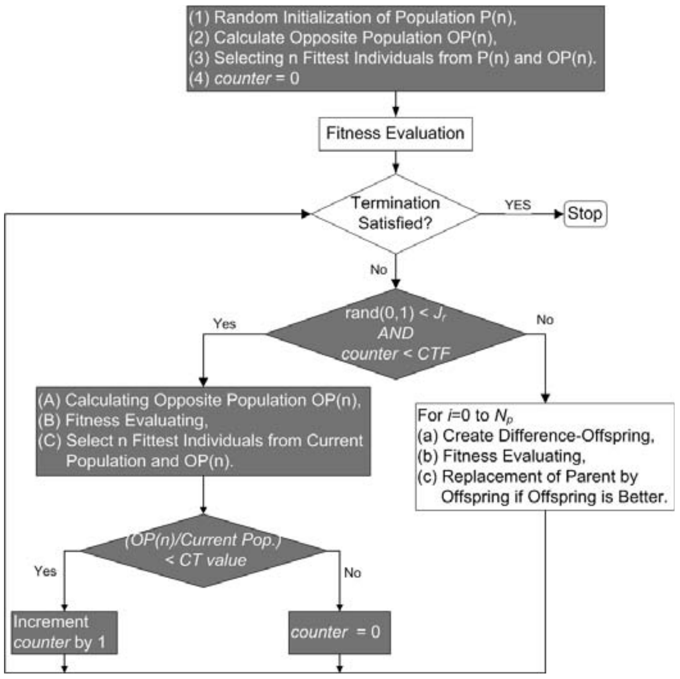
As an example, let us assume that for a given run, $CT = 10\%$ and $CTF = 5$. In ODEPGJ algorithm, when there is a generation jumping, and the percentage of OP candidates in the current population is less than CT value (i.e., 10%), ODEPGJ will start counting the number of consecutive jumps in which CT is less than 10%. If the *counter* value is equal to CTF value (in this case, 5), then a flag is raised and generation jumping of ODE is deactivated for the rest of the search.

IV. Experimental Verifications

A. General Control Parameter Settings

All the common parameters for these simulations, defined below, are set to the same values as in [10], in order to have the similar and fair experiments as the reference.

- Population size, $N_p = 100$



(1) Random Initialization of Population P(n),
(2) Calculate Opposite Population OP(n),
(3) Selecting n Fittest Individuals from P(n) and OP(n),
(4) counter = 0
Fitness Evaluation
Termination Satisfied?
YES
Stop
No
rand(0,1) < J_r AND counter < CTF
Yes
No
(A) Calculating Opposite Population OP(n),
(B) Fitness Evaluating,
(C) Select n Fittest Individuals from Current Population and OP(n).
For i=0 to N_p
(a) Create Difference-Offspring,
(b) Fitness Evaluating,
(c) Replacement of Parent by Offspring if Offspring is Better.
(OP(n)/Current Pop.) < CT value
Yes
No
Increment counter by 1
counter = 0

Fig. 3. The flowchart presentation of ODEPGJ, where *counter* represents the number of jumps in which OP percentage in the current population is less than CT value; the gray boxes represent the components which are modified or added in the classical DE algorithm.

- Differential amplification factor, F=0.5
- Crossover probability constant, $C_r = 0.9$
- Jumping Rate value, $J_r = 0.3$
- Cut-off Threshold, $CT = 10\%$
- Cut-off Threshold Frequency , $CTF = 5$
- Strategy, $DE/rand/1/bin$
- Value-to-reach, VTR=10^{-8}
- Maximum number of function calls (termination criteria), MAX_{NFC}=$1,000,000$

B. Benchmark Functions

The following benchmark functions are the functions used in the ODE paper [10] to investigate the improvement of ODE comparing to classical DE, for different schemes of jumping rates. The following benchmark functions taken for this paper are those which have a Success Rate (SR) of less than 1 in [10].

