A fuzzy adaptive turbulent particle swarm optimisation

Hongbo Liu

School of Computer Science and Engineering,
Dalian Maritime University,
Dalian 116024, China

Department of Computer,
Dalian University of Technology,
Dalian 116023, China
E-mail: lhb@dlut.edu.cn

Ajith Abraham*

School of Computer Science and Engineering,
Dalian Maritime University,
Dalian 116024, China

School of Computer Science,
Yonsei University, Seoul 120-749, Korea
E-mail: ajith.abraham@ieee.org
*Corresponding author

Weishi Zhang

School of Computer Science and Engineering,
Dalian Maritime University,
Dalian 116024, China
E-mail: teesiv@dlmu.edu.cn

Abstract: Particle Swarm Optimisation (PSO) algorithm is a stochastic search technique, which has exhibited good performance across a wide range of applications. However, very often for multimodal problems involving high dimensions, the algorithm tends to suffer from premature convergence. Analysis of the behaviour of the particle swarm model reveals that such premature convergence is mainly due to the decrease of velocity of particles in the search space that leads to a total implosion and ultimately fitness stagnation of the swarm. This paper introduces Turbulence in the Particle Swarm Optimisation (TPSO) algorithm to overcome the problem of stagnation. The algorithm uses a minimum velocity threshold to control the velocity of particles. The parameter, minimum velocity threshold of the particles is tuned adaptively by a fuzzy logic controller embedded in the TPSO algorithm, which is further called as Fuzzy Adaptive TPSO (FATPSO). We evaluated the performance of FATPSO and compared it with the Standard PSO (SPSO), Genetic Algorithm (GA) and Simulated Annealing (SA). The comparison was performed on a suite of 10 widely used benchmark problems for 30 and 100 dimensions. Empirical results illustrate that the FATPSO could prevent premature convergence very effectively and it clearly outperforms SPSO and GA.

Keywords: swarm-based computing; particle swarm optimisation; PSO; fuzzy controller; genetic algorithm; GA; simulated annealing; SA.

Reference to this paper should be made as follows: Liu, H., Abraham, A. and Zhang, W. (2007) 'A fuzzy adaptive turbulent particle swarm optimisation', *Int. J. Innovative Computing and Applications*, Vol. 1, No. 1, pp.39–47.

Biographical notes: Hongbo Liu is an Associate Professor at the School of Computer Science, Dalian Maritime University, Dalian, China. He received a PhD in Computer Science from Dalian University of Technology. His research interests are in computational intelligence involving brain-like computing, soft computing, nature-inspired computing, swarm intelligence, multiagent systems and other heuristics. Ajith Abraham is an Academic under the IITA Professorship Program at Yonsei University, Korea. He also holds an Adjunct Professor appointment in Dalian Maritime University, China. His

main research interests are in advanced computational intelligence with a focus on hybridising intelligent techniques involving connectionist network learning, fuzzy inference systems, swarm intelligence, evolutionary computation, distributed artificial intelligence, multiagent systems and other heuristics.

Weishi Zhang is the Dean of School of Computer Science, Dalian Maritime University, Dalian, China. His research interests are in artificial intelligence, soft computing, nature-inspired computing, swarm intelligence and generic information systems development. He is an Editorial Board Member of the International Journal of Computational Intelligence Research.

1 Introduction

Particle Swarm Optimisation (PSO) algorithm is mainly inspired by social behaviour patterns of organisms that live and interact within large groups. In particular, PSO incorporates swarming behaviours observed in flocks of birds, schools of fish or swarms of bees and even human social behaviour, from which the idea of swarm intelligence is emerged (Kennedy and Eberhart, 2001). It could be applied to solve various function optimisation problems or the problems that can be transformed to function optimisation problems. PSO has exhibited good performance across a wide range of applications (Boeringer and Werner, 2004; Du et al., 2005; Lu et al., 2003; Parsopoulos and Vrahatis, 2002; Phan et al., 2004; Schute and Groenwold, 2005; Sousa et al., 2004; Ting et al., 2003). However, its performance deteriorates as the dimensionality of the search space increases, especially for multimodal optimisation problems (Kennedy and Spears, 1998; Parsopoulos and Vrahatis, 2004). PSO algorithm often demonstrates faster convergence speed in the first phase of the search, and then slows down or even stops as the number of generations is increased. Once the algorithm slows down, it is difficult to achieve better fitness values. This state is called as stagnation or premature convergence. The trajectory of particles was given a lot of importance rather than their velocities. In this paper, we attempt to discuss the relation between the algorithm convergence and the velocities of the particles. It is found that the stagnation state is mainly due to a decrease of velocity of particles in the search space which leads to a total implosion and ultimately fitness stagnation of the swarm. We introduce Turbulent Particle Swarm Optimisation (TPSO) algorithm to improve the optimisation performance and 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 in the algorithm, which is further called as Fuzzy Adaptive TPSO (FATPSO).

This paper is organised as follows. PSO is reviewed briefly and the effects on the change of the velocities of particles are analysed in Section 2. In Section 3, we describe the TPSO model and the fuzzy adaptive processing method. Experiment settings, results and discussions are given in Section 4 followed by some conclusions in the final section.

