Opposition-based Particle Swarm Algorithm with Cauchy Mutation

Hui Wang
School of Computer Science
China University of Geosciences
Wuhan, 430074 China
wanghui_cug@yahoo.com.cn

Yong Liu
University of Aizu
Tsuruga, Ikki-machi, Aizu-Wakamatsu
Fukushima 965-8580 Japan
yliu@u-aizu.ac.jp

Sanyou Zeng
School of Computer Science
China University of Geosciences
Wuhan, 430074 China
sanyou-zeng@263.net

Changhe Li
School of Computer Science
China University of Geosciences
Wuhan, 430074 China
lch_wfx@yahoo.com.cn

***Abstract*—Particle Swarm Optimization (PSO) has shown its fast search speed in many complicated optimization and search problems. However, PSO could often easily fall into local optima. This paper presents an Opposition-based PSO (OPSO) to accelerate the convergence of PSO and avoid premature convergence. The proposed method employs opposition-based learning for each particle and applies a dynamic Cauchy mutation on the best particle. Experimental results on many well-known benchmark optimization problems have shown that OPSO could successfully deal with those difficult multimodal functions while maintaining fast search speed on those simple unimodal functions in the function optimization.**

Ⅰ. Introduction

Particle Swarm Optimization (PSO) was firstly introduced by Kennedy and Eberhart in 1995 [1]. It is a simple evolutionary algorithm which differs from other evolutionary algorithms in which it is motivated the simulation of social behavior. PSO has shown good performance in finding good solutions to optimization problems [2], and turned out to be another powerful tool besides other evolutionary algorithms such as genetic algorithms [3].

Like other evolutionary algorithms, PSO is also a population-based search algorithm and starts with an initial population of randomly generated solutions called particles [4]. Each particle in PSO has a position and a velocity. PSO remembers both the best position found by all particles and the best positions found by each particle in the search process. For a search problem in an n-dimensional space, a potential solution is represented by a particle that adjusts its position and velocity according to Eqs. (1) and (2):

$$\begin{aligned} V_i^{(t+1)} = w * V_i^{(t)} + c_1 * rand_1() * (pbest_i - X_i^{(t)}) \\ + c_2 * rand_2() * (gbest - X_i^{(t)}) \end{aligned} \tag{1}$$

$$X_i^{(t+1)} = X_i^{(t)} + V_i^{(t+1)} \tag{2}$$

where X_i and V_i are the position and velocity of particle i, $pbest_i$ and *gbest* are previous best particle for the *i*th particle and the global best particle found by all particles so far respectively, and *w* is an inertia factor proposed by Shi and Eberhart [5], and $rand_1()$ and $rand_2()$ are two random numbers independently generated within the range of [0,1], and c_1 and c_2 are two learning factors which control the influence of the social and cognitive components.

One problem found in the standard PSO is that it could easily fall into local optima in many optimization problems. Some research has been done to tackle this problem [6-9]. The standard PSO was inspired by the social and cognitive behavior of swarm. According to equation (1), particles are largely influenced by its previous best particles and the global best particle. Once the best particle has no change in a local optimum, all the rest particles will quickly converge to the position of the best particle. So enhancing the mutated probability of particles will lead to some changes of the best particle. These changes could help the best particle escape from local optima. In this paper, a new opposition-based PSO algorithm called OPSO is proposed. It avoids premature convergences and allows OPSO to continue search for global optima by applying opposition-based learning [10-11] and a dynamic Cauchy mutation operator with local search [12]. The main idea of OPSO makes use of the estimate and opposite estimate at the same time to achieve a better approximation for each particle, and mutates the global best particle. It is to hope that the long jump from Cauchy mutation could get the best position out of the local optima where it has fallen. OPSO has been tested on both unimodal and multimodal function optimization problems. Comparison bas been conducted between OPSO and standard PSO.

The rest of the paper is organized as follows: Section 2 describes the new OPSO algorithm. Section 3 defines the benchmark continuous optimization problems used in the experiments, and gives the experimental settings. Section 4 presents and discusses the experimental results. Finally, Section 5 concludes with a summary and a few remarks.

