International Journal of Innovative
Computing, Information and Control
Volume **7**, Number **7(B)**, July **2011**

ICIC International ©2011 ISSN 1349-4198
pp. **4263–4276**

# A NOVEL FUNCTION OPTIMIZATION APPROACH USING OPPOSITION BASED GENETIC ALGORITHM WITH GENE EXCITATION

Muhammad Amjad Iqbal[1,2,*], Naveed Kazim Khan[2], Hasan Mujtaba[2] and A. Rauf Baig[2]

[1]Faculty of Information Technology
University of Central Punjab
1 - Khayaban-e-Jinnah Road, Johar Town, Lahore, Pakistan
[*]Corresponding author: amjad.iqbal@ucp.edu.pk

[2]Department of Computer Science
National University of Computer and Emerging Sciences
A.K. Brohi Road, H-11/4, Islamabad, Pakistan
{ amjad.iqbal; naveed.kazim; hasan.mujtaba; rauf.baig }@nu.edu.pk

Received October 2009; revised February 2010

Abstract. *Nonlinear Complex optimization problems are the key area of research in the field of optimization. Evolutionary Algorithms (EAs) are applied to solve these optimization problems successfully. The EAs suffer a lot due to their slow convergence rate, primarily due to evolutionary nature of these algorithms. It has been proved that distribution of initial population into the search space effects the evolutionary algorithm performance. This paper presents a novel initialization method for genetic algorithms, in which opposite of the population is created. The best individuals from the population and its opposite are selected as the initial population. This provides a better starting point for search through the solution space. To increase the convergence speed of EAs, a probabilistic excitation scheme for chromosomes is also introduced. This scheme tunes the population effectively during the evolutionary process. The performance of the algorithm is tested over suit of 10 functions with different dimensions. Opposition based Differential Evolution and Genetic Algorithms are used as competitor algorithms to compare the results of the proposed algorithm. Various sets of experiments are performed. The results show that the proposed method outperforms Opposition based Differential Evolution and Genetic algorithms for most of the test functions.*

**Keywords:** Genetic algorithms, Opposition based learning, Convergence speed, Excitation rate

1. **Introduction.** Evolutionary Algorithms are applied to the optimization problems successfully. The Genetic Algorithm (GA), one of the major techniques of EAs, was proposed by J. Holland in 1970s [11]. The GA is simple and robust enough to be applied to optimization problems. Evolutionary process of the GA requires only a few initial parameters for its execution. And it has been proven to attain superior performance to other optimization techniques when it comes to the terms of convergence speed and robustness over many real world problems like [13-15] and benchmark problems [7-9]. Almost all of the population based optimization approaches like the GA, Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO), etc. have slow convergence rate due to the probabilistic/evolutionary nature of these algorithms. So, these algorithms can be an excellent candidate for the acceleration of their convergence rate. If the convergence speed is made fast enough, then these algorithms can be a preferable choice for use in situation with real time restrictions. Many remedies have been proposed to accelerate

the convergence rate of these algorithms. One of the techniques used is Opposition-based Learning (OBL). OBL was established by Tizhoosh [3]. OBL has been used to speed up the reinforcement learning [3], differential evolution [4-6] and back-propagation neural networks [10]. The foundation idea of OBL is to spawn the initial random solutions and the opposite of the solutions for the better covering of the search space of the problem. Opposite $x'$ of a point $x$ is calculated by using the formula $x' = a+b-x$, where $a$ and $b$ are the upper and lower value of the domain of $x$. After that, the best solutions are selected from initial population and its opposite and these solutions are used for the evolutionary searching process. We also integrated this idea of OBL to speed up the convergence rate of the GA. In our algorithm, initial population is selected from the initial random solutions and its opposite population. During the evolutionary process of the algorithm, some of solutions are selected probabilistically and opposite of these solutions are spawned. Then, best of the selected solutions and their opposites are selected as the part of next population. It has been stated in [6] where the random selection of solutions from a given search space can result in exploiting the infertile areas of the search space. The use of random population and its opposite, instead of only random selection, diminishes the chances of exploring barren areas of search space [6]. It is empirically proven that the use of opposite number is better than using random population or even using population with double number of solutions [6]. Population initialization and Excitation schemes are very simple and global that it can be integrated in other variations of Genetic Algorithm and in evolutionary Algorithms as well. This paper presents a set of experimental verifications of our proposed approach. Particularly, we investigate: 1) the convergence speed; 2) robustness; 3) the population size; and 4) the setting of a newly added control parameter, the excitation rate (defined in Section 3.4). For all these experiments, a set of ten well-known benchmark test functions is employed. Furthermore, in order to evaluate the algorithms, a set of performance measures has been utilized. These measures are the average number of function calls (NFCs), success rate (SR), average SR, number of unsolved problems, and the average of the best found solutions and the corresponding standard deviation as calculated after particular number of function evaluations has been reached. Organization of the remaining of this paper is as follows: the Genetic Algorithm and the concept of OBL are briefly explained in Section 2; the proposed algorithm and related terms are presented in Section 3; Section 4 consists of the performance measures and experimental results.

2. **Related Work.** The opposition based learning is relatively new area of research and it was first proposed by Tizhoosh [3] in 2005. Several opposition based learning algorithms have been suggested after that. This includes Reinforcement Learning [3], Back Propagation neural networks [10] and Differential Evolution [6]. Opposition based Differential Evolution [4] uses opposition based initialization scheme. In this method, opposite of initial population is created and best from either is chosen, and it is followed by the standard process of differential evolution. The actual process of Differential Evolution is augmented by the opposition phase. It has been proven that [6] OBL process increases the convergence speed and accelerates the evolutionary process. An additional method of opposition based Differential Evolution with jumping rate was proposed in [6]. Tizhoosh et al. suggested in this paper a constant jumping rate and the individuals in a population are allowed to jump once the probability is met. Best from either of the two was chosen as new population member. This facilitates faster convergence during the evolution phase and lesser fit population members have a chance to tune themselves. It is noted that the jumping rate has an effect of accelerating the convergence process until the population starts to converge. Once the population starts converging, it has rather unpleasant effect