2 Particle swarm optimisation

PSO refers to a relatively new family of algorithms that may be used to find optimal (or near optimal) solutions to numerical and qualitative problems. Some researchers have done much work on its study and development during the recent years (Jiang and Etorre, 2005; Mahfouf et al., 2004; Parsopoulos and Vrahatis, 2004; van den Bergh, 2002). We review briefly the standard particle swarm model and then analyse the various effects in the change in the velocities of particles.

2.1 Standard particle swarm model

The particle swarm model consists of a swarm of particles, which are initialised with a population of random candidate solutions. They move iteratively through the d-dimension problem space to search the new solutions, where the fitness f can be calculated as the certain qualities measure. Each particle has a position represented by a position-vector $\vec{p}_i$ (i is the index of the particle), and a velocity represented by a velocity-vector $\vec{v}_i$. Each particle remembers its own best position so far in a vector $\vec{p}_i^{\#}$ and its jth dimensional value is $p_{ij}^{\#}$. The best position-vector among the swarm so far is then stored in a vector $\vec{p}^*$ and its jth dimensional value is p_j^*. During the iteration time t, the update of the velocity from the previous velocity to the new velocity is determined by Equation (1). The new position is then determined by the sum of the previous position and the new velocity by Equation (2).

$$v_{ij}(t) = wv_{ij}(t-1) + c_1 r_1(p_{ij}^{\#}(t-1) - p_{ij}(t-1)) + c_2 r_2(p_j^*(t-1) - p_{ij}(t-1)) \quad (1)$$

$$p_{ij}(t) = p_{ij}(t-1) + v_{ij}(t) \quad (2)$$

where r_1 and r_2 are the random numbers, uniformly distributed within the interval [0,1] for the jth dimension of ith particle. c_1 is a positive constant, called as coefficient of the self-recognition component, c_2 is a positive constant, called as coefficient of the social component. The variable w is called as the inertia factor, which value is typically setup to vary linearly from 1 to near 0 during the iterated processing. From Equation (1), 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.

In the particle swarm model, the particle searches the solutions in the problem space within 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 is clamped in between the maximum velocity $[-v_{\max}, v_{\max}]$ given in Equation (3) and similarly for its moving range given in Equation (4):

$$v_{i,j} = \text{sign}(v_{i,j}) \min\left(\left|v_{i,j}\right|, v_{\max}\right) \tag{3}$$

$$p_{i,j} = \text{sign}(p_{i,j}) \min\left(\left|p_{i,j}\right|, p_{\max}\right) \tag{4}$$

The value of $v_{\max}$ is $\rho \times s$, with $0.1 \leq \rho \leq 1.0$ and is usually chosen to be s, that is, $\rho = 1$.

2.2 Velocities analysis in particle swarm

Some previous studies have discussed the trajectory of particles and the convergence of the algorithm (Clerc and Kennedy, 2002; Trelea, 2003; van den Bergh, 2002). It has been shown that the trajectories of the particles oscillate as different sinusoidal waves and converge quickly, sometimes prematurely. We analyse the effects of the change in the velocities of particles.

The gradual change of the particle's velocity can be explained geometrically. 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. From Equation (1), $v_{i,j}$ can attain a smaller value, but if the second term and the third term in RHS of Equation (1) are both small, it cannot resume a larger value and could eventually loose the exploration capabilities in the future iterations. Such situations could occur even in the early stages of the search. When the second term and the third term in RHS of Equation (1) are zero, $v_{i,j}$ will be damped quickly with the ratio of w. In other words, if a particle's current position coincides with the global best position/particle, the particle will only move away from this point if its previous velocity and w are non-zero. If their previous velocities are very close to zero, then all the particles will stop moving once they catch up with the global best particle, which many lead to premature convergence. 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. This state owes to the second term and the third term in the RHS of Equation (1), the cognitive components of the PSO. But if the cognitive components of the PSO algorithm are invalidated, all particles always search the solutions using the initial velocities. Then the algorithm is merely a degenerative stochastic search without the characteristics of PSO.

3 Turbulent swarm optimisation

In this section, we introduce a new velocity update approach for the particles in PSO and analyse its effect on the particle's behaviour. We also illustrate a Fuzzy Logic Controller (FLC) scheme to adaptively control the parameters (Herrera and Lozano, 2003; Mark and Shay, 2005; Yun and Gen, 2003).

