Fuzzy Adaptive Turbulent Particle Swarm Optimization

Hongbo Liu
Department of Computer Science
Dalian University of Technology, Dalian, China
lhb@dlut.edu.cn

Ajith Abraham
School of Computer Science and Engineering
Chung-Ang University, Seoul, Korea
ajith.abraham@ieee.org

Abstract

In this paper, we introduce Turbulence in the Particle Swarm Optimization (TPSO) and illustrate how this approach could be used for function optimization problems involving high dimensions. The proposed algorithm uses a minimum velocity threshold to control the velocity of particles. TPSO mechanism is similar to a turbulent pump, which supplies some power to the swarm system to explore new search spaces (better solutions). The minimum velocity threshold of the particles is tuned adaptively by using a fuzzy logic controller, which is further called as Fuzzy Adaptive TPSO (FATPSO). We evaluate and compare the performance of SPSO (Standard PSO), TPSO and FATPSO. Empirical results clearly demonstrate that the performance of SPSO degrades remarkably with the increase in the dimensions of the problem, while the influence is very little in the case of TPSO and FATPSO.

1. Introduction

Particle Swarm Optimization (PSO) is inspired by swarm intelligence and theory in general such as bird flocking, fish schooling and even human social behavior [1]. It could be applied to various function optimization problems, or the problems that can be transformed to the function optimization problems. PSO has exhibited good performance across a wide range of applications [2, 3, 4]. However, its performance deteriorates as the dimensionality of the search space increases, especially for multi-modal optimization problems. PSO often demonstrates faster convergence speed in the first phase of the search, and then slows down or even becomes stagnant as the number of iterations are increased. Once the algorithm slows down, it is difficult to achieve better fitness values (good solutions). Recently, several investigations have been undertaken to improve the performance of standard PSO (SPSO). Lobjerg et al. [5] presented a hybrid PSO model with breeding and subpopulations. Mahfouf et al. [6] proposed adaptive weighted swarm optimization, which introduced a new update of the weight. In chaotic particle swarm optimization proposed by Jiang and Etorre [7], the objective was to introduce chaos theory so that chaotic mapping enjoys certainty, ergodicity and stochastic property to improve the exploration ability of the algorithm.

In this paper, we introduce turbulence in the conventional Particle Swarm Optimization (TPSO) algorithm to overcome the premature convergence problem. The basic idea is to drive those lazy particles and get them to explore new search spaces. TPSO uses a minimum velocity threshold to control the velocity of particles and also avoids clustering of particles and maintains diversity of population in the search space. The minimum velocity threshold of the particles is tuned adaptively by using a fuzzy logic controller, which is further called as Fuzzy Adaptive TPSO (FATPSO).

2 Turbulent Particle Swarm Optimization (TPSO)

2.1 Velocity Update of the Particles

The standard PSO model consists of a swarm of particles, which are initialized with a population of random candidate solutions. Each particle has a position represented by a position-vector, a velocity represented by a velocity-vector. They move iteratively through the d-dimension problem space to search the new solutions, where the fitness, f, can be calculated as a certain qualities measure. A particle decides where to move next, considering its own experience, which is the memory of its best past position, and the experience of its most successful particle in the swarm. The particle searches the solutions in the problem space with a range $[-s, s]$ (If the range is not symmetrical, it can be translated to the corresponding symmetrical range). In order to guide the particles effectively in the search space, the maximum moving distance during one iteration and its moving range must be clamped in between the maximum velocity. Interested readers could refer to Kennedy and Eberhart [1] for more details.

Some previous studies have discussed the trajectory of particles and the convergence of the algorithm [8, 9]. It has been shown that the trajectories of the particles oscillate in different sinusoidal waves and converge quickly, sometimes prematurely. During each iteration, the particle is attracted towards the location of the best fitness achieved so far by the particle itself and by the location of the best fitness achieved so far across the whole swarm. The particle could eventually loose the exploration capabilities in the future iterations. Such situations could occur even in the early stages of the search. In fact, this does not even guarantee that the algorithm has converged to a local minimum and it merely means that all the particles have converged to the best position discovered so far by the swarm.