as it permit individuals to jump away from the optimal solution. So, to counter this problem a method of using variable jumping rate was proposed in [5]. In this method, two variable jumping rates were proposed, one represented higher jumping rate during exploration and lower during exploiting phase. Other acted opposite to this. Both of them were used to analyze the effects of Generation jumping during the evolution. Opposition based learning had been explored using the differential evolution. Yet the idea needs to be explored using other evolutionary algorithms. It can also be applied to Swarm based algorithms. The idea of using opposition in genetic algorithm was also proposed in [3]. Genetic Algorithm has been used to tune the network controller for the complex electromechanical process [15]. Multiobjective version of the genetic algorithm is applied to solve this problem. This is also used to optimize Radial Basis Function Networks [16]. We have used the opposition based excitation of population for of GAs. GA can perform better in terms of efficiency, i.e., ability of finding the global optimum, and number of function evaluations, or vice versa [2]. Function optimization has been a task of interest to study behavior of evolutionary algorithms. Evolutionary algorithms tend to perform well for the optimization problem. We tackled function optimization problem to study the effectiveness, and comparison of Opposition based Genetic Algorithm with Classic Genetic Algorithm. Ten different functions were have been used for optimization and comparison of Genetic Algorithms and Opposition based Genetic Algorithm.

## 3. **Proposed Method.**

3.1. **Opposition-based learning and genetic algorithm.** The perception of OBL method is founded upon a common observation that, in actual life, people oppose one another. The strengths and weaknesses of these opposing ones are relative, i.e., the opposition of a weak person is strong relative to him. Similarly traits are often opposite to each other and in the same way opposition of a weak trait is usually strong one. In OBL method, it generate a second set of solutions which is opposite of the original solution set so that our probability of choosing better solutions can increase. In this paper, we propose an implementation strategy of opposition based learning method for genetic algorithm. In proposing this technique, we have introduced gene excitation as a fourth operator along with elitism, crossover and mutation. Gene excitation is used to elevate any precise solutions based on probability. Opposite of a number $x$ can be calculated using the following equation.

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

where $x \in R$ within an interval of $[a, b]$. In the case of multidimensional vector $x_i$, its opposite vector $x_i'$ will be calculated as:

$$x_i' = a_i + b_i - x \tag{2}$$

This opposite vector will be calculated using the boundaries $[a_i, b_i]$.

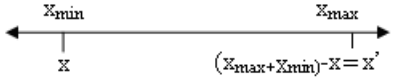


FIGURE 1. Calculating opposition of a point

Similarly, an opposite position in the search space can be defined as a point where all the dimensions of this point are replaced by their respective opposites. Any point in hyper dimensional space has a unique opposite point. The solution set for a given optimization problem encloses a multiple number of solutions. The population of genetic algorithm is

composed of chromosomes. A chromosome can be viewed as a potential solution among a set of solutions (set of chromosomes) for the given problem. A simple GA is explained in the following way.

1. Create a random population pool.
2. Initialize operator probabilities.
3. Repeat steps 4-6 until termination condition.
4. Evaluate fitness of each individual.
5. Perform Crossover based upon crossover probability.
6. Perform Mutation based upon mutation probability.

Create new population from children and parent population Opposite of a chromosome can be defined in a similar manner. Let our specific problem is to optimize a function $f(x)$. GA will use $f(x)$ as the fitness function to evaluate solution quality. $f(x)$ is a multi-dimensional minimization function. Let $C_i = (x_1, x_2, \ldots, x_n)$ be a chromosome from an n-dimensional space and $x_i \in [a_i, b_i]$ $\forall i \in 1, 2, 3, \ldots, n$. The opposite of this chromosome will be $C_i' = (x_1', x_2', \ldots, x_n')$. The boundary values $a_i$ and $b_i$ are calculated dynamically so that we are able to explore new regions while at the same time being able to contain when solutions seem to converge. Reproduction operators of GA have a tendency to expand the boundaries of search. Fitness function is the only mechanism to contain this expansion. Here, by calculating the boundary values dynamically, we have introduced a new mechanism in GA along with fitness function to contain the search space while at the same time enabling us to explore more regions. In this way, we shall continuously have a contracting solution space while being able to explore more. The step by step execution of opposition based GA is described in Figure 2. Brief description of main elements of this algorithm is given in following subsection.

3.2. **Chromosomes, opposite chromosomes and initialization block A.** As we stated previously, the GA has a population of chromosomes which work as solution set. A population of chromosome is used during the evolutionary process of this algorithm. Population is initialized by using the random initialization technique. Every chromosome is an $n$-dimensional real valued vector where $n$ represents the $n$-dimensions of the input vector used for the problem at hand. Each chromosome of the population is evaluated using the fitness function ($f(x)$) which is to be optimized. Afterward, opposites of these chromosomes are calculated using initial dimension boundaries. The method of calculating these opposites is elaborated in Section 3.1. So, each chromosome $C_i$ will have an opposite chromosome $C_{i,o}$. Each member of the opposite population is evaluated using the same function as used for actual population. After this process, we have a set of chromosomes and a set of opposite of chromosomes. Actual population and the opposite population are merged into a single population. Initial population for genetic algorithm is selected by choosing the m-best chromosomes of the merged population, where m is the population size for GA.

3.3. **Genetic algorithm block B.** At the completion of first stage described in Section 3.2, we have a set of best solutions. Reproduction operators like mutation, crossover and elitism are applied on this set of solutions. Fitness criterion is the value of function calculated using the chromosome. The reproduction operators are described as follows.

i **Elitism:** Most fit individual of the population is selected and included as an individual for the next generation of the algorithm. Selecting elite individual guarantees that solution quality will never degrade in next coming generations. In the proposed technique, we selected a single chromosome as the elite member.