3.1 Velocity update of the particles

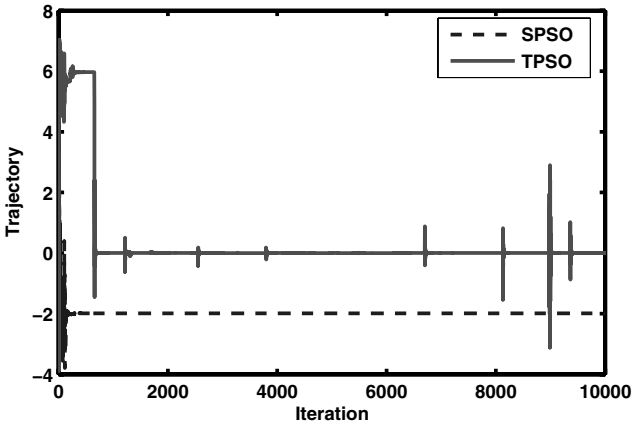
As discussed in the previous section, one of the main reason for reason for premature convergence of PSO is due to the stagnation of the particles exploration of a new search space. We introduce a strategy to drive those lazy particles and let them explore better solutions. If a particle's velocity decreases to a threshold v_c, a new velocity is assigned using Equation (6). Thus, we present the TPSO using a new velocity update equations:

$$\begin{aligned} v_{ij}(t) = w\hat{v} &+ c_1 r_1 (x_{ij}^{\#}(t-1) - x_{ij}(t-1)) \\ &+ c_2 r_2 (x_j^{*}(t-1) - x_{ij}(t-1)) \end{aligned} \tag{5}$$

$$\hat{v} = \begin{cases} v_{ij} & \text{if } \left|v_{ij}\right| \geq v_c \\ u(-1,1)v_{\max}/\rho & \text{if } \left|v_{ij}\right| < v_c \end{cases} \tag{6}$$

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. Figure 1 illustrates the trajectory of a single particle in Standard Particle Swarm Optimisation (SPSO) and TPSO, respectively.

Figure 1 Trajectory of a single particle



The change of the particle's situation 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.

3.2 Fuzzy parameter control

A 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 (Cordón et al., 1997). In our algorithm, two variables are selected as inputs to the fuzzy system: the Current Best Performance Evaluation (CBPE) (Shi and Eberhart, 2001; Shi et al., 1999) and the Current Velocity (CV) of the particle. For adapting to a wide range of optimisation problems, CBPE is normalised as Equation (7):

$$\text{NCBPE} = \frac{\text{CBPE} - \text{CBPE}_{\min}}{\text{CBPE}_{\max} - \text{CBPE}_{\min}} \tag{7}$$

where $\text{CBPE}_{\min}$ is the estimated (or real) minimum, $\text{CBPE}_{\max}$ is the worst solution to the minimisation 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 $\text{CBPE}_{\min}$ and $\text{CBPE}_{\max}$, 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 Equation (8):

$$v_c = e - [10(1 + Vck)] \tag{8}$$

The fuzzy inference system is listed briefly as follows:

[System]
Name='$FATPSO$'

[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 above mentioned 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 is illustrated in Algorithm 1.

Algorithm 1 FATPSO

01. Initialize parameters and the particles
02. While (the end criterion is not met) do
03. $t = t + 1$
04. Calculate the fitness value of each particle
05. $x^* = argmin_{i=1}^{n}(f(x^*(t-1)), f(x_1(t)),$
06. $f(x_2(t)), \cdots, f(x_i(t)), \cdots, f(x_n(t)))$
07. For i= 1 to n
08. $x_i^{\#}(t) = argmin_{i=1}^{n}(f(x_i^{\#}(t-1)), f(x_i(t))$
09. For j = 1 to d
10. If $abs(v_{ij}) < 1e-6$
11. Obtain the velocity threshold
12. {
13. fismat = readfis('$FATPSO.fis$')
14. FO = evalfis([$NCBPE$ CV], fismat)
15. }
16. Endif

Algorithm 1 FATPSO (continued)

17. Update the j-th dimension value of $\vec{x}_i$
18. and $\vec{v}_i$ according to Eqs. (1), (2) and (3)
19. Next j
20. Next i
21. End While

4 Experiments and discussions

In our experiments, the algorithms used for comparison were mainly SPSO (Eberhart and Shi, 1998), FATPSO, as well as Genetic Algorithm (GA) (Cantu-Paz, 2000) and Simulated Annealing (SA) (Orosz and Jacobson, 2002; Triki et al., 2005). The four algorithms share many similarities. GA and SA are powerful stochastic global search and optimisation methods, which are also inspired from the nature like the PSO.

GAs mimic an evolutionary natural selection process. Generations of solutions are evaluated according to a fitness value and only those candidates with high fitness values are used to create further solutions via crossover and mutation procedures.

SA is based on the manner in which liquids freeze or metals recrystalise in the process of annealing. In an annealing process, a melt, initially at high temperature and disordered, is slowly cooled so that the system at any time is approximately in thermodynamic equilibrium. In terms of computational simulation, a global minimum would correspond to such a 'frozen'(steady) ground state at the temperature $T = 0$.