II. Opposition-based PSO Algorithm

A. Opposition-based learning method

Opposition-based learning (OBL), originally introduced by Hamid R. Tizhoosh [13-15], has proven to be an effective method to differential evolution (ODE) in some optimization problems [10-11]. When evaluating a solution x to a given problem, we can guess the opposite solution of x to get a better solution x'. By doing this, the distance of x from optima solution can be reduced. For instance, if x is -10 and the optimum solution is 30, then the opposite solution x' is 10 and the distance of x from the optimum solution is 40. But the distance of x' from the optimum solution is only 20. So the opposite solution x' is closer to the optimum solution. The opposite solution x' can be calculated as follows [13]:

$$x' = a + b - x \tag{3}$$

where $x \in R$ within an interval of $[a, b]$.

If the solution x is a multidimensional vector, we can generalize the opposition-based learning method analogously. Assume $P(x_1, x_2, \ldots, x_n)$ is a solution in n-dimensional space with $x_1, x_2, \ldots, x_n \in R$ and $x_i \in [a_i, b_i]$ $\forall i \in \{1, 2, \ldots n\}$. The opposite solution $OP(x_1', x_2', \ldots, x_n')$ is defined below [13]:

$$x_i' = a_i + b_i - x_i \tag{4}$$

B. Opposition-based method used in PSO

Assume $f(x)$ is fitness function to a given problem. Let $P_i(x_1, x_2, \ldots, x_n)$ be a particle in n-dimension space with $x_i \in [a_i, b_i]$ $\forall i \in \{1, 2, \ldots n\}$. $OP_i(x_1', x_2', \ldots, x_n')$ is the opposite position of P_i. If $f(OP_i)$ is better than $f(P_i)$, then update P_i with OP_i.

In order to control the step size of opposition, a_i and b_i should be updated dynamically in term of the search space of current population. It means, the minimum and maximum values of each dimension in current population ($[a^p_i, b^p_i]$) are used to calculate the opposite solution instead of predefined interval boundaries ($[a_i, b_i]$). The dynamic opposition will help particles search better positions, and accelerate the convergence. The new opposition-based method is computed as:

$$OP_{i,j} = a_j^p + b_j^p - P_{i,j} \tag{5}$$

where $P_{i,j}$ is the jth position vector of the ith particle in the population, $OP_{i,j}$ is the opposite position of $P_{i,j}$, a^p_j and b^p_j are the minimum and maximum values of the jth dimension in current population respectively.

C. Dynamic Cauchy mutation operator

Some theoretical results have shown that the particle in PSO will oscillate between their pervious best particle and the global best particle found by all particles so far before it converges [12-13]. If the searching neighbors of the global best particle would be added in each generation, it would extend the search space of the best particle. It is helpful for the whole particles to move to the better positions. This can be accomplished by having a Cauchy mutation [12] on the global best particle in every generation. The one-dimensional Cauchy density function centered at the origin is defined by:

$$f(x) = \frac{1}{\pi} \frac{t}{t^2 + x^2}, \quad -\infty < x < \infty \tag{6}$$

where t > 0 is a scale parameter [16]. The Cauchy distributed function is

$$F_t(x) = \frac{1}{2} + \frac{1}{\pi} \arctan\left(\frac{x}{t}\right) \tag{7}$$

The reason for using such a mutation operator is to increase the probability of escaping from a local optimum [17]. The Cauchy mutation operator used in OPSO is described as follows:

$$W(i) = \left(\sum_{j=1}^{PopSize} V[j][i] \right) / PopSize \tag{8}$$

where V[j][i] is the ith velocity vector of the jth particle in the population, PopSize is the population size. W(i) is a weight vector within $[-W_{\max}, W_{\max}]$, and $W_{\max}$ is set to 1 in this paper.

$$gbest'(i) = gbest(i) + w(i) * N(X_{\min}, X_{\max}) \tag{9}$$

where N is a Cauchy distributed function with the scale parameter $t = 1$, and $N(X_{\min}, X_{\max})$ is a random number within $[X_{\min}, X_{\max}]$, which is a defined domain of a test function. The main steps of OPSO algorithm are as follows:

Table 1. The main steps of OPSO algorithm.

```
Begin
  n = population size;
  P = current population
  OP = the opposite population of P
  $x^p_j \in [a^p_j, b^p_j]$;  //interval boundaries of the jth dimension in current population
  $p_o$ = opposite probability;
  best_fitness_value_so_far = the fitness value of the best particle found by
                              all particles so far
  VTR = value to reach;
  $Max_{NFC}$ = maximum number of function calls (NFC)

  while (best_fitness_value_so_far > VTR and NFC <= $Max_{NFC}$)
    if(rand(0,1)< $p_o$)
      Update the interval boundaries [$a^p_j$, $b^p_j$] in current population;
      for i = 1 to n
        Calculate the opposite particle $OP_i$ according to equation (5);
      for end
      Calculate fitness value of each particle in OP;
      Select n fittest particles in P and OP as a new current population;
      Update $pbest_i$, gbest if needed;
    else
      for each particle $P_i$
        Calculate particle velocity according to equation (1);
        Update particle position according to equation (2);
        Calculate fitness value of particle $P_i$;
        Update $pbest_i$, gbest if needed;
      for end
    if end
    for i = 1 to n
      Update W[i] according to equation (8)
      if fabs(W[i] > $W_{max}$)  W[i] = $W_{max}$
      If end
    for end
    Mutate gbest according to equation (9);
    if the fitness value of gbest' is better than gbest
      gbest = gbest'
    if end
  while end
End
```

III. Experiments

A. Benchmark problems

8 well-known test functions used in [17-18] have been chosen in our experimental studies. They are high-dimensional problems, in which functions f_1 to f_4 are unimodal functions, and functions f_5 to f_8 are multimodal functions. All the functions used in this paper are to be minimized.

Table 2. The 8 test functions used in our experimental studies, where n is the dimension of the functions, $f_{\min}$ is the minimum values of the function, and $X \subseteq R_n$ is the search space.

Test Function	n	X	$f_{\min}$
$f_1(x) = \sum_{i=1}^{n} x_i^2$	30	$[-5.12, 5.12]$	0
$f_2(x) = \sum_{i=1}^{n} i * x_i^2$	30	$[-5.12, 5.12]$	0
$f_3 = \sum_{i=1}^{n} i * x_i^4 + random[0,1)$	30	$[-1.28, 1.28]$	0
$f_4(x) = \sum_{i=1}^{n} [100(x_{i+1} - x_i^2)^2 + (1 - x_i^2)^2]$	30	$[-30, 30]$	0
$f_5 = \sum_{i=1}^{n} -x_i * \sin(-\sqrt{\lvert x_i \rvert})$	30	$[-500, 500]$	-12569.5
$f_6 = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$	30	$[-5.12, 5.12]$	0
$f_7 = -20 * \exp\left(-0.2 * \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e$	30	$[-32, 32]$	0
$f_8 = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$	30	$[-600, 600]$	0

B. Parameter settings

The selection of the parameters w, c_1, c_2 of Eq. (1) is very important. It can greatly influence the performance of PSO algorithms and its variations. By following the suggestions given in [19], $c_1 = c_2 = 1.49618$, $w = 0.72984$ and the maximum velocity V_{max} was set to the half range of the search space on each dimension [20]. To evaluate the performance of convergence, the average number of function calls (NFC) [10-11] was employed. All common parameters of PSO and OPSO are set the same to have a fair comparison. The specific parameter settings are listed as follows.

Table 3. The specific parameter settings

PopSize	Maximum number of function calls (MAX_{NFC})	$W_{\max}$	P_o
10	100,000	1	0.3

C. Experimental setup

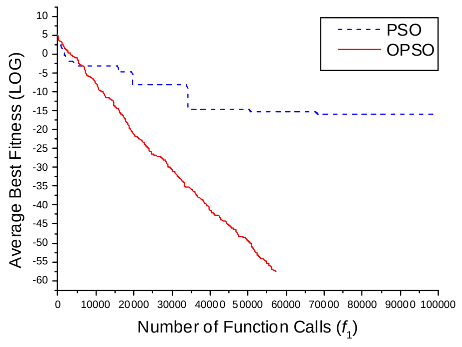
The algorithms used for comparison were PSO and OPSO. Each algorithm was tested with all of the numerical benchmarks shown in Table 2. For each algorithm, the maximum number of function calls (MAX_{NFC}) allowed was set to 100,000. If the fitness value of the best particle found by all particles so far (*best_fitness_value_so_far*) is reach to a approximate value VTR (-12569.5 for f_5 and 10^{-25} for the rest functions), we consider the current population has converged to the optimum, and the algorithm is terminated. All the experiments were conducted 50 times with different random seeds, and the average fitness of the best particles throughout the optimization run was recorded. The results below 10^{-25} will be reported as 0.