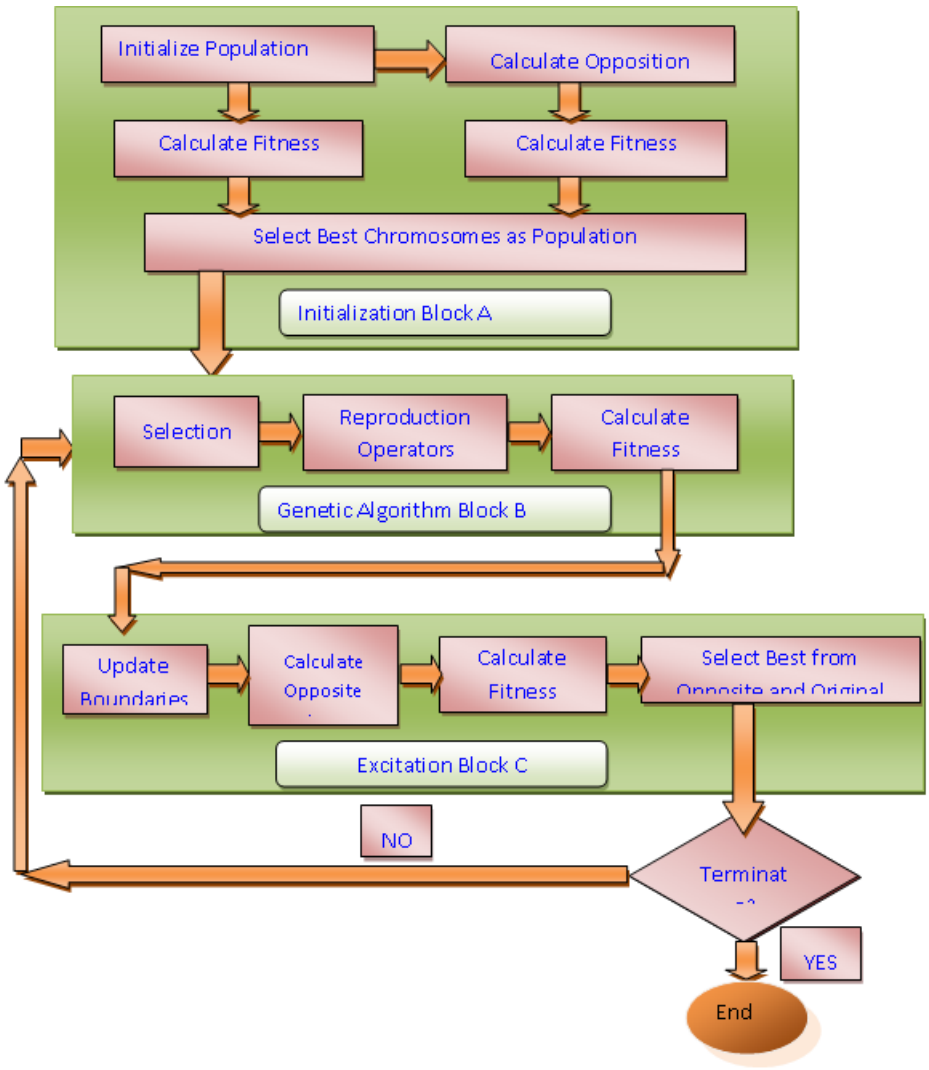


FIGURE 2. The proposed algorithm

ii **Crossover:** Crossover is the process of combining traits of two different individuals to generate solutions that can be of better fitness then parents. Two parent individuals are selected from the current population of the set to generate a new offspring. Number of offspring chromosomes is determined using the crossover probability. Roulette wheel selection is used for selecting the parent individuals. A two point crossover is used for performing this operator.

iii **Mutation:** Individuals are perturbed probabilistically to bring a change in the individuals. Crossover cannot introduce any new features since it merely combines the existing features in anew generation. Using mutation operator, there is a probability that some new features might appear due to change in the chromosome. Gaussian mutation is used to mutate the individuals. Mutation is performed on the basis of pre-determined mutating probability.

The above three operators are used together to form a population of new solutions. Next solution set is formed by selecting best individuals from the parent population and this new population.

3.4. **Excitation block C.** Dimension Boundaries are updated by calculating the maximum and minimum value of each dimension. For each of the individual in the population

a random value $r$ is generated. If the value $r$ is less than the excitation probability then opposite of the individual is calculated. This opposite individual is evaluated using the fitness function. Comparison of the fitness of original and opposite chromosome is made and the best is kept as member of next population. This process is executed for every member of the currently selected population. Instead of using the static dimension boundaries, the values in the interval $[a_i^s, b_i^s]$ are used to calculate the opposite of these chromosomes. To create the opposite of the population, we take the opposite $C_{o,i}$ of each member of the population chromosome $C_i$. The opposite point $C_{o,i}$ of the chromosome $C_l$ is calculated as below.

$$C_{o,i} = a_k^s + b_k^s - C_{i,k} \tag{3}$$

In above equation, $C_{i,k}$ represents the $k^{\text{th}}$ dimension of the $k^{\text{th}}$ chromosome $C_i$, and the terms $a_k^s$ and $a_k^s$ denote the maximum and minimum value of the $k^{\text{th}}$ dimension respectively over all the current population. The term $C_{oi,k}$ denotes the opposite-value of chromosome $C_i$ in the $k^{\text{th}}$ dimension.

4. **Experimental Results.** A suit of test functions including ten global optimization functions are used to assess the performance of Opposition based GA. The functions used for optimization process are mentioned in Table 9, along with their optimum values and dimensionality.

4.1. **Performance measures.** Following performance measures were used to assess the performance of GA and Opposition Based GA. These performance measures were used for comparison of ODE in [5,6].

4.1.1. *Number of function calls.* Genetic Algorithm (GA) and OGAE are compared by measuring Number of function calls (NFC). The termination criterion is to achieve a value smaller than the target value. Higher convergence speed is measured by having smaller number of NFCs. NFC is the most frequently used metric in literature. NFCs are averaged over 50 iterations for each of the global optimization function. So, for the comparison of the NFCs of the two algorithms Acceleration Rate (AR) [6] is calculated. AR is described by the following equations for the NFCs of GA and OGAE.

$$AR = \frac{NFC_{GA}}{NFC_{OGAE}} \tag{4}$$

If $AR > 1$, it means OGAE is converging faster than the GA.