Both methods are valid and efficient methods in numeric programming and have been employed in various fields due to their strong convergence properties. In our experiments, the specific parameter settings for each of the considered algorithms are described in Table 1 (Eiben et al., 1999). Each algorithm was tested with all the numerical functions shown in Table 2. The first two functions, namely Rosenbrock's and Quadric function, have a single minimum, while the other functions are highly multimodal with multiple local minima. A new function, Generalised Shubert was constructed temporarily for which global minimum function is unknown for us. It is also useful for us to validate the algorithms without knowing the optimal value. Some of the functions have the sum of their variables, some of them have the product (multiplying), some of them have dimensional effect (ix_i). We tested the algorithms on the different functions in 30 and 100 dimensions, yielding a total of 20 numerical benchmarks. For each of these functions, the goal was to find the global minima. Each algorithm (for each benchmark) was repeated 10 times with different random seeds. Each trial used a fixed number of 18,000 iterations. The objective functions were evaluated 360,000 times in each trial. Since the swarm size in all PSOs was 20, the size of the population in GA was 20 and the number operations before temperature adjustment (SA) were 20. The average fitness values of the best solutions throughout the optimisation run were recorded and 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.

Table 1 Parameter settings for the algorithms

SPSO	
Swarm size	20
Self-recognition coefficient c_1	1.49
Social coefficient c_2	1.49
Inertia weight w	$0.9 \rightarrow 0.1$
FATPSO	
Swarm size	20
Self-recognition coefficient c_1	1.49
Social coefficient c_2	1.49
Inertia weight w	0.7
GA	
Size of the population	20
Probability of crossover	0.8
Probability of mutation	0.02
SA	
Number operations before temperature adjustment	20
Number of cycles	10
Temperature reduction factor	0.85
Vector for control step of length adjustment	2

Table 2 Numerical benchmark functions

Rosenbrock (f_1):

$$f_1 = \sum_{i=1}^{n} \left(100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right);$$
$$\vec{x} \in [-2.048, 2.048]^n,$$
$$\min(f_1(\vec{x}^*)) = f_1(\vec{1}) = 0$$

Quadric (f_2):

$$f_2 = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j\right)^2;$$
$$\vec{x} \in [-100, 100]^n,$$
$$\min(f_2(\vec{x}^*)) = f_2(\vec{0}) = 0$$

Schwefel 2.22 (f_3):

$$f_3 = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|;$$
$$\vec{x} \in [-10, 10]^n,$$
$$\min(f_3(\vec{x}^*)) = f_3(\vec{0}) = 0$$

Schwefel 2.26 (f_4):

$$f_4 = 418.9829n - \sum_{i=1}^{n} (x_i sin(\sqrt{|x_i|}));$$
$$\vec{x} \in [-500, 500]^n,$$
$$\min(f_4(\vec{x}^*)) = f_4(420.\vec{9}687) \approx 0$$

Levy (f_5):

$$f_5(\vec{x}) = \pi/n\Big(k \sin^2(\pi y_1) + \sum_{i=1}^{n-1}((y_i - a)^2 (1 + k \sin^2(\pi y_{i+1}))) + (y_n - a)^2\Big),$$
$$y_i = 1 + 1/4(x_i - 1), \quad k = 10, \quad a = 1;$$
$$\vec{x} \in [-10, 10]^n,$$
$$\min(f_5(\vec{x}^*)) = f_5(\vec{1}) = 0$$

Table 2 Numerical benchmark functions (continued)

Generalised Shubert (f_6):

$f_6 = \prod_{i=1}^{n} \sum_{j=1}^{5} (j \cos((j+1)x_i + j))$;

$\vec{x} \in [-10, 10]^n$,

$\min(f_6(\vec{x}^*))$ is unknown

Rastrigin (f_7):

$f_7 = \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i) + 10\right)$

$\vec{x} \in [-5.12, 5.12]^n$,

$\min(f_7(\vec{x}^*)) = f_7(\vec{0}) = 0$

Griewank (f_8):

$f_8 = 1/4000 \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(x_i/\sqrt{i}\right) + 1$;

$\vec{x} \in [-300, 300]^n$,

$\min(f_8(\vec{x}^*)) = f_8(\vec{0}) = 0$

Ackley (f_9):

$f_9 = -20\exp\left(-0.2\sqrt{1/n \sum_i^n x_i^2}\right)$
$- \exp\left(1/n \sum_{i=1}^{n} cos(2\pi x_i)\right) + 20 + e$;

$\vec{x} \in [-32, 32]^n$,

$\min(f_9(\vec{x}^*)) = f_9(\vec{0}) = 0$

Zakharov (f_{10}):

$f_{10} = \sum_i^n x_i^2 + \left(\sum_i^n 1/2i x_i\right)^2 + \left(\sum_i^n 1/2i x_i\right)^4$;

$\vec{x} \in [-10, 10]^n$,

$\min(f_{10}(\vec{x}^*)) = f_{10}(\vec{0}) = 0$

Figures 2–11 illustrate the mean best function values for the ten functions with two different dimensions (i.e. 30- and 100-D) using the four algorithms. Each algorithm for different dimensions of the same objective function has similar performance. But in general, the higher the dimension is, the higher the fitness values are. It is observed that for almost all algorithms, the solutions get trapped in a local minimum within the first 2000 iterations except for FATPSO. For the low dimensional problems, SA is usually a cost-efficient choice. For example, SA for 30-D f_8 has a good performance than that in other situations. It is interesting that even if other algorithms are very close to or better than FATPSO in 30-D benchmarks, but a very large difference emerges in the case of 100-D benchmark problems. FATPSO becomes much better than other algorithms in general besides for f_4. The averages and the standard deviations for 10 trials are showed in Table 3. The larger the averages are, wider the standard deviations are usually. There is not too large difference of the standard deviations between the different algorithms for the same benchmark functions. Referring to the empirical results depicted in Table 3, for most of considered functions, FATPSO demonstrated a consistent performance pattern among all the considered algorithms. FATPSO performed extremely well with the exception of 30-D f_4, 100-D f_4, 30-D f_5, 30-D f_{10}, in which the results have little difference between the considered algorithms. It is to be noted that FATPSO could be an ideal choice for solving complex problems (e.g. f_2) when all other algorithms failed to give a better solution.