IV. Experimental Results

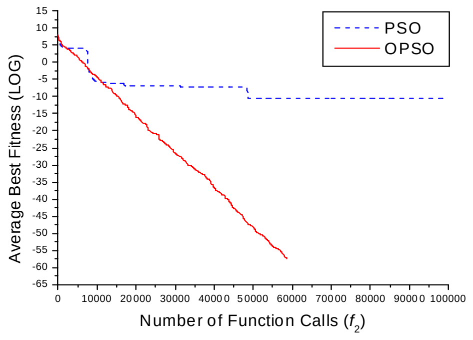
A. Comparisons between PSO and OPSO

Table 4 shows the comparison between PSO and OPSO for function f_1 to f_8, where "Mean Best" indicates the mean best function values found in the last generation, and "Std Dev" stands for the standard deviation ,and "Best" and "Worst" shows the best value and the worst value achieved by different algorithms over 50 trials respectively. It is obvious that OPSO performs better than standard PSO. However, OPSO behaves badly on f_6, but better than PSO. Figure 1 shows performance comparison between standard PSO and OPSO.

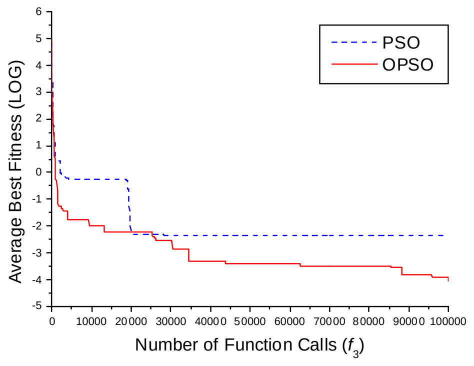
The significant improvement achieved by OPSO can be contributed to the opposition-based learning and the search ability of dynamic Cauchy mutation operator. The opposition-based learning adds the changing probability of particles, and the Cauchy mutation operator extends the search space of the best particles. Such enhanced mutated probability and extended neighbor search space will greatly help particles move to better positions. In some cases, the extended neighbors have included the global optima. Therefore, OPSO had reached better solutions than the standard PSO.