One of the main reason for premature convergence of PSO is due to the stagnation of the particles exploration of a new search space. We propose a strategy to drive those lazy particles and let them explore better solutions. If a particle's velocity decreases to a threshold v_c, a new velocity is assigned using (2). Thus, we present the turbulent particle swarm optimization using a new velocity update equation:

$$\begin{aligned} v_{ij}(t+1) = w\hat{v} + c_1 r_1(x^{\#}_{ij}(t) - x_{ij}(t)) \\ + c_2 r_2(x^{*}_{j}(t) - x_{ij}(t)) \end{aligned} \tag{1}$$

$$\hat{v} = \begin{cases} v_{ij} & \text{if } |v_{ij}| \geq v_c \\ u(-1,1)v_{max}/\rho & \text{if } |v_{ij}| < v_c \end{cases} \tag{2}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{3}$$

where $u(-1,1)$ is the random number, uniformly distributed with the interval [-1,1], and ρ is the scaling factor to control the domain of the particle's oscillation according to v_{max}. v_c is the minimum velocity threshold, a tunable threshold parameter to limit the minimum of the particles' velocity. The performance of the algorithm is directly correlated to two parameter values, v_c and ρ. A large v_c shortens the oscillation period, and it provides a great probability for the particles to leap over local minima using the same number of iterations. But a large v_c compels particles in the quick "flying" state, which leads them not to search the solution and forcing them not to refine the search. In other words, a large v_c facilitates a global search while a smaller value facilitates a local search. By changing it dynamically, the search ability is dynamically adjusted. The value of ρ changes directly the particle oscillation domain. It is possible for particles not to jump over the local minima if there would be a large local minimum available in the objective search space. But the particle trajectory would more prone to oscillate because of a smaller value of ρ. For the desired exploration-exploitation trade-off, we divide the particle search into three stages. In the first stage the values for v_c and ρ are set at large and small values respectively. In the second stage, v_c and ρ are set at medium values and in the last stage, v_c is set at a small value and ρ is set at a large value. This enable the particles to take very large steps to explore solutions in the early stages, by scanning the whole solution space for good local minima and then in the final stages particles perform a fine grain search. The use of fuzzy logic would be suitable for dynamically tuning the velocity threshold, since it starts a run with an initial value which is changed during the run. By using the fuzzy control approach, the parameters can be adaptively regulated according to the problem environment.

2.2 Fuzzy Parameter Control

An Fuzzy Logic Controller (FLC) is composed of a knowledge base, that includes the information given by the expert in the form of linguistic control rules, a fuzzification interface, which has the effect of transforming crisp data into fuzzy sets, an inference system, that uses them together with the knowledge base to make inference by means of a reasoning method, and a defuzzification interface, that translates the fuzzy control action thus obtained to a real control action using a defuzzification method [10]. In our algorithm, two variables are selected as inputs to the fuzzy system: the Current Best Performance Evaluation ($CBPE$) [11] and the Current Velocity (CV) of the particle. For adapting to a wide range of optimization problems, $CBPE$ is normalized as (4):

$$NCBPE = \frac{CBPE - CBPE_{min}}{CBPE_{max} - CBPE_{min}} \tag{4}$$

where $CBPE_{min}$ is the estimated (or real) minimum, $CBPE_{max}$ is the worst solution to the minimization problem, which usually is the $CBPE$ at half the number of iterations. If we do not have any prior information about the objective function and if it is difficult to estimate $CBPE_{min}$ and $CBPE_{max}$, we can do some preliminary experiments by decreasing linearly from 1 to 0 during the run. One of the output variables is ρ, the scaling factor to control the domain of the particle's oscillation. Another is Vck, which controls the change of the velocity threshold according to (5):