Figure 2 30-D Quadric (f_2) function performance

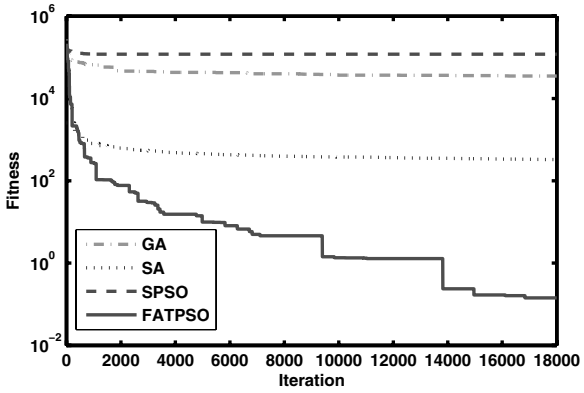


Figure 3 100-D Quadric (f_2) function performance

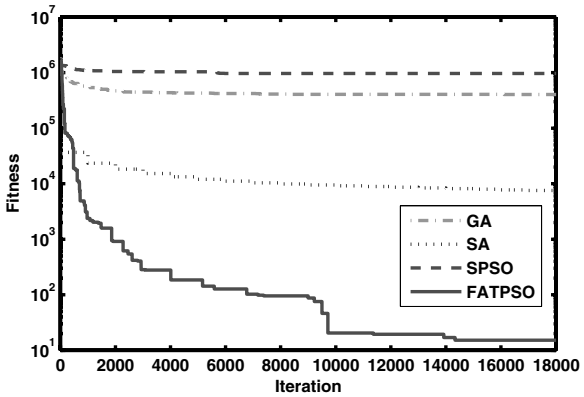


Figure 4 30-D Schwefel 2.26 (f_4) function performance

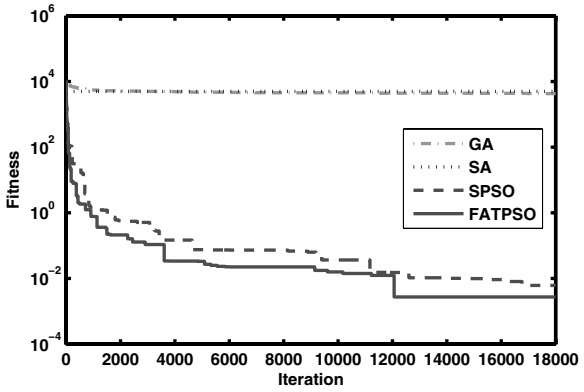


Figure 5 100-D Schwefel 2.26 (f_4) function performance

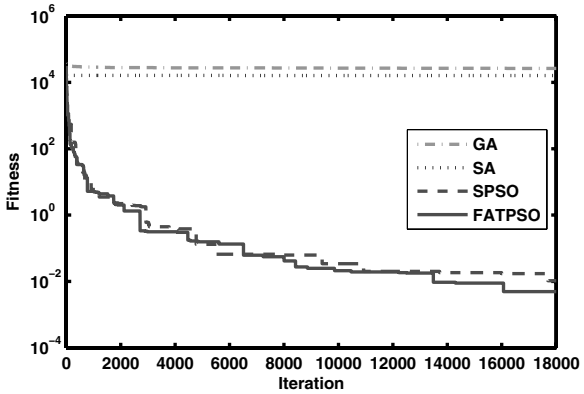


Figure 6 30-D Levy (f_5) function performance

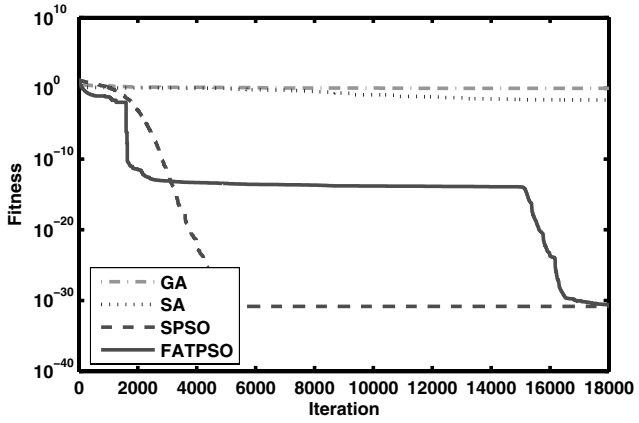


Figure 7 100-D Levy (f_5) function performance

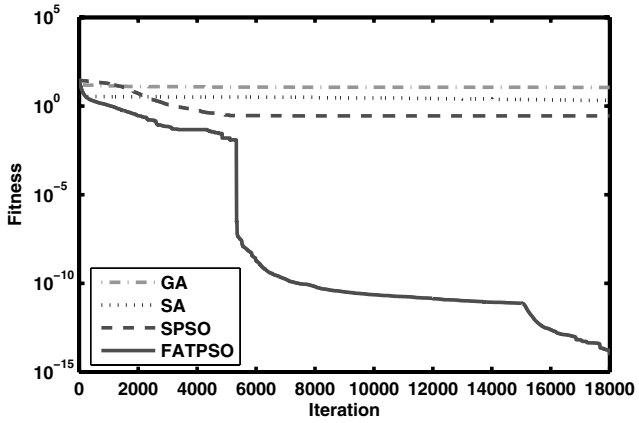


Figure 8 30-D Griewank (f_8) function performance

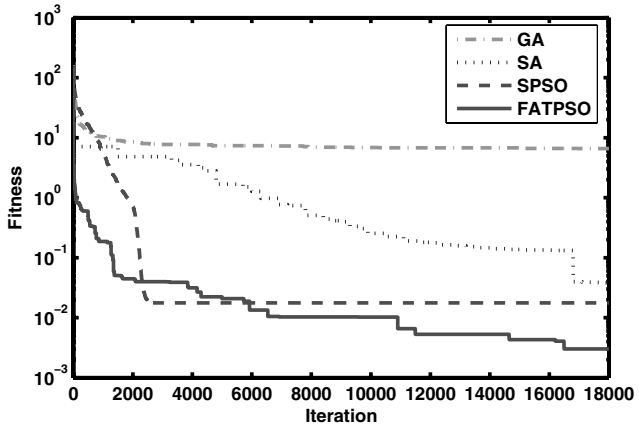


Figure 9 100-D Griewank (f_8) function performance

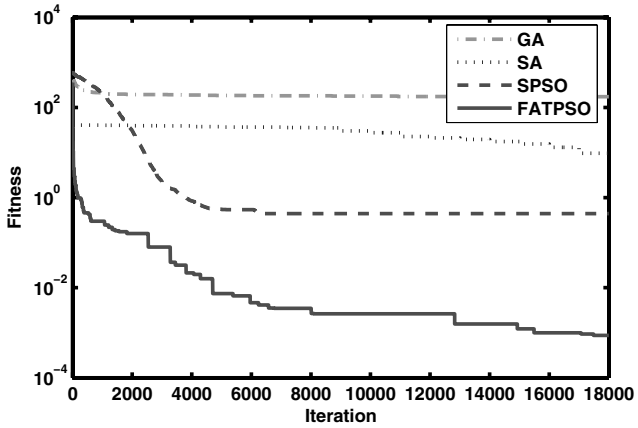


Figure 10 30-D Zakharov (f_{10}) function performance

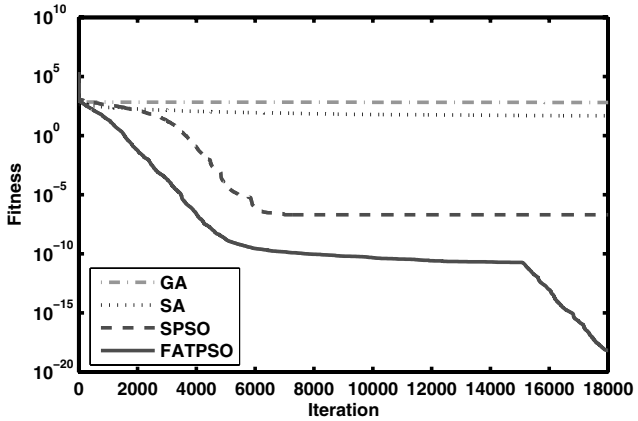