- Rosenbrock's valley

$$f_4(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2],$$
$$\text{with} -2 \leq x_i \leq 2$$
$$min(f_4) = f_4(1, ..., 1) = 0.$$

- Rastrigin's function

$$f_5(x) = 10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i)),$$
$$\text{with} - 5.12 \leq x_i \leq 5.12$$
$$min(f_5) = f_5(0, ..., 0) = 0.$$

- Griewangk's function

$$f_6(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1,$$
$$\text{with} - 600 \leq x_i \leq 600$$
$$min(f_6) = f_6(0, ..., 0) = 0.$$

- Perm function

$$f_{17}(x) = \sum_{k=1}^{n} [\sum_{i=1}^{n} (i^k + 0.5)((\frac{1}{i}x_i)^k - 1)]^2,$$
$$\text{with} - n \leq x_i \leq n$$
$$min(f_{17}) = f_{17}(1, 2, 3, ..., n) = 0.$$

- Michalewicz function

$$f_{18}(x) = -\sum_{i=1}^{n} \sin(x_i)(sin(ix_i^2/\pi))^{2m},$$
$$\text{with } 0 \leq x_i \leq \pi, m = 10$$
$$min(f_{18_{(n=2)}}) = -1.8013,$$
$$min(f_{18_{(n=5)}}) = -4.687658,$$
$$min(f_{18_{(n=10)}}) = -9.66015.$$

- Schwefel's problem 2.21

$$f_{22}(x) = \max_i \{|x_i|, 1 \leq i \leq n\},$$
$$\text{with } - 100 \leq x_1 \leq 100$$
$$min(f_{22}) = f_{22}(0, ..., 0) = 0.$$

- Kowalik's function

$$f_{25}(x) = \sum_{i=1}^{11} [a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}]^2,$$
$$\text{with } - 5 \leq x_i \leq 5$$
$$min(f_{25}) = f_{25}(0.19, 0.19, 0.12, 0.14) = 0.0003075.$$

where:

$$a = \begin{bmatrix} 0.1957 & 0.1947 & 0.1735 & 0.1600 & 0.0844 \\ 0.0627 & 0.0456 & 0.0342 & 0.0323 & 0.0235 \\ 0.0246 & & & & \end{bmatrix}_{1\times 11},$$
$$b^{-1} = \begin{bmatrix} 0.25 & 0.5 & 1 & 2 & 4 & 6 & 8 & 10 & 12 & 14 & 16 \end{bmatrix}$$

- f_{49} - Multi-Gaussian Problem
- f_{50} - Neumaier 2 Problem
- f_{51} - Odd Square Problem
- f_{55} - Price's Transistor Modeling Problem
- f_{57} - Schaffer 2 Problem

C. Simulation Strategy

Similar to other studies in the evolutionary optimization [11]–[14], for all conducted experiments, trials are repeated 25 times per function.

D. Simulation Results

The results presented in this paper compare the ODEPGJ algorithm with the original ODE algorithm, in terms of Number of Function Calls (NFC) and Success Rate (SR) values. The summary of the results are given in Table I.

Furthermore, for some of the results, there are graphs which represent the percentage of opposite-individuals in the current population vs. NFC, at each generation jumping instance. The graphs are all presented in Fig.s 4 and 5.

TABLE I
THE NUMBER OF FUNCTION CALLS (NFC) AND SUCCESS RATE (SR) OF SELECTED BENCHMARK FUNCTIONS FOR ODE AND ODEPGJ. THE **BOLD** VALUES REPRESENT IMPROVED (AND BEST) VALUES IN EACH FUNCTION.

		ODE		**ODEPGJ**	
F	D	NFC	SR	NFC	SR
f_4	30	1000000	0	**695250**	**0.04**
f_5	10	**69543**	0.84	146200	0.84
f_6	30	**69504**	0.96	86483	0.96
f_{17}	4	536500	0.04	**435070**	**0.08**
f_{18}	10	250230	**0.56**	**189175**	0.48
f_{22}	30	72250	0.88	**72188**	**1**
f_{25}	4	**15140**	0.60	17253	**0.76**
f_{49}	2	5996	0.96	**5884**	**1**
f_{50}	4	**383180**	0.16	443904	**1**
f_{51}	10	18352	0.84	**16662**	0.84
f_{55}	9	184360	0.40	**158950**	0.40
f_{57}	2	55980	0.76	**7005**	**0.84**
Avg.		221753	0.59	**189502**	**0.69**