$$v_c = e - [10(1 + Vck)] \tag{5}$$

The fuzzy inference system is listed as follows:

```
[System]
Name='FATPSO'
Type='mamdani'
Version=2.0
NumInputs=2
NumOutputs=2
NumRules=6
AndMethod='min'
OrMethod='max'
```

```
ImpMethod=‘min’
AggMethod=‘max’
DefuzzMethod=‘centroid’

[Input1] Name=‘NCBPE’
Range=[0 1]
NumMFs=3
MF1=‘Low’:‘gaussmf’, [0.005 0]
MF2=‘Medium’:‘gaussmf’, [0.03 0.1]
MF3=‘High’:‘gaussmf’, [0.25 1]

[Input2]
Name=‘CV’
Range=[0 1e-006]
NumMFs=2
MF1=‘Low’:‘trapmf’, [0 0 1e-030 1e-020]
MF2=‘High’:‘trapmf’, [1e-010 1e-008 1e-006 1e-006]

[Output1]
Name=‘Vck’
Range=[-1 2.2]
NumMFs=3
MF1=‘Low’:‘trimf’, [-1 -0.8 -0.5]
MF2=‘Medium’:‘trimf’, [-0.6 0 0.2]
MF3=‘High’:‘trimf’, [0.1 1.1 2.2]

[Output2]
Name=‘ρ’
Range=[1 120]
NumMFs=3
MF1=‘Small’:‘trimf’, [1 1 4]
MF2=‘Medium’:‘trimf’, [2.214 10.71 59.29]
MF3=‘Large’:‘trimf’, [47.15 120 120]

[Rules]
1 1, 3 0 (1) : 1
2 0, 2 0 (1) : 1
3 2, 1 0 (1) : 1
1 1, 0 3 (1) : 2
2 0, 0 2 (1) : 2
3 2, 0 1 (1) : 2
```

In the list, there are three parts: the first part is the configuration of the fuzzy system, the second one is the definition of the membership functions, and the third one is the rule base. There are two inputs and two outputs based on six rules. In the rule base, the first two columns correspond to the input variables, the second two columns correspond to the output variables, the fifth column displays the weight applied to each rule, and the sixth column is short form that indicates whether this is an AND (1) rule or an OR (2) rule. The numbers in the first four columns refer to the index number of the membership function, in which the number 1 encodes fuzzy set ‘Low’, 2 encodes ‘Medium’, and 3 encodes ‘High’. For example, the first rule is “If (NCBPE is Low) and (CV is Low) then (Vck is High) with the weight 1.”

The general structure of the FATPSO algorithm is as follows:

```
Begin FATPSO
Initialize parameters and the particles
While (the end criterion is not met) do
  $t = t + 1$
  Calculate the fitness value of each particle
  $x^* = argmin_{i=1}^{n}(f(x^*(t-1)), f(x_1(t)), f(x_2(t)),$
        $\cdots, f(x_i(t)), \cdots, f(x_n(t)))$
  For $i$= 1 to $n$
    $x_i^{\#}(t) = argmin_{i=1}^{n}(f(x_i^{\#}(t-1)), f(x_i(t))$
    For $j$ = 1 to $Dimension$
      If $abs(v_{ij}) < 1e-6$
        Obtain the velocity threshold
        {
            fismat = readfis(‘$FATPSO.fis$’)
            $FO$ = evalfis([$NCBPE \quad CV$], fismat)
        }
      Endif
      Update the $j$-th dimension value of $\vec{x}_i$ and $\vec{v}_i$
      according to equations (1, 2 and 3)
    Next $j$
  Next $i$
End While
End FATPSO
```

3. Experiment settings, Results and Discussions

In our experiments, the algorithms used for comparison were SPSO (standard PSO), TPSO (turbulent PSO) and FATPSO (Fuzzy Adaptive TPSO). Each algorithm was tested using three numerical functions. The first function, namely Quadric function, has a single minimum, the second one is highly multimodal with multiple local minima, and the third one has dimensional effect ($i * x_i$) with noise. We tested the algorithms exploring 30, 100 and 200 dimensions. For each of these functions, the goal was to find the global minima. The parameters c_1 and c_2 were set to 1.49 for all the PSO algorithms. Inertia weight w was decreased linearly from 0.9 to 0.1 for SPSO while w is set as a constant 0.7 in TPSO and FATPSO. In TPSO, ρ and v_c were set to three-pairs of different values for 5000, 1000 and 15000 iterations as shown in Table 1. v_{max} was set to s, the range of the function domain. In FATPSO, $CBPE_{min}$ was set to 0, and $CBPE_{max}$ was set equal to the value of CBPE (obtained during the first half of the number of iterations) for the second half of the number of iterations. In other words,