4.1.2. *Success rate.* This can be defined by number of times the algorithm able to achieve the target value to number of total trials [6].

$$SR = \frac{\text{Number of times achieve target value}}{\text{total number of runs}} \tag{5}$$

Average success rates $SR_{Ave}$ and average acceleration rate $AR_{Ave}$ are also calculated for n benchmark test functions using the following definitions [6].

$$SR_{Ave} = \frac{1}{n}\sum_{i=1}^{n} SR_i \tag{6}$$

$$AR_{Ave} = \frac{1}{n}\sum_{i=1}^{n} AR_i \tag{7}$$

Following parameter values are used for GA and OGAE. It is mentioned in the experiment whenever value of a parameter is changed.

TABLE 1. Parameter values used in experimentation

| PARAMETER | VALUE |
|---|---|
| Population size Ps | 30 |
| Mutation rate | 0.5 |
| No of generation | 200 |
| Crossover rate | 0.8 |
| No. of Elite individuals | 1 |
| Target fitness | 0 |
| Target Value | 1.00E-02 |
| Excitation rate | 0.3 |

All the parameters were selected empirically. Parameters values for all the experiment will remain same unless mentioned otherwise. For all the experiments, the values are reported by averaging them over 100 runs. Extra NFCs for opposition initialization and during excitation phase are also counted.

4.2. **Results.** A broad set of experiments are performed and categorized as follows.

1. Effect of performance and robustness are compared for OGAE and GA in Section 4.2.1.
2. Effect of Dimensionality of problems is evaluated in Section 4.2.2.
3. Effect of population size is investigated in Section 4.2.3.
4. Setting of Excitation rate for the OGAE is discussed in Section 4.3.4.

4.2.1. *Comparison of OGAE and GA.* First of all, we need to compare the parent algorithm GA with OGAE in terms of convergence speed and robustness. Speed of the system is checked by the number of function calls needed to achieve target value. Robustness is addressed by seeing for how many functions an algorithm achieve the target values. The results of solving 10 benchmark functions (see Table 9) are given in Table 2. The best result of the NFCs and the SR for each function are highlighted in boldface. The average success rates and the average acceleration rate on 10 test functions are shown in the last row of the table. OGAE outperforms GA on 7 test functions, while GA surpasses OGAE on 3 functions. Average acceleration rate is 1.30, which means OGAE is on average 30% faster than GA. While the average success rate for OGAE is 0.91 and for the GA it is 0.61. OGAE is more successful than GA when it is compared on the basis of Success Rate. Some sample graphs for the performance comparison between GA and OGAE are given in Figure 3 and Figure 4. These curves (best solution versus NFCs) show that how OGAE converges faster than GA toward the optimal solution. Other than average NFCs, minimum, maximum and median NFCs are also reported in the table.

**Results analysis:** With the same control parameter settings for both algorithms and fixing the excitation rate for the OGAE (Er = 0.3), their SRs are comparable, while OGAE shows better convergence speed than GA (30% faster in overall). It can be clearly scene through the results the OGAE achieve the same robustness while consuming less number of function calls. The excitation rate is an important control parameter which, if optimally set, can achieve even better results. The discussion about this parameter is covered in Section 4.2.4.

Figure 3 and Figure 4 represent two graphs that show convergence behavior of Opposition based Genetic Algorithm and simple Genetic algorithm for optimization of Rosenbrok's valley function. This can be seen that Opposition based Genetic algorithm converged quickly around 100.

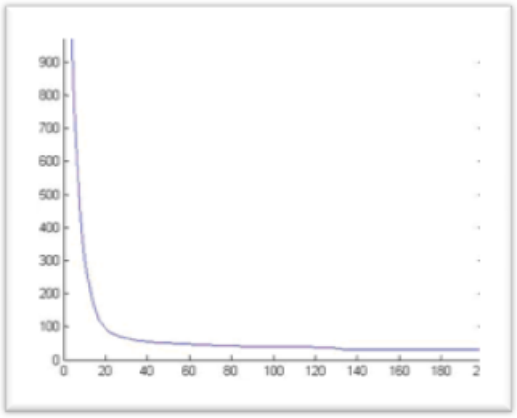


FIGURE 3. GA result for Rosenbrok's valley function F4

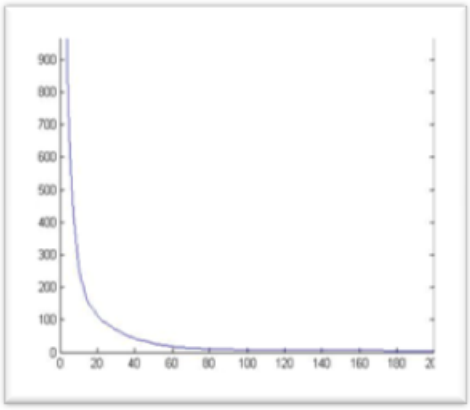


FIGURE 4. OGAE result for Rosenbrok's valley function F4

TABLE 2. Comparison of OGAE and GA on the basis of number of function calls ($NFC_s$)