E. Results Analysis

As seen in Table I, for functions f_5, f_6, f_{51}, and f_{55}, the SR values between ODE and ODEPGJ are remained unchanged, and the number of function evaluations for f_5 and f_6 are less for ODE. This implies that using the cut-off generation jumping did not improve solving the functions f_5 and f_6 in respect to NFC and SR values. ODEPGJ algorithm did improve the majority of functions in terms of NFC or SR value, or both. Overall, as seen in Table I, ODEPGJ outperformed ODE on 10 out of 12 functions tested, in terms of either NFC or SR value, or both. The main improvements, are observed on f_4, f_{22}, f_{49}, and f_{57}. In all those four functions, the NFC and SR values have been improved significantly. The average NFC and SR values of ODEPGJ show improvement of 15% and 10%, respectively, compared to ODE.

The visualization of the protective jumping mechanism is presented in Fig.s 4 and 5. These figures demonstrate the percentage of opposite individuals which are selected to be in the current population at each generation jumping. In the figures, each graph is the representation of the percentage of opposite individuals (OP) in the current population (P) vs. the NFC value after each generation jumping. The purpose of the graphs are to show an approximate indication of when the generation jumping is stopped. An important thing to note is that since the simulations and algorithm of DE and ODE are stochastic, at each run the values for NFC might vary; therefore, it is not possible to match the exact NFC values from ODE graph to that of ODEPGJ graph, for each function, since they are run separately.

In Fig. 4 (b), which represents the function f_{17} for ODE run, the percentage of OP in current population goes below 10% (i.e., the CT value set for experiments of this paper) roughly around 6,200 NFCs, while in Fig. 4 (a) (for ODEPGJ), the generation jumping is stopped at around the 5,900 NFCs.

On the other hand, function f_{22}, represented in Fig. 4 (c) and (d), is the only instance where there is no need to stop jumping, and the graph confirms that it has not been stopped correctly. We can observe that the ODE graph (d) has no sections where the percentage of opposite individuals go below CT value (10%). Therefore, the ODEPGJ for f_{22} (graph (c)), has no stoppage of jumping. For function f_{49} represented in Fig. 4 (e)(f), ODE representation shown in graph (f), the percentage of opposite-individuals is below 10% for 5 consecutive jumps roughly around 3,100 NFCs. The ODEPGJ algorithm stops generation jumping at around 3,000 NFCs in graph (e). According to Table I, for function f_{49}, ODEPGJ has a higher SR and lower NFCs.

Furthermore, for function f_{55} presented in Fig. 5 (c) and (d), we can observe that, the percentage of opposite individuals in current population drops below 10% roughly around 6,700 NFCs. Hence, in ODEPGJ in function f_{55} (graph (c)), generation jumping is stopped at roughly around 6,300 NFCs. By observing Table I, the SR value for both ODE and ODEPGJ algorithms is the same (0.40) for f_{55}; however, the NFC value of the function using ODEPGJ algorithm is considerably less than that for ODE algorithm. This indicates that the problem was solved with less function calls.

Implementing ODEPGJ saves overhead computational resources and cost of keeping generating opposite-population when for the rest of the search process no reasonable portion of individuals from OP is selected for the current population.

In overall, ODE is outperformed by ODEPGJ, in terms of NFCs and SR rate.