Table 1. Comparing the results for the function optimization problems.

f	5,000		10,000		15,000	
	v_c	ρ	v_c	ρ	v_c	ρ
f_1	$1e-6$	2	$1e-10$	10	$1e-20$	100
f_2	$1e-8$	2	$1e-10$	5	$1e-15$	10
f_3	$1e-7$	2	$1e-10$	5	$1e-20$	10

$CBPE_{max}$ decreased during the first-half of iterations, and stood during the second-half of the number of iterations. All experiments (for each benchmark) were repeated 10 times with different random seeds. Each trial had a fixed number of 20,000 iterations. The objective functions were evaluated 400,000 times during each trial, because the swarm size in all PSO algorithms were set to 20. The average fitness values of the best solutions throughout the optimization run were recorded. The averages and the standard deviations were calculated from the 10 different trials. The standard deviation indicates the differences in the results during the 10 different trials.

Benchmark functions:

- Quadric function:
  $f_1 = \sum_{i=1}^{n}(\sum_{j=1}^{i} x_j)^2$
  $\vec{x} \in [-100, 100]^n$, $min(f_1(\vec{x}^*)) = f_1(\vec{0}) = 0$.

- Rastrigin function:
  $f_2 = \sum_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i) + 10)$
  $\vec{x} \in [-5.12, 5.12]^n$, $min(f_2(\vec{x}^*)) = f_2(\vec{0}) = 0$.

- Quartic function with noise:
  $f_3 = \sum_{i=1}^{n}(ix_i^4) + random[0, 1)$
  $\vec{x} \in [-1.28, 1.28]^n$, $min(f_3(\vec{x}^*)) = f_3(\vec{0}) = 0$.

Figures 1 to 9 illustrate the mean best function values for the ten functions with three different dimensions (i.e. 30-D, 100-D and 200-D) using the three algorithms. Each algorithm for different dimensions of the same objective function has similar performance. The higher the dimension is, the higher the fitness values are. It is evident that the conventional PSO performance degrades for higher dimensions and the solutions usually get trapped in a local minimum much earlier than the complete 2000 iterations. TPSO and FATPSO have sustaining search process for better solutions and almost are not influenced by increasing the dimensions of the problem. TPSO and FATPSO both outperform SPSO significantly. In general, the performance of FATPSO was slightly better than TPSO. Although the final results of FATPSO were not much better than the TPSO, at least we need not have to set the values for the minimum

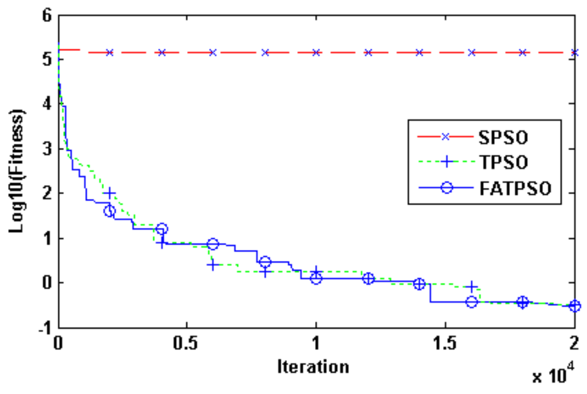

Figure 1. 30-D Quadric (f_1) function performance.