| **F** | **D** | | **OGAE** | | | | **GA** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **Avg.** | **Min** | **Max** | **Med** | **SR** | **Avg.** | **Min** | **Max** | **Med** | **SR** | **AR** |
| **F1** | **30** | **4388±2015.36** | 2520 | 15540 | 5520 | **0.98** | 61142781.10 | 1500 | 11760 | 4050 | 0.96 | 1.39 |
| **F2** | **30** | **4589±2361.62** | 2460 | 18420 | 6570 | **1** | 7517±3685.27 | 1140 | 12000 | 4020 | 0.98 | 1.63 |
| **F3** | **20** | **4727±2175.49** | 2100 | 16080 | 5610 | **1** | 6202±2806.16 | 1680 | 12000 | 4290 | 0.97 | 1.31 |
| **F4** | 30 | 11448±1559.12 | 2160 | 19260 | 8790 | **0.23** | **11053±6266.16** | 5100 | 12000 | 12000 | 0.2 | 0.97 |
| **F5** | **10** | **4094±1615.75** | 1920 | 14340 | 5460 | **1** | 5828±2329.65 | 1080 | 8880 | 3720 | 0.97 | 1.42 |
| **F6** | **30** | **3840±1462.62** | 2040 | 12240 | 5070 | **1** | 5470±2094.94 | 1440 | 8400 | 3600 | 0 | 1.42 |
| **F7** | 30 | 1225±1418.90 | 180 | 4380 | 870 | **1** | **981±676.54** | 120 | 8100 | 720 | 0.03 | 0.8 |
| **F8** | **30** | **4125.6±1820.78** | 1814 | 16148 | 4834 | 0.95 | 5364.2±2288.68 | 1500 | 12000 | 3690 | **0.96** | **1.3** |
| **F9** | 2 | 13430±3800.09 | 420 | 19320 | 1794 | **0.95** | **12000±0.0** | 12000 | 12000 | 12000 | 0.04 | 0.89 |
| **F10** | **4** | **548±381.43** | 240 | 6660 | 780 | **1** | 1037±921.15 | 180 | 2940 | 480 | **1** | **1.89** |
| **Ave.** | | | | | | **0.91** | **Ave.** | | | | **0.61** | **1.3** |

4.2.2. *Effect of problem dimensionality.* In order to investigate the effect of the problem dimensionality, the same experiments are repeated for dimension Dnew = D/2 and for Dnew = 2*D for each scalable function from our test set. All other control parameters are kept unchanged. This experiment is necessary to see the algorithm's performance over the problems having different dimensions. Also the experiment shows the performance of

TABLE 3. Number of function calls for $D_{NEW}$ = D/2 and $D_{NEW}$ = 2*D

| F | $D_{NEW}$ | **OGAE** | | | | | **GA** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Avg. | Min | Max | Med | SR | Avg. | Min | Max | Med | SR | AR |
| F1 | 15 | 32301252 | 1666 | 12250 | 3048 | 1 | 2392893 | 400 | 8800 | 2300 | 1 | 0.74 |
| | 60 | 253879560 | 9700 | 48376 | 23857 | 0.96 | 200167541 | 7200 | 40000 | 18300 | 0.98 | 0.78 |
| F2 | 15 | 3317965 | 920 | 6456 | 3123 | 1 | 2600919 | 1200 | 7000 | 2400 | 1 | 0.78 |
| | 60 | 262889691 | 9130 | 48348 | 23787 | 0.96 | 211787994 | 9000 | 40000 | 19300 | 0.95 | 0.8 |
| F3 | 10 | 2253556 | 672 | 4092 | 2358 | 1 | 1672470 | 400 | 2800 | 1600 | 1 | 0.74 |
| | 20 | 113093613 | 4480 | 23318 | 10615 | 1 | 112004585 | 4400 | 28600 | 10900 | 1 | 0.99 |
| F4 | 15 | 111447745 | 4514 | 46916 | 7606 | 1 | 395543172 | 14400 | 40000 | 40000 | 0.02 | 3.55 |
| | 60 | 465125854 | 12228 | 48542 | 48195 | 0.12 | 39946540 | 34600 | 40000 | 40000 | 0.01 | 0.85 |
| F5 | 5 | 1053280 | 658 | 1686 | 1145 | 1 | 710254 | 400 | 1800 | 600 | 1 | 0.67 |
| | 20 | 4016954 | 2368 | 7922 | 4025 | 1 | 3158650 | 2200 | 6400 | 3000 | 1 | 0.78 |
| F6 | 15 | 3148615 | 1868 | 4802 | 3081 | 1 | 2430495 | 800 | 3400 | 2400 | 1 | 0.77 |
| | 60 | 179406056 | 9802 | 43824 | 16852 | 1 | 157865477 | 7200 | 38000 | 14900 | 1 | 0.87 |
| F7 | 15 | 1003287 | 646 | 1658 | 923 | 1 | 758281 | 400 | 1600 | 800 | 1 | 0.75 |
| | 60 | 21831330 | 674 | 12766 | 1905 | 1 | 27444684 | 400 | 40000 | 1600 | 0.99 | 1.26 |
| F8 | 15 | 3144536 | 1876 | 5282 | 3089 | 1 | 2408649 | 1200 | 6200 | 2200 | 1 | 0.77 |
| | 60 | 234948729 | 10812 | 45038 | 21789 | 1 | 192766938 | 6800 | 40000 | 19100 | 0.98 | 0.82 |
| **Ave. D/2** | | | | | | **1** | | | | | 0.87 | 1.1 |
| **Ave. 2D** | | | | | | **0.88** | | | | | 0.86 | 0.89 |
| **Ave.** | | | | | | **0.94** | | | | | 0.87 | 0.99 |

algorithm when it is applied to same problem with different level of difficulty in terms of dimensions. Results for and are given in Table 3 for 8 test functions.

TABLE 4. Optimal values achieved

| | OGAE | | | | GA | | | |
|---|---|---|---|---|---|---|---|---|
| Runs | **Mean** | **Min** | **Max** | **Med** | **Mean** | **Min** | **Max** | Med |
| F1 | 0.0 ±0.0 | 0 | 0.005 | 0 | 0.0±0.00 | 0 | 0 | 0 |
| F2 | 0.02 ±0.14 | 0 | 1 | 0 | 0.03 ±0.22 | 0 | 2 | 0 |
| F3 | 0.00 ±0.0 | 0 | 0 | 0 | 0.01 ±0.10 | 0 | 1 | 0 |
| F4 | 11.98 ±20.77 | 0 | 74.84 | 0 | 61.56 ±57.91 | 0 | 283.32 | 29 |
| F5 | 0.0 ±0.0 | 0 | 0 | 0 | 0.00 ±0.0 | 0 | 0 | 0 |
| F6 | 0.0 ±0.0 | 0 | 0 | 0 | 0.0 ±0.0 | 0 | 0 | 0 |
| F7 | 0.004 ±0.0 | 0 | 0.01 | 0.004 | 0.04 ±0.0 | 0 | 0.01 | 0.002 |
| F8 | 8.88E-16±0.0 | 8.88E-16 | 8.88E-16 | 8.88E-16 | 7.17E-03±0.07 | 0 | 0.72 | 0 |
| F9 | 0.43 ±0.68 | 0.003 | 4.45 | 0.145 | 1.15 ±1.07 | 0.03 | 4.45 | 0.78 |
| F10 | 0.0 ±0.0 | 0 | 0 | 0 | 0.0±0.0 | 0 | 0 | 0 |