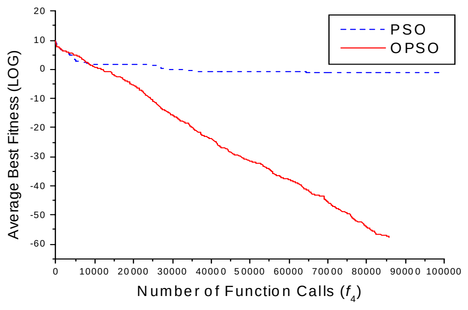
f_1



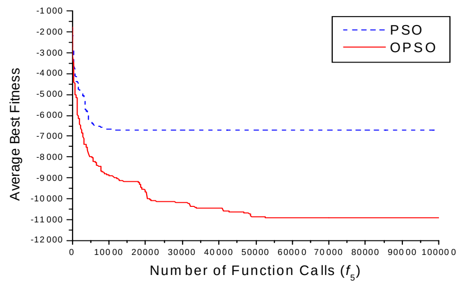
f_2



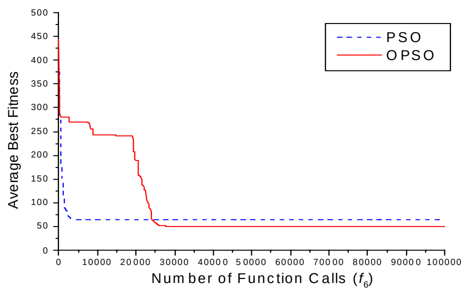
f_3



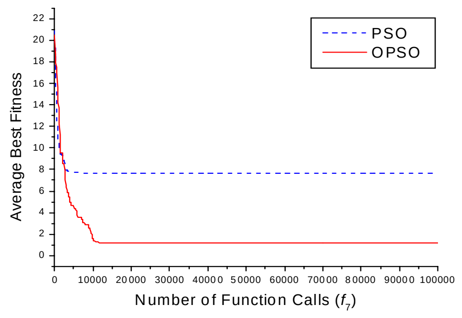
f_4



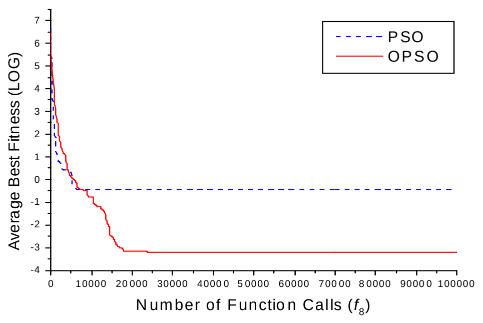
f_5



f_6



f_7



f_8

Fig. 1. Performance comparison between Standard PSO and OPSO. The horizontal axis is the average number of function calls and the vertical axis is the average best fitness value over 50 trials

Table 4. The results achieved for f_1 to f_8 using different algorithms.

Function	PSO				OPSO			
	Mean Best	Std Dev	Best	Worst	Mean Best	Std Dev	Best	Worst
f_1	6.06e-7	2.07e-6	1.98e-19	8.95e-6	0	0	0	0
f_2	3.50e-005	1.60e-4	1.26e-18	8.66e-4	0	0	0	0
f_3	9.75e-2	7.03e-2	1.33e-2	0.282	1.84e-2	5.36e-3	1.14e-2	3.29e-2
f_4	2.87	6.31	3.02e-4	21.46	0	0	0	0
f_5	-6705.1	631.2	-8621.4	-5303.6	-10986.7	207.344	-11250.6	-10674.5
f_6	62.22	12.70	34.82	83.58	49.95	11.29	36.81	59.70
f_7	7.49	2.08	4.17	12.83	1.19	1.06	0	3.35
f_8	0.659	1.15e-4	0.896	3.82	4.72e-2	4.33e-2	0	0.176

B. *Average number of evaluations*

However, the opposite population used in the OPSO will increase its computational complexity. So investigating the average number of function calls for each algorithm will be very meaningful for evaluating the performance of PSO and OPSO. All the results are given in Table 5. By contrast standard PSO, OPSO does not only cost fewer evaluations, but also achieves better solutions on f_1, f_2, f_4, f_7 and f_8. On multimodal function f_3, f_5 and f_6, though OPSO and PSO have the same average number of evaluations, but OPSO gets better solutions than PSO.

Table 5. The average number of function calls (NFC).

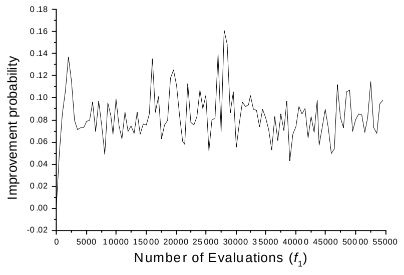
Function	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
NFC (PSO)	100,000	100,000	100,000	100,000	100,000	100,000	100,000	100,000
NFC (OPSO)	57,977	59,089	100,001	85,960	100,001	100,001	87,571	96,442

C. *Improvement probability*

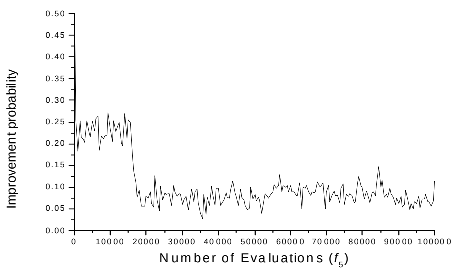
The OPSO is composed of opposition-based learning operator and standard PSO operator. Though the opposition operator occurred with a probability (P_o = 0.3) in OPSO, it showed good performance from the above result. So it is very important to observe the effects of opposition method. In this experiment, improvement probability of generated offspring by opposition operator in OPSO was calculated every 500 evaluations. The improvement probability is computed as:

$$Imp\_prob(i) = Num\_Winner \,/\, Num\_OP \qquad (10)$$

where *Num_Winner* is the number of offspring which are better than its parents in the *i*th 500 evaluations, and *Num_OP* is the number of opposition what had happened in the *i*th 500 evaluations.