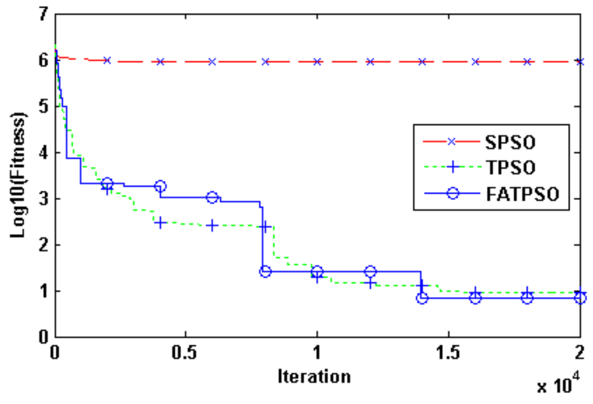

Figure 2. 100-D Quadric (f_1) function performance.

velocity threshold v_c and the scaling factor ρ. The averages and the standard deviations for 10 trials are depicted in Table 2. Larger the averages are, wider the standard deviations are usually. Not much differences in the standard deviations are observed for the different algorithms for the same benchmark functions. Referring to the empirical results, for most of considered functions FATPSO demonstrated a consistent performance pattern among all the considered algorithms.

4. Conclusions

In this paper, we introduced the Turbulent Particle Swarm Optimization (TPSO) as an alternative method for high dimension problem to overcome the problem of premature convergence in the conventional PSO algorithm. TPSO uses a minimum velocity threshold to control the velocity of particles. TPSO mechanism is similar to a

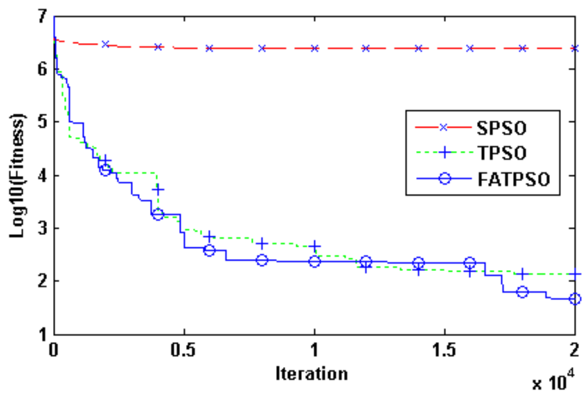

Figure 3. 200-D Quadric (f_1) function performance.

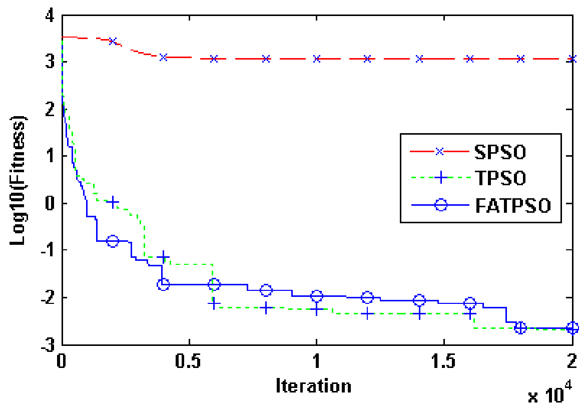

Figure 6. 200-D Rastrigin (f_2) function performance.

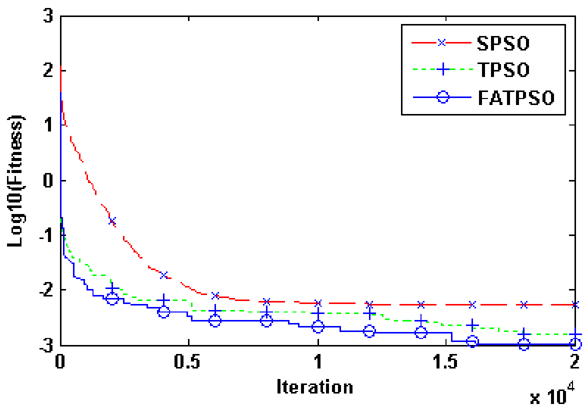

Figure 4. 30-D Rastrigin (f_2) function performance.