According to the obtained results, OGAE outperform GA on 2 test functions, while GA outperforms OGAE on remaining 8 functions in terms of NFCs. Both algorithms are able to solve all the problems before meeting the maximum NFCs, but SR values are not same for all the functions. The average AR is equal to 1.1 for D/2 dimensions, denoting that that OGAE performs 1% faster than GA but achieving Success Rate equal to 100%, while GA was able to achieve the Success Rate equal to 87% only. The average AR is equal to 0.89 for 2*D dimensions, denoting that that OGAE performs 11% slower than GA but achieving Success Rate equal to 88%, while GA was able to achieve the Success Rate equal to 86%.

TABLE 5. Comparison of GA and OGAE (population size = 50)

| F | D | | OGAE | | | | GA | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Avg. | Min | Max | Med | SR | Avg. | Min | Max | Med | SR | AR |
| **F1** | **30** | 6281±2693 | 2386 | 15670 | 5754 | 1 | 5437±2825 | 2000 | 18600 | 5000 | 1 | 0.87 |
| **F2** | **30** | 7078±3694 | 2596 | 24194 | 6339 | 0.99 | 6091±3137 | 2100 | 20000 | 5050 | 0.99 | 0.86 |
| **F3** | **20** | 6077±2721 | 2852 | 21706 | 5493 | 1 | 2896±1662 | 1200 | 14600 | 2600 | 1 | 0.48 |
| **F4** | 30 | 17791±7571 | 3086 | 24306 | 24010 | 0.47 | 19741±1320 | 11200 | 20000 | 20000 | 0.06 | 1.11 |
| **F5** | **10** | 5737±2878 | 2404 | 18136 | 4789 | 1 | 1004±304 | 200 | 2100 | 1000 | 1 | 0.18 |
| **F6** | **30** | 5292±2138 | 2408 | 13396 | 4788 | 1 | 4524±1737 | 2000 | 11700 | 4050 | 1 | 0.85 |
| **F7** | 30 | 1195±838 | 340 | 6008 | 1013 | 1 | 1137±945 | 200 | 6000 | 900 | 1 | 0.95 |
| **F8** | **30** | 24099±73 | 23958 | 24292 | 24090 | 0.7 | 5179±2401 | 2000 | 12800 | 4500 | 1 | 0.21 |
| **F9** | 2 | 19753±9060 | 216 | 24310 | 24086 | 1 | 18816±4710 | 100 | 20000 | 20000 | 0.06 | 0.95 |
| **F10** | **4** | 3881±973 | 440 | 24214 | 1628 | 0.94 | 1483±1578 | 300 | 9000 | 1100 | 1 | 0.38 |
| **Ave.** | | | | | | **0.91** | **Ave.** | | | | **0.81** | **0.68** |

For the 2*D dimensions, OGAE could not be able to achieve more promising results but able to outperform GA in terms of SR by only 2%. The average SR for GA and OGAE for dimensions (D/2 and 2*), are 0.87 and 0.94, respectively.

**Results analysis:** It is shown by the Table 4 that OGAE perform better in terms of Success Rate. Declining the overall SR for GA and OGAE was not surprising because by doubling the problem dimension, algorithms could not solve the problem sometimes before reaching the upper limit of NFCs (which is a fixed number for all experiments). However, as seen, OGAE performs better for high-dimensional problems and the low-dimensional problems in terms of SR. OGAE can perform more gracefully than GA for the higher dimensional problems.

TABLE 6. Comparison of GA and OGAE (population size = 200)

| F | D | | OGAE | | | | GA | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Avg. | Min | Max | Med | SR | Avg. | Min | Max | Med | SR | AR |
| **F1** | **30** | 10822±3485 | 2386 | 15670 | 5754 | **1** | **7724±2507** | 4800 | 22800 | 6800 | **1** | 0.71 |
| **F2** | **30** | 12465±6594 | 2596 | 24194 | 6339 | **1** | **8876±2854** | 4400 | 17200 | 8000 | **1** | 0.71 |
| **F3** | **20** | 7338±1416 | 2852 | 21706 | 5493 | **1** | **5640±1139** | 3600 | 10000 | 5600 | **1** | 0.77 |
| **F4** | 30 | **45543±2472** | 3086 | 24306 | 24010 | **0.92** | 79724±2760 | 52400 | 80000 | 80000 | 0.01 | 1.75 |
| **F5** | **10** | 3206±770 | 2404 | 18136 | 4789 | **1** | **2336±587** | 800 | 3600 | 2400 | **1** | 0.73 |
| **F6** | **30** | 10090±1619 | 2408 | 13396 | 4788 | **1** | **7980±1402** | 5600 | 14400 | 7600 | **1** | 0.79 |
| **F7** | 30 | 2289±589 | 340 | 6008 | 1013 | **1** | **1788±572** | 800 | 3200 | 2000 | **1** | 0.78 |
| **F8** | **30** | 9901±2041 | 23958 | 24292 | 24090 | **1** | **7724±1202** | 5200 | 13200 | 7600 | **1** | 0.78 |
| **F9** | 2 | **25823±1653** | 216 | 24310 | 24086 | **0.75** | 41528±3945 | 400 | 80000 | 80000 | 0.49 | 1.61 |
| **F10** | **4** | 5423±2314 | 440 | 24214 | 1628 | **1** | **3252±3471** | 1200 | 28000 | 2400 | **1** | 0.6 |
| **Ave.** | | | | | | **0.97** | **Ave.** | | | | 0.85 | **0.92** |