f_1



f_5

Fig. 2. The improvement probability of OPSO

Because of the limited number of pages, only the evolution process of the improvement probability of opposition-based operator on functions f_1 and f_5 is shown in Fig. 2. For f_1, it could be found that opposition-based operator had rather low improvement probability. It is because that current population (*P*) and its opposite population (*OP*) had the same function values for the initial boundary a and b. With updating the boundaries dynamically, opposition-based operator could be able to make *OP* different to *P* so that it is possible to find better *OP* than *P*. For f_5, the opposition-based operator was able to find better particles at the very beginning with rather higher probabilities since *OP* and *P* had different values from the initial boundary values. The results suggest that it is essential to make *OP* be different to *P* in order to find better particles. Besides having different values, it is also important to control the step size of the opposition-based operator so that opposition-based operator could continue to find better particles.

D. *The operator probability P_o in OPSO*

To investigate the effects of the probability of the opposition-based method used in OPSO, different values of P_o are used in the following experiments. P_o is set to 0, 0.3, 0.5, 0,7 and 1.0 separately and the algorithms are run 50 times for each P_o. Table 6 shows the performance comparisons under different values of P_o. "Mean" indicates the mean best function values found in the last generation, and NFC stands for the average number of function calls over 50 trials. It suggests that P_o between 0 and 0.7 is best for the proposed algorithm. For P_o = 0, opposition-based learning does not occur at all in the OPSO, but the results still outperform standard PSO because of the search ability of Cauchy mutation operator on the best particle.

Table 6. Different values of P_o used in the OPSO.

Test	P_o = 0		P_o = 0.3		P_o = 0.5		P_o = 0.7		P_o = 1.0	
Function	Mean	NFC	Mean	NFC	Mean	NFC	Mean	NFC	Mean	NFC
f_1	8.55e-8	58,341	0	57,977	0	96,111	5.14e-12	100,001	34.45	100,001
f_2	0	58,671	0	59,089	0	96,769	2.08e-11	100,001	464.05	100,001
f_3	0.107	100,001	1.84e-2	100,001	4.47e-2	100,001	8.93e-2	100,001	8.71	100,001
f_4	1.42e-6	100,001	0	85,960	2.12e-15	100,001	6.21e-8	100,001	2336.3	100,001
f_5	-9584.8	100,001	-10986.7	100,001	-10887.1	100,001	-10893.7	100,001	-10834.7	100,001
f_6	56.51	100,001	49.95	100,001	37.86	100,001	199.21	100,001	248.32	100,001
f_7	5.04	100,001	1.19	87,571	0.568	100,001	0.832	100,001	8.65	100,001
f_8	9.94e-2	100,001	4.72e-2	96,442	6.19e-2	100,001	0.248	99,987	0.727	100,001

V. Conclusions

The idea of OPSO is to use an opposition-based learning method and a dynamic Cauchy mutation operator to help avoid local optima and accelerate the convergence of PSO. By estimating positions and the opposite positions, and applying a Cauchy mutaiton on the best particle found by all particles so far in each generation, OPSO could find better solutions than PSO.

OPSO has been compared with the standard PSO on both 4 unimodal functions and 4 multimodal functions. The results have shown that OPSO could have faster convergence on those simple unimodal functions, and better global search ability on those multimodal functions compared to the standard PSO. However, there are still fewer cases where OPSO had fallen in the local optima as what had happened on OPSO for the function f_6. It suggests that the proposed mthod might not be enough to prevent the search from falling in the local optima. Further study will foucs on how to improve the efficiency of the opposition-based method and the Cauchy mutation operator.

References

[1] J. Kennedy and R. Eberhart, Particle Swarm Optimization, IEEE International Conference on Neural Networks, Perth, Australia. 1995.