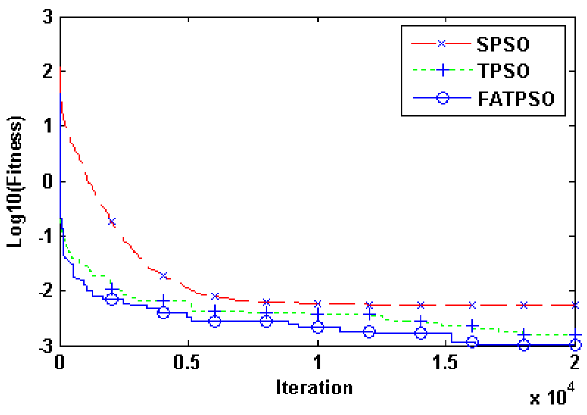

Figure 7. 30-D Quartic with Noise (f_3) function performance.

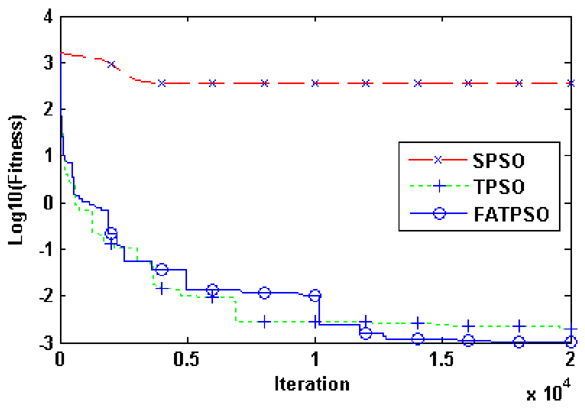

Figure 5. 100-D Rastrigin (f_2) function performance.

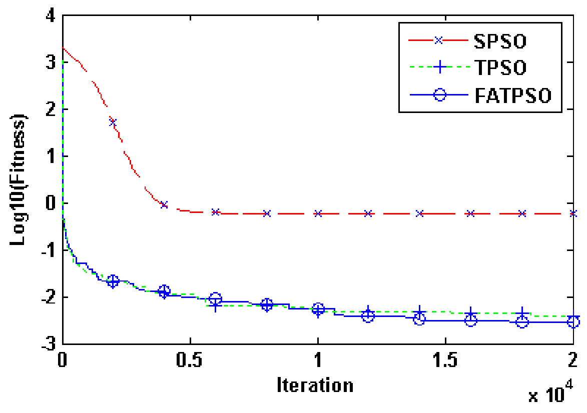

Figure 8. 100-D Quartic with Noise (f_3) function performance.

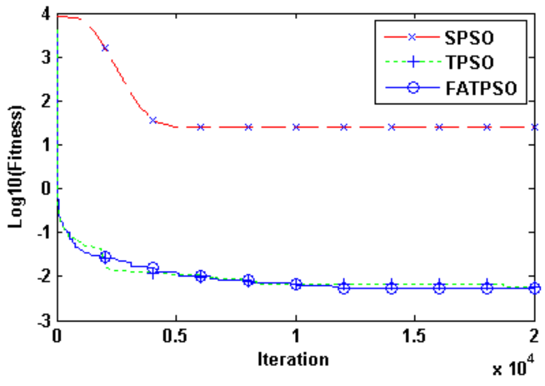

Figure 9. 200-D Quartic with Noise (f_3) function performance.

Table 2. Comparing the results for the function optimization problems.