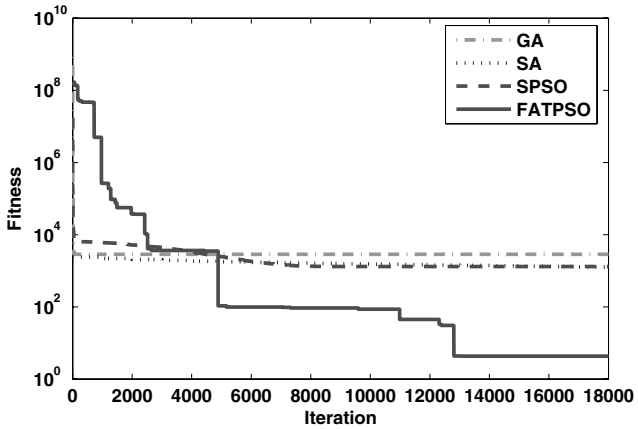


Figure 11 100-D Zakharov (f_{10}) function performance



5 Conclusion

We introduced the TPSO as an alternative method 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 turbulence pump, which supply some power to the swarm system. The basic idea is to control the velocity 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 and maintain the diversity of the population until the algorithm converges. We proposed a fuzzy logic-based system to tune adaptively the velocity threshold, which is further called as FATPSO. We evaluated and compared the performance of SPSO, FATPSO, GA and SA algorithms on a suite of 20 widely used benchmark problems. The results from our research demonstrated that FATPSO generally outperforms the other algorithms, especially for high dimensional, multimodal functions.