4.2.3. *Evaluation of the effect of population size.* By varying population size, the capabilities of algorithm for exploration can be scene. In order to investigate the effect of the population size, the same experiments (conducted in Section 4.2.1 for Ps = 100) are repeated for Ps = 50 and Ps = 200. The results for Ps = 50 and Ps = 200 are given in Table 5 and Table 6, respectively. For Ps = 50, the average SR for GA and OGAE is 0.81 and 0.91, respectively (OGAE performs better than GA). However, GA could achieve the SR of 0.06 for both f4 and f9 and the OGAE was able to achieve SR of 0.47 and 1 for f4

and f9 respectively. GA was able to outperform OGAE for only function f10 by achieving SR of 1 as compared to SR of OGAE equal to 0.94. For all other functions the SR values of GA and OGAE are equal. AR for all the functions is 0.68, showing that OGAE has to perform more function evaluations than GA to locate the optimal value of the functions. However, OGAE outperforms GA on two functions, whereas GA outperforms OGAE on a function only. And for all other functions, the results of both functions are comparable in terms of SR. OGAE performed much better in terms of average SR by achieving 0.91 as compared to 0.81 of the GA. And for the Ps = 200, OGAE outperforms GA on all the functions by achieving better SR values. The average SR of OGAE is 0.97 as compared to 0.85 of GA. For the function F4, GA could solve it only 4% of the time and OGAE solve it 92% of the time. AR value is equal to 0.92, showing that OGAE takes more NFCs to solve the problems, which can be the results of more evaluations of excitation rate during the evolutionary process.

**Results analysis:** According to the results of Sections 4.2.3, for the majority of functions, OGAE performs better when the dimension of the problems increases. On the other hand, for higher dimensional problems a larger population size should be employed. According to the results of this section, OGAE performs better for larger population sizes.

TABLE 7. Number of generations for optimization

| Func. | OGAE | | | | GA | | | |
|---|---|---|---|---|---|---|---|---|
| | **Avg** | **Max** | **Min** | **Med** | **Avg** | **Max** | **Min** | Med |
| F1 | 67.61 ±31.64 | 176 | 25 | 60 | **69.02 ±33.59** | 196 | 25 | **68** |
| F2 | 82.79 ±40.66 | 200 | 27 | 74 | **72.71 ±39.36** | 200 | 19 | **67** |
| F3 | **68.00 ±30.50** | **171** | **22** | **61.5** | 72.38 ±36.26 | 200 | 28 | **72** |
| F4 | **121.70 ±68.80** | **200** | **25** | **95** | 188.34 ±25.98 | 200 | 85 | **200** |
| F5 | **63.84 ±26.49** | **160** | **20** | **61** | 67.04 ±26.93 | 148 | 18 | **62** |
| F6 | **59.78 ±22.52** | **133** | **23** | **56.5** | 200 ±24.38 | 140 | 24 | **60** |
| F7 | **10.22 ±7.23** | **44** | **2** | **9** | 20.42 ±23.65 | 135 | 2 | **12** |
| F8 | **63.15 ±27.14** | **191** | **21** | **57** | 68.76 ±30.35 | 200 | 25 | **62** |
| F9 | **190.38 ±42.15** | **200** | **4** | **200** | 200 ±0.00 | 200 | 200 | **200** |
| F10 | **10.92 ±10.27** | **74** | **3** | **8** | **9.91 ±6.36** | **49** | **3** | **8** |

4.2.4. *Finding the optimal value of excitation rate for functions.* In the proposed algorithm, a new control parameter Er is added to GA's parameters. Although this parameter was fixed for all experiments, the performance of OGAE can vary for different values Er. The Excitation rate for our current study was set to 0.3 without any effort to find an optimal value. In some trials, we observed that an excitation rate higher than 0.6 is not suitable for many functions and causes a premature convergence or an unusual growing of the number of function evaluations. On the other hand, excitation rate equal to 0 means no excitation at all. By this way, simply the mean value of 0 and 0.6 (Er = 0.3), was selected for all conducted experiments as a default value. So, we try to find the optimal value of excitation rate for all the optimization functions at hand. A new question arises here that for choosing optimal excitation rate which quality measure should be used. There are two choices available, one is SR and the other is NFCs. We cannot prefer one parameter over the other simply. We used the quality measure Success Performance SP as introduced [12].

$$SP = \frac{(\text{mean (NFC For Successfull Runs)})}{SR} \tag{8}$$

Experiment was performed by varying the value of excitation rate from 0 to 0.6 by an increment of 0.1, by keeping the values of all other parameters same. The optimal values of the excitation rate with respect to Success Performance SP, for all the functions are reported in the Table 8. It can be noted from the Table 8 that for all the low dimension functions, excitation rate value is low except F9. And this is the function for which the performance of OGAE is not as good as in other functions. That's why higher excitation rate was needed to improve the performance of OGAE over F9. For all the high dimension functions the excitation rate was set to 0.2 or 0.3 mostly.

TABLE 8. Optimal excitation rate $\mathrm{Er}_{OP}$ values w.r.t. SP

| | **F1** | **F2** | **F3** | **F4** | **F5** | **F6** | **F7** | **F8** | **F9** | **F10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **D** | 30 | 30 | 20 | 30 | 10 | 30 | 30 | 30 | 2 | 4 |
| $\mathbf{Er_{OP}}$ | 0.3 | 0.2 | 0.1 | 0.4 | 0.2 | 0.2 | 0.2 | 0.2 | 0.5 | 0.3 |

**Results analysis:** Excitation rate adds extra NFCs to the evolutionary process step, and however, it is observed that it speed-up the convergence rate of algorithm. So, what should be the specific excitation rate for the algorithm is to be determined carefully, or it should be chosen generally for all kinds of problems. It is observed using this experiment it is observed that like GA's other control parameters, the optimal excitation rate should be chosen empirically. Low excitation rates are recommended for the high dimension problems as can be seen in the Table 8.

5. **Conclusions.** It has been observed that Opposition Based Genetic Algorithm has performed better than Genetic Algorithm. It achieved faster convergence and higher success rate, due to its ability to represent the search space unevenly. It was also compared by genetic algorithm with random chromosomes, and however, Opposition based Genetic Algorithm performed better, confirming that this acceleration is not merely due to more population members or their comparisons. This work can be further extended for variable Excitation rate and other complex problems can used to study the effect of opposition.