[2] K. E Parsopoulos, V. P. Plagianakos, G. D. Magoulas, M. N. Vrahatis, Objective Function “stretching” to Alleviate Convergence to Local Minima, Nonlinear Analysis TMA 47, 3419-3424, 2001.

[3] R. Eberhart and Y. Shi, Comparison between Genetic Algorithms and Particle Swarm Optimization, The 7th Annual Conference on Evolutionary Programming, San Diego, USA, 69-73, 1998

[4] X.. Hu, Y. Shi and R. Eberhart, Recentt Advenes in Particle Swarm, Congress on Evolutionary Computation, Portland, Oregon, June 19-23, 90-97, 2004

[5] Y. Shi and R. Eberhart, A Modified Partilce Swarm Optimzer, Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1998), Piscataway, NJ. 69-73, 1998.

[6] F. van den Bergh, A. P. Engelbrecht, Cooperative Learning in Neural Networks using Particle Swarm Optimization, South African Computer Journal, 84-90, Nov. 2000.

[7] X, Xie, W, Zhang, Z, Yang, Hybrid Particle Swarm Optimizer with Mass Extinction, International Conf. on Communication, Circuits and Systems (ICCCAS), Chengdu, China. 1170-1174, 2002.

[8] M. Lovbjerg, T. Krink, Extending Particle Swarm Optimisers with Self-Organized Criticality, Proceedings of Fourth Congress on Evolutionary Computation, vol. 2, 1588-1593, 2002.

[9] L. S. Coelho, and R. A. Krohling, Predictive controller tuning using modified particle swarm optimization based on Cauchy and Gaussian

distributions, in Proceedings of the 8th On-Line World Conference on Soft Computing in Industrial Applications. WSC8, 2003.

[10] S. Rahnamayan, H. R. Tizhoosh, M. M. A. Salama, Opposition-based dufferential evolution algorithms, IEEE Congress on Evolutionary Computation, Vancourver, BC, Canada, 2006

[11] S. Rahnamayan, H. R. Tizhoosh, M. M. A. Salama, Opposition-based dufferential evolution for optimization of noisy problems, IEEE Congress on Evolutionary Computation, Vancourver, BC, Canada, 2006

[12] Hui Wang, Yong Liu, Changhe Li, Sanyou Zeng, A hybrid particle swarm algorithm with Cauchy mutation, IEEE Swarm Intelligence Symposimu 2007 (SIS 2007), Honolulu, Hawaii, USA, 2006, in press.

[13] H. R. Tizhoosh, Opposition-based learning,: A new scheme for machine intelligence, International Conference on Computational Intelligence for Modeling Control and Automation – CIMCA 2005, Vienna, Austria, val. I, 695-701, 2005

[14] H. R. Tizhoosh, Reinforcement learning based on actions and opposite actions, ICGST International Conference on Artificial Intelligence and Machine Learning (AIML 05), Cairo, Egypt, 2005.

[15] H. R. Tizhoosh, Opposition-based reinforcement learning, Journal of Advanced Computational Intelligence and Intelligent Informatics, vol. 10, no. 3, 2006.

[16] W. Feller, An Introduction to Probability Theory and Its Applications, volume 2, John Wiley & Sons, Inc., 2nd edition, 1971.

[17] X. Yao, Y. Liu and G. Lin, Evolutionary Programing Made Faster, IEEE Transacations on Evolutionary Computation, vol. 3, 82-102, July 1999.

[18] K. Veeramachaneni, T. Peram, C. Mohan, L. A. Osadciw, Optimization Using Particle Swarms with Near Neighbor Interactions, Proc. Genetic and Evolutionary Computation (GECCO 2003), vol. 2723,110-121, 2003.

[19] F. van den Bergh, An Analysis of Particle Swarm Optimizers. PhD thesis, Department of Computer Science, University of Pretoria, South Africa, 2002.

[20] Wenjun Zhang, Xiaofeng Xie, DEPSO: Hybrid particle swarm with differential evoluion operator, IEEE Int. Conf. on System, Man & Cybernetics (SMCC), Washington, USA, 3816-3821, 2003.