V. Concluding Remarks

The Opposition-based Differential Evolution (ODE) has been successful, in outperforming classical Differential Evolution (DE) in terms of solving problems with less function evaluations. One key component of ODE is the *generation jumping* which is based on a constantly-set value, J_r. It creates opposite-population from the current population, and from union of both current and opposite populations, selects N_p fittest individuals to continue. The rate at which generation jumping occurs is a constant value. However, each generation jumping involves computational cost and resources. For some benchmark functions, the generation jumping is useless. In this paper, we improved ODE by controlling the *generation jumping* based on the ratio of survived individuals from the opposite population. More specifically, by noticing when generation jumping is not useful anymore, we stop the jumping process for the rest of the search, based on two parameters introduced in this paper: *Cut-off Threshold (CT)* and *Cut-off Threshold Frequency (CTF)*. We call this enhanced ODE as *Opposition-based Differential Evolution with Protective Generation Jumping*, ODEPGJ. This algorithm was tested on 12 benchmark functions which previously showed to have poor success rate during solving by ODE. The results have shown that for 10 out of 12 functions, ODEPGJ has improved either number of function calls or success rate, or both.

As a future work, smarter jumping control mechanisms can be investigated. A smarter ODEPGJ can in fact turn generation jumping on and off throughout the search, based on new parameters depending on the usefulness of jumping after the temporary stoppage of jumping.

References

[1] L. Han, X. He, *A Novel Opposition-Based Particle Swarm Optimization for Noisy Problems*, Third International Conference on Natural Computation, 2007 (ICNC'07), Haikou, Vol. 3, pp. 624-629, 2007.

[2] S. Das, A. Abraham, U.K. Chakraborty, A. Konar, *Differential Evolution Using a Neighborhood-Based Mutation Operator*, IEEE Transactions On Evolutionary Computation, Vol. 13, No. 3, pp. 526-553, June 2009.

[3] J. Zhang, A.C. Sanderson, *JADE: Adaptive Differential Evolution with Optional External Archive*, IEEE Transactions On Evolutionary Computation, Vol. 13, No. 5, pp. 945-958, Oct. 2009.

[4] M. Ergezer, D. Simon, D. Du, *Oppositional Biogeography-Based Optimization*, IEEE International Conference on Systems, Man and Cybernetics (SMC'09), San Antonio, TX, pp. 1009-1014 , 2009.

[5] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, *Opposition-Based Differential Evolution (ODE) with Variable Jumping Rate*, IEEE Symposium on Foundations of Computational Intelligence (FOCI'07), Honolulu, HI, Vol. III, pp. 81-88, 2007.

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

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

[8] 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.

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

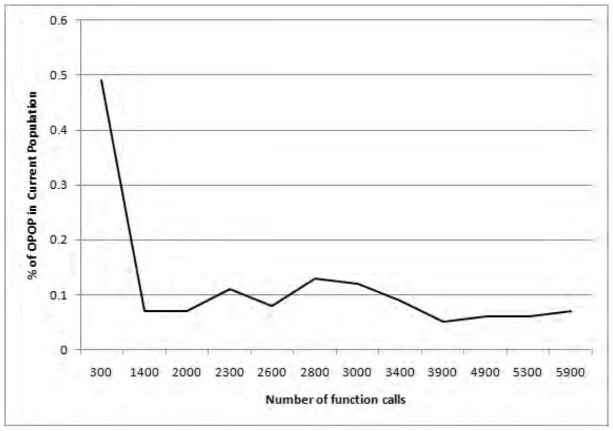
[10] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, *Opposition-Based Differential Evolution*, Proceedings of the IEEE Transactions On Evolutionary Computation, Vol. 12, No. 1, pp. 64-79, February 2008.

[11] J. Vesterstrøm 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'04), IEEE Publications, Vol. 2, pp. 1980-1987, 2004.

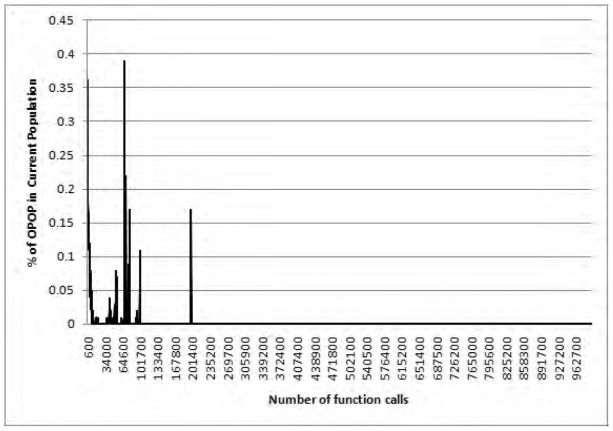
[12] S. Rahnamayan, H. R. Tizhoosh, M. M.A. Salama, *Opposition-Based Differential Evolution for Optimization of Noisy Problems* Proceedings of the Congress on Evolutionary Computation (CEC'06), IEEE Publications, pp. 6756-6763, 2006.