Table 3 Comparing the results for the function optimisation problems

f	D	*SPSO*	*FATPSO*	*GA*	*SA*
f_1	30	25.4594	1.1048×10^{-004}	222.9510	29.0552
		± 16.5424	± 0.0017	± 26.4874	± 4.8291
	100	228.6963	6.9026×10^{-004}	7.2730×10^{003}	138.3233
		± 675.7348	± 0.0080	± 459.1044	± 38.1029
f_2	30	1.1927×10^{005}	2.9699	3.7843×10^{004}	382.7578
		± 41.3785	± 24.9744	$\pm 4.4308 \times 10^{003}$	± 103.9384
	100	9.6398×10^{005}	54.0376	4.0615×10^{005}	9.5252×10^{003}
		$\pm 3.7652 \times 10^{004}$	± 482.4480	$\pm 2.2613 \times 10^{004}$	± 4.8500+003
f_3	30	2.3732×10^{-008}	5.9520×10^{-006}	20.2291	0.4991
		± 0.3763	$\pm 1.3009 \times 10^{-005}$	± 1.4324	± 1.8212
	100	55.5606	9.2702×10^{-004}	1.2391×10^{013}	23.4349
		$\pm 2.3719 \times 10^{-007}$	± 2.6465	$\pm 1.2269 \times 10^{017}$	± 5.0520
f_4	30	0.0501	0.0279	4.5094×10^{003}	4.9754×10^{003}
		± 0.2215	± 0.1086	± 294.7204	± 4.2394
	100	0.0481	0.0220	2.7101×10^{004}	1.6131×10^{004}
		± 0.7209	± 0.6902	± 528.3332	± 51.7519
f_5	30	1.4685×10^{-031}	1.5535×10^{-030}	1.0734	0.1617
		± 0.0021	$\pm 2.6040 \times 10^{-012}$	± 0.1996	± 0.4583
	100	0.2806	2.6011×10^{-011}	11.4534	2.8817
		± 2.1761	± 0.1219	± 0.4760	± 0.4526
f_6	30	-7.4305×10^{033}	-4.0465×10^{034}	-5.1931×10^{020}	-1.5457×10^{032}
		$\pm 2.3497 \times 10^{033}$	$\pm 1.2176 \times 10^{034}$	$\pm 6.9217 \times 10^{020}$	$\pm 1.2010 \times 10^{016}$
	100	-2.9776×10^{096}	-3.2111×10^{114}	-1.5347×10^{055}	-3.0040×10^{104}
		$\pm 1.2330 \times 10^{096}$	$\pm 2.4430 \times 10^{114}$	$\pm 9.4580 \times 10^{054}$	$\pm 4.2442 \times 10^{101}$
f_7	30	33.7291	8.4007×10^{-010}	204.0560	32.7997
		± 17.7719	± 9.3676	± 6.8450	± 6.9936
	100	391.0421	19.9035	1.2070×10^{003}	177.8810
		± 176.3618	± 115.9034	± 23.8156	± 37.7808
f_8	30	0.0177	0.0102	6.8463	0.3193
		± 0.3157	± 0.0149	± 0.6060	± 1.7880
	100	0.4400	0.0720	179.5966	31.4270
		± 14.4633	± 0.6945	± 7.3908	± 11.4656
f_9	30	0.6206	5.4819×10^{-004}	1.7437	0.6606
		± 0.2996	± 0.0086	± 0.0427	± 0.0657
	100	1.0666	0.0011	2.3570	1.0167
		± 0.3921	± 0.0059	± 0.0079	± 0.0532
f_{10}	30	2.0098×10^{-007}	5.911×10^{-011}	659.0997	62.2253
		± 52.8218	± 0.0626	± 12.0276	± 46.5389
	100	1.3223×10^{003}	90.1373	2.8632×10^{003}	1.5625×10^{003}
		$\pm 1.4259 \times 10^{003}$	$\pm 1.7697 \times 10^{004}$	$\pm 4.7935 \times 10^{-013}$	± 294.7468

Acknowledgements

We would like to thank Drs. Ran He and Bo Li for their scientific collaboration in this research work. This work is supported partly by NSFC (60373095), MOST (2005CB321904) and MOE (KP0302). Ajith Abraham acknowledges the support received from the International Joint Research Grant of the IITA (Institute of Information Technology Assessment) foreign professor invitation programme of the Ministry of Information and Communication, South Korea.

References

Boeringer, D.W. and Werner, D.H. (2004) 'Particle swarm optimization versus genetic algorithms for phased array synthesis', *IEEE Transactions on Antennas and Propagation*, Vol. 52, No. 3, pp.771–779.