f	D	SPSO	TPSO	FAPSO
f_1	30	1.3989e+005 ±1.0432e+005	0.3152 ±0.5708	0.2986 ±0.4154
	100	8.9865e+005 ±6.3507e+005	9.3231 ±12.9035	6.8493 ±9.2521
	200	2.4444e+006 ±1.3719e+006	137.2367 ±259.2582	45.2474 ±61.9466
f_2	30	39.4998 ±13.1666	0.0022 ±0.0057	6.7117e-004 ±0.0010
	100	361.1619 ±62.9856	0.0020 ±0.0043	0.0011 ±0.0013
	200	1.1873e+003 ±129.7462	0.0020 ±0.0031	0.0022 ±0.0035
f_3	30	0.0053 ±0.0021	0.0016 ±0.0013	0.0010 ±7.8158e-004
	100	0.5886 ±0.3916	0.0037 ±0.0023	0.0029 ±0.0017
	200	24.6138 ±18.5729	0.0058 ±0.0029	0.0053 ±0.0034

turbulence pump, which supply some power to the swarm system. The basic idea is to control the velocity of the particles to get out of possible local optima and continue exploring optimal search spaces. The minimum velocity threshold can make the particle continue moving until the algorithm converges. We evaluated and compared the performance of SPSO, TPSO and FAPSO using the different benchmark functions for higher dimensions. The results from our research demonstrate that the performance of SPSO remarkably degrades according to the increase in the dimensions, while the influence is very little in the case of TPSO and FATPSO.

Acknowledgments

This work was supported by NSFC (60373096), MOE (KP0302) and MOST (2001CCA00700). The second author acknowledges the support received from the International Joint Research Grant of the IITA (Institute of Information Technology Assessment) foreign professor invitation program of the Ministry of Information and Communication, South Korea.

References

[1] J. Kennedy and R. Eberhart, *Swarm intelligence,* Morgan Kaufmann Publishers, Inc., San Francisco, CA. 2001.

[2] K. E. Parsopoulos and M. N. Vrahatis, "Recent Approaches to Global Optimization Problems through Particle Swarm Optimization", *Natural Computing,* Kluwer Academic Publishers, 1, 2002, pp. 235-306.

[3] T. Ting, M. Rao, C. K. Loo and S. S. Ngu, "Solving Unit Commitment Problem Using Hybrid Particle Swarm Optimization", *Journal of Heuristics,* Kluwer Academic Publishers, 9, 2003, pp. 507-520.

[4] D. W. Boeringer and D. H. Werner, "Particle Swarm Optimization versus Genetic Algorithms for Phased Array Synthesis", *IEEE Transactions on Antennas and Propagation,* IEEE press, 52(3), 2004, pp. 771-779.

[5] M. Lobjerg, T. K. Rasmussen and K. Krink, "Hybrid particle swarm optimizer with breeding and subpopulations". In: Proceedings of the Third Genetic and Evolutionary Computation Conference, IEEE press, 1, 2001, pp. 469-476.

[6] M. Mahfouf , M. Y. Chen and D. A. Linkens, "Adaptive Weighted Swarm Optimization for Multiobjective Optimal Design of Alloy Steels", In: X. Yao, et al. (eds.), Lecture Notes in Computer Science, Springer-Verlag Heidelberg, PPSN VIII, LNCS 3242, 2004, pp. 762-771.

[7] C. W. Jiang and B. Etorre, "A Hybrid Method of Chaotic Particle Swarm Optimization and Linear Interior for Reactive Power Optimisation", *Mathematics and Computers in Simulation,* Elsevier, 2005, 68, pp.57-65.

[8] M. Clerc and J. Kennedy, "The Particle Swarm-Explosion, Stability, and Convergence in a Multidimensional Complex Space", *IEEE Transactions on Evolutionary Computation,* IEEE press, 6(1), 2002, pp. 58-73.

[9] T. I. Cristian, "The Particle Swarm Optimization Algorithm: Convergence Analysis and Parameter Selection", *Information Processing Letters,* Elsevier, 85(6), 2003, pp. 317-325.

[10] Y. S Yun and M. Gen, "Performance Analysis of Adaptive Genetic Algorithms with Fuzzy Logic and Heuristics", *Fuzzy Optimization and Decision Making,* Kluwer Academic Publishers, 2, 2003, pp. 161-175.

[11] Y. H. Shi and R. C. Eberhart, "Fuzzy Adaptive Particle Swarm Optimization", *Proceedings of IEEE International Conference on Evolutionary Computation,* IEEE, 2001, pp. 101-106.