[13] S. Das, A. Konar, Uday K. Chakraborty, *Improved Differential Evolution Algorithms for Handling Noisy Optimization Problems*, Proceedings of IEEE Congress on Evolutionary Computation, CEC2005, Vol. 2, pp. 1691-1698, 2005.

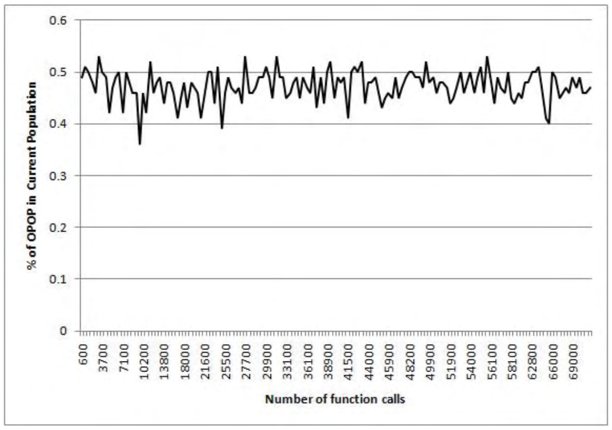
[14] T. Krink, B. Filipič, Gary B. Fogel, *Noisy optimization problems - A Particular Challenge for Differential Evolution?*, Proceedings of the 2004 Congress on Evolutionary Computation, CEC2004, Vol. 1, pp. 332-339, 2004.



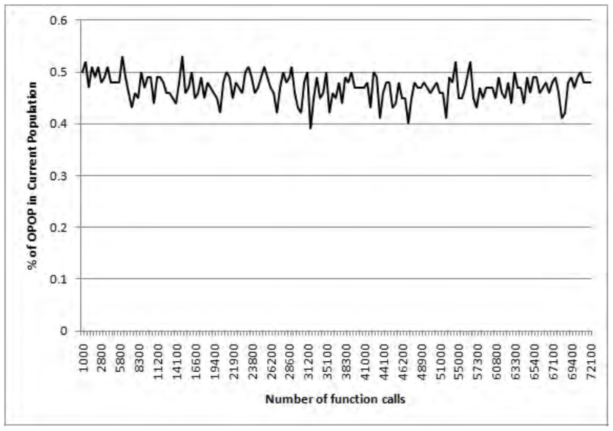
(a) f_{17}(4D), Perm Function - by ODEPGJ