Cantu-Paz, E. (2000) *Efficient and Accurate Parallel Genetic Algorithms*, Kluwer Academic Publishers.

Clerc, M. and Kennedy, J. (2002) 'The particle swarm-explosion, stability, and convergence in a multidimensional complex space', *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 1, pp.58–73.

Cordón, O., Herrera, F. and Peregrin, A. (1997) 'Applicability of the fuzzy operators in the design of fuzzy logic controllers', *Fuzzy Sets and Systems*, Vol. 86, pp.15–41.

Du, F., Shi, W.K., Chen, L.Z., Deng, Y. and Zhu, Z.F. (2005) 'Infrared image segmentation with 2-D maximum entropy method based on particle swarm optimization', *Pattern Recognition Letters*, No. 26, pp.597–603.

Eberhart, R.C. and Shi, Y.H. (1998) 'Comparison between genetic algorithms and particle swarm optimization', *Proceedings of IEEE International Conference on Evolutionary Computation*, pp.611–616.

Eiben, A.E., Hinterding, R. and Michalewicz, Z. (1999) 'Parameter control in evolutionary algorithms', *IEEE Transations on Evolutionary Computation*, Vol. 3, No. 2, pp.124–141.

Herrera, F. and Lozano, M. (2003) 'Fuzzy adaptive genetic algorithms: design, taxonomy, and future directions', *Soft Computing*, Vol. 7, pp.545–562.

Jiang, C.W. and Etorre, B. (2005) 'A hybrid method of chaotic particle swarm optimization and linear interior for reactive power optimisation', *Mathematics and Computers in Simulation*, Vol. 68, pp.57–65.

Kennedy, J. and Eberhart, R. (2001) *Swarm Intelligence*, San Francisco, CA: Morgan Kaufmann Publishers, Inc.

Kennedy, J. and Spears, W.M. (1998) 'Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator', *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp.78–83.

Lu, W.Z., Fan, H.Y. and Lo, S.M. (2003) 'Application of evolutionary neural network method in predicting pollutant levels in downtown area of Hong Kong', *Neuro Computing*, Vol. 51, pp.387–400.

Mahfouf, M., Chen, M.Y. and Linkens, D.A. (2004) 'Adaptive weighted swarm optimization for multiobjective optimal design of alloy steels', *Lecture Notes in Computer Science*, PPSN VIII, Vol. 3242, pp.762–771.

Mark, L. and Shay, E. (2005) 'A fuzzy-based lifetime extension of genetic algorithms', *Fuzzy Sets and Systems*, Vol. 149, pp.131–147.

Orosz, J.E. and Jacobson, S.H. (2002) 'Analysis of static simulated annealing algorithms', *Journal of Optimzation Theory and Applications*, Vol. 115, No. 1, pp.165–182.

Parsopoulos, K.E. and Vrahatis, M.N. (2002) 'Recent approaches to global optimization problems through particle swarm optimization', *Natural Computing*, Vol. 1, pp.235–306.

Parsopoulos, K.E. and Vrahatis, M.N. (2004) 'On the computation of all global minimizers through particle swarm optimization', *IEEE Transactions on Evolutionary Computation*, Vol. 8, No. 3, pp.211–224.

Phan, H.V., Lech, M. and Nguyen, T.D. (2004) 'Registration of 3D range images using particle swarm optimization', *Lecture Notes in Computer Science*, ASIAN, Vol. 3321, pp.223–235.

Schute, J.F. and Groenwold, A.A. (2005) 'A study of global optimization using particle swarms', *Journal of Global Optimization*, Vol. 31, pp.93–108.

Shi, Y.H. and Eberhart, R.C. (2001) 'Fuzzy adaptive particle swarm optimization', *Proceedings of IEEE International Conference on Evolutionary Computation*, pp.101–106.

Shi, Y.H., Eberhart, R.C. and Chen, Y. (1999) 'Implementation of evolutionary fuzzy systems', *IEEE Transactions on Fuzzy System*, Vol. 7, No. 2, pp.109–119.

Sousa, T., Silva, A. and Neves, A. (2004) 'Particle swarm based data mining algorithms for classification tasks', *Parallel Computing*, Vol. 30, pp.767–783.

Ting, T., Rao, M., Loo, C.K. and Ngu, S.S. (2003) 'Solving unit commitment problem using hybrid particle swarm optimization', *Journal of Heuristics*, Vol. 9, pp.507–520.

Trelea, I.C. (2003) 'The particle swarm optimization algorithm: convergence analysis and parameter selection', *Information Processing Letters*, Vol. 85, No. 6, pp.317–325.

Triki, E., Collette, Y. and Siarry, P. (2005) 'A theoretical study on the behavior of simulated annealing leading to a new cooling schedule', *European Journal of Operational Research*, Vol. 166, pp.77–92.

van den Bergh, F. (2002) 'An analysis of particle swarm optimizers', PhD Thesis, University of Pretoria, South Africa.

Yun, Y.S. and Gen, M. (2003) 'Performance analysis of adaptive genetic algorithms with fuzzy logic and heuristics', *Fuzzy Optimization and Decision Making*, Vol. 2, pp.161–175.