**Acknowledgment.** The authors, M. Amjad Iqbal, 041106082CU-077 would like to acknowledge the Higher Education Commission (HEC) of Pakistan for providing the funding and required resources to complete this work. It would have been impossible to complete this complete without their continuous support.

## REFERENCES

[1] S. Mangano, *Computer Design*, 1995.
[2] M. Sentinella, Comparison and integrated use of differential evolution and genetic algorithms for space trajectory optimization, *IEEE Congress on Evolutionary Computation*, pp.973-978, 2007.
[3] H. Tizhoosh, Opposition-based learning: A new scheme for machine intelligence, *Proc. of International Conference on Computational Intelligence for Modeling Control and Automation – MCA*, Vienna, Austria, vol.1, pp.695-701, 2005.
[4] S. Rahnamayan, H. Tizhoosh and M. Salama, A novel population initialization method for accelerating evolutionary algorithms, *International Journal of Computers and Mathematics with Applications*, vol.53, no.10, 2007.
[5] S. Rahnamayan, H. Tizhoosh and M. Salama, Opposition-based differential evolution (ODE) with variable jumping rate, *Proc. of the 2007 IEEE Symposium on Foundations of Computational Intelligence*, 2007.
[6] S. Rahnamayan, H. Tizhoosh and M. Salama, Opposition-based differential evolution algorithms, *IEEE Congress on Evolutionary Computation*, 2006.

TABLE 9. List of functions used for optimization

| Sr | D | Function |
|---|---|---|
| F1 | N | Sphere Model<br>$f_1(x) = \sum_{i=1}^{n} x_i^2$ , with $-5.12 \leq x_i \leq 5.12$<br>$\min(f_1) = f_1(0, \ldots, 0) = 0$ |
| F2 | N | Axis parallel hyperellipsoid<br>$f_2(x) = \sum_{i=1}^{n} ix^2$ with $-5.12 \leq x_i$<br>$\min(f_2) = f_2(0, \ldots, 0) = 0$ |
| F3 | N | Schewfel Problem $f_3(x) = \sum_{i=1}^{n} \left( \sum_{j=1}^{i} x_j \right)$ with $-65 \leq x_i \leq 65$<br>$\min(f_3) = f_3(0, \ldots, 0) = 0$ |
| F4 | N | Rosenbrock's Valley<br>$f_4(x) = \sum [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$ with $-5.12 \leq x_i$<br>$\min(f_4) = f_4(0, \ldots, 0) = 0$ |
| F5 | N | Rastrigin's<br>$f_5(x) = 10n + \sum_{i=1}^{n} [(x_i^2 - 10\cos(2\pi x_i))]$ with $-5.12 \leq x_i$<br>$\min(f_5) = f_5(0, \ldots, 0) = 0$ |
| F6 | N | Griewangk's<br>$f_6(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)$ with $-600 \leq x_i \leq 600$<br>$\min(f_6) = f_6(0, \ldots, 0) = 0$ |
| F7 | N | Sum of Different Powers<br>$f_7(x) = \sum_{i=1}^{n} \left\lvert x_i^{(i+1)} \right\rvert$ with $-1 \leq x_i \leq 1$<br>$\min(f_7) = f_7(0, \ldots, 0) = 0$ |
| F8 | N | Ackley's Path function<br>$f_8(x) = -20\exp\left(-0.2\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}\right) - \exp\left(\frac{\sum_{i=1}^{n} \cos(2\pi x_i)}{n}\right) + 20 + e$<br>with $-32 \leq x_i \leq 32, \min(f_8) = f_8(0, \ldots, 0) = 0$ |
| F9 | 2 | Beale function<br>$f_9(x) = [1.5 - x_1(1 - x_2)]^2 + [2.25 - x_1(1 - x_2^2)]^2 + [2.625 - x_1(1 - x_2^3)]^2$<br>with $-4.5 \leq x_i \leq 4.5, \min(f_9) = f_9(0, \ldots, 0) = 0$ |
| F10 | 4 | Colville function<br>$f_{10}(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2$<br>$+10.1\left((x_2 - 1)^2 + (x_4 - 1)^2\right) + 19.8(x_2 - 1)(x_4 - 1)$,<br>with $-10 \leq x_i \leq 10, \min(f_{10}) = f_{10}(0, \ldots, 0) = 0$ |

[7] J. Vesterstroem and R. Thomsen, A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems, *Proc. of Congr. Evol. Comput.*, vol.2, pp.1980-1987, 2004.

[8] J. Andre, P. Siarry and T. Dognon, An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization, *Advances in Engineering Software*, vol.32, no.1, pp.49-60, 2001.

[9] O. Hrstka and A. Kučerová, Improvement of real coded genetic algorithm based on differential operators preventing premature convergence, *Advances in Engineering Software*, vol.35, no.3-4, pp.237-246, 2004.

[10] M. Ventresca and R. Tizhoosh, Improving the convergence of backpropagation by oppoiste transfer functions, *International Joint Conference on Neural Networks*, Vancouver, Canada, 2006.

[11] J. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, 1975.

[12] 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, *Tech. Rep., Rep. No. 2005005*, Nanyang Technological University, 2005.

[13] L. Yang and K. Li, The railway transportation planning problem and its genetic algorithm based tabu search algorithm, *ICIC Express Letters*, vol.3, no.3(A), pp.361-366, 2009.

[14] Y. Su, M. Namba and H. Murao, Solving a timetabling problem using distributed genetic algorithm, *ICIC Express Letters*, vol.3, no.4(A), pp.1055-1060, 2009.

[15] D. Martin, R. D. Toro, R. Haber and J. Dorronsoro, Optimal tuning of a networked linear controller using a multi-objective genetic algorithm and its application to one complex electromechanical process, *International Journal of Innovative Computing, Information and Control*, vol.5, no.10(B), pp.3405-3414, 2009.

[16] C. W. Lee and Y. C. Shin, Growing radial basis function networks using genetic algorithm and orthogonalization, *International Journal of Innovative Computing, Information and Control*, vol.5, no.11(A), pp.3933-3948, 2009.