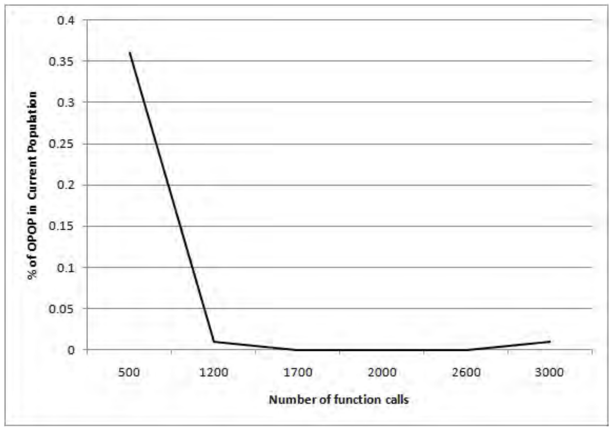
(b) f_{17}(4D), Perm Function - by ODE



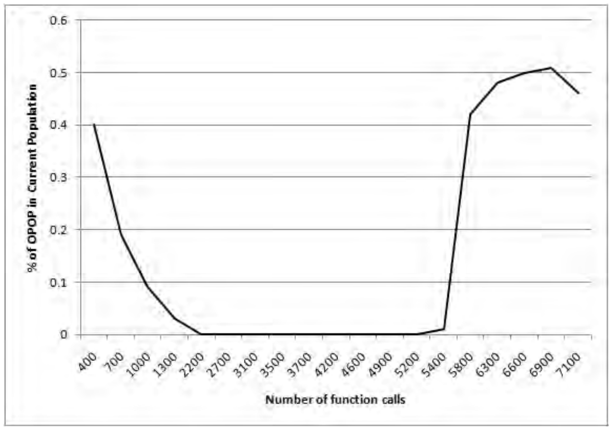
(c) f_{22}(30D), Schwefel's problem 2.21 - by ODEPGJ



(d) f_{22}(30D), Schwefel's problem 2.21 - by ODE

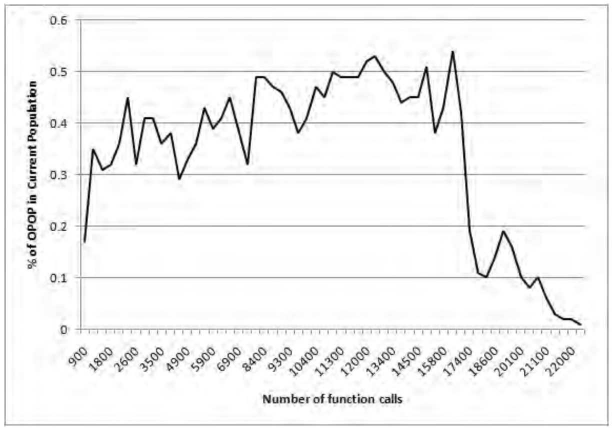


(e) f_{49}(2D), Multi-Gaussian Problem - by ODEPGJ

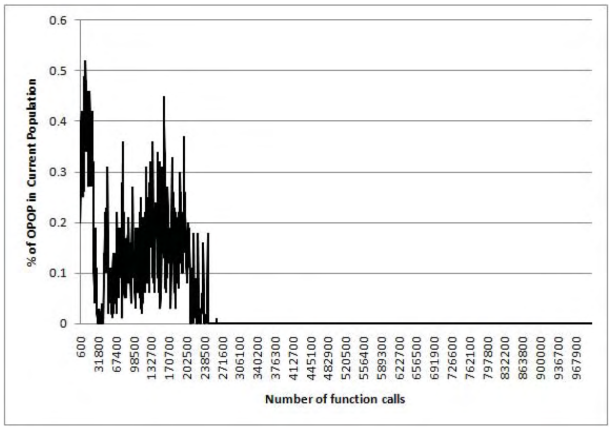


(f) f_{49}(2D), Multi-Gaussian Problem - by ODE

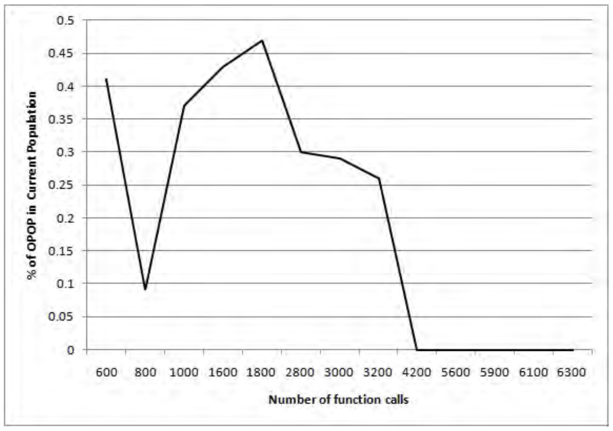
Fig. 4. Some sample graphs to show the protective jumping mechanism. The graphs for f_{17}, f_{22}, f_{49}, indicating the percentage of opposite individuals in the current population vs. number of function calls (NFC) at each generation jumping, for ODE (the graphs on the right) and ODEPGJ (the graphs on the left). Each pair of graphs for a given function show the jumping stoppage in the ODEPGJ from its corresponding ODE graph. The jumping of ODE on f_{17} is stopped around 5,900 NFC (graph (a)) using ODEPGJ; jumping of ODE on f_{22} is not stopped for the entire search; jumping of ODE on f_{49} is stopped around 3,000 NFC (graph (e)) using ODEPGJ.



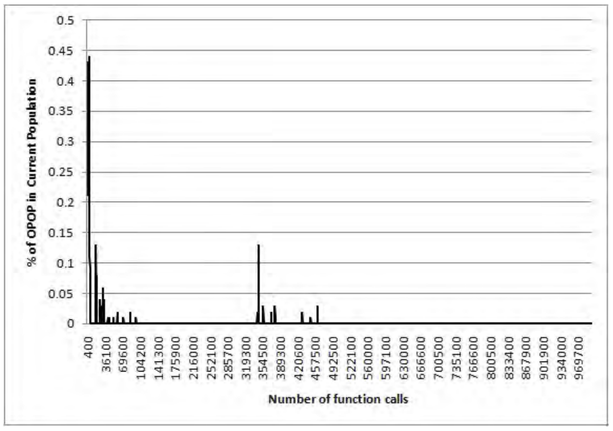
(a) f_{51}(10D), Odd Square Problem - by ODEPGJ



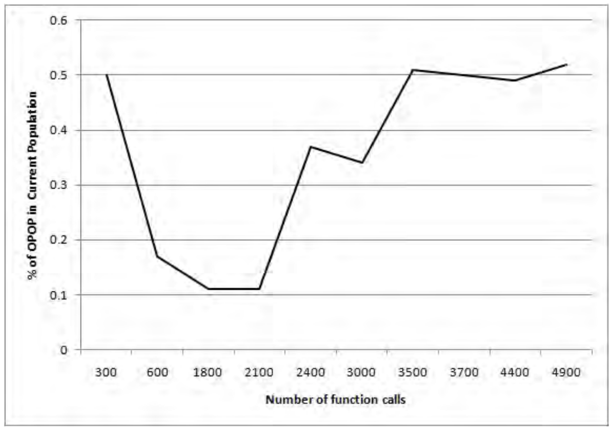
(b) f_{51}(10D), Odd Square Problem - by ODE



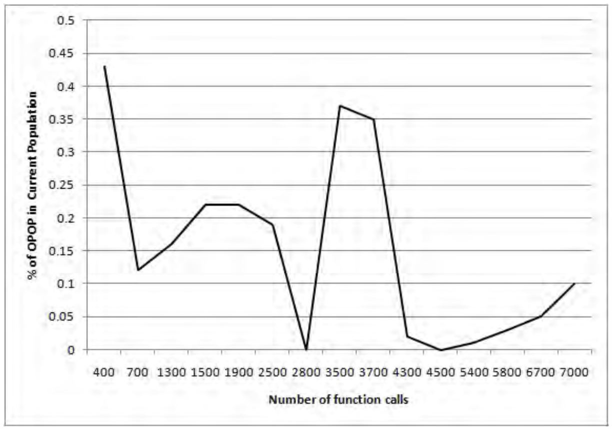
(c) f_{55}(9D), Price's Transistor Modeling Problem - by ODEPGJ



(d) f_{55}(9D), Price's Transistor Modeling Problem - by ODE



(e) f_{57}(2D), Schaffer 2 Problem - by ODEPGJ



(f) f_{57}(2D), Schaffer 2 Problem - by ODE

Fig. 5. Some sample graphs to show the protective jumping mechanism. The graphs for f_{51}, f_{55}, f_{57}, indicating the percentage of opposite individuals in the current population vs. number of function calls (NFC) at each generation jumping, for ODE (the graphs on the right) and ODEPGJ (the graphs on the left). Each pair of graphs for a given function show the jumping stoppage in the ODEPGJ from its corresponding ODE graph. The jumping of ODE on f_{51} is stopped around 22,200 NFC (graph (a)) using ODEPGJ; jumping of ODE on f_{55} is stopped around 6,300 NFC (graph (c)) using ODEPGJ; jumping of ODE on f_{57} is stopped around 4,900 NFC (graph (e)) using ODEPGJ.