Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A note on the learning automata based algorithms for adaptive parameter selection in PSO

A.B. Hashemi*, M.R. Meybodi

Soft Computing Laboratory, Department of Computer Engineering and Information Technology, Amirkabir University of Technology, 424 Hafez Ave., Tehran 15914, Iran

A R T I C L E   I N F O

Article history:
Received 29 October 2008
Received in revised form
28 November 2009
Accepted 6 December 2009
Available online 4 January 2010

Keywords:
Particle swarm optimization
Learning automata
Parameter adaptation

A B S T R A C T

PSO, like many stochastic search methods, is very sensitive to efficient parameter setting such that modifying a single parameter may cause a considerable change in the result. In this paper, we study the ability of learning automata for adaptive PSO parameter selection. We introduced two classes of learning automata based algorithms for adaptive selection of value for inertia weight and acceleration coefficients. In the first class, particles of a swarm use the same parameter values adjusted by learning automata. In the second class, each particle has its own characteristics and sets its parameter values individually. In addition, for both classed of proposed algorithms, two approaches for changing value of the parameters has been applied. In first approach, named *adventurous*, value of a parameter is selected from a finite set while in the second approach, named *conservative*, value of a parameter either changes by a fixed amount or remains unchanged. Experimental results show that proposed learning automata based algorithms compared to other schemes such as SPSO, PSOIW, PSO-TVAC, PSOLP, DAPSO, GPSO, and DCPSO have the same or even higher ability to find better solutions. In addition, proposed algorithms converge to stopping criteria for some of the highly multi modal functions significantly faster.

1. Introduction

The particle swarm optimization (PSO) algorithm is introduced by Kennedy and Eberhart [1] based on the social behavior metaphor. In particle swarm optimization, a group of particles, without quality and volume, fly through a D-dimensional space, adjusting their positions in search space according to their own experience and their neighbors. The position of the particle i is represented with a position vector $p_i = (p_{i1}, p_{i2}, \ldots, p_{iD})$ and a velocity vector $v_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$. In every time step t, particle i changes its velocity and position according to the following equations:

$$v_i(t+1) = wv_i(t) + c_1 r_1 (pbest_i - p_i(t)) + c_2 r_2 (gbest - p_i(t)) \tag{1}$$

$$p_i(t+1) = p_i(t) + v_i(t+1) \tag{2}$$

where w is the inertial weight, and c_1 and c_2 are positive acceleration coefficients used to scale the contribution of cognitive and social components, respectively. r_1 and r_2 are uniform random variables in range [0,1]. $pbest_i$ is the best personal position of particle i which has been visited during the lifetime of the particle. $gbest$ is the global best position that is the best position of all particles in the swarm.

Performance of PSO is very sensitive to properly setting the above parameters. Several attempts have been done to improve the performance of basic PSO, which were classified by Niu et al. [2] as:

i. Adjusting the parameters in standard PSO [3–5].
ii. Designing different population topologies [6–9].
iii. Combining PSO with other search techniques [10–12].
iv. Incorporating bio-inspired mechanisms into the basic PSO [13–17].
v. Utilizing multi-population scheme instead of single population of the basic PSO [18–20].

Although many of these studies improved performance of the basic PSO, they introduced new parameters and issues, which increases the complexity of the model. We follow the opposite direction and try to improve the basic PSO by making the parameter of PSO adaptive while keeping the model as simple as possible.

Learning automata are adaptive decision making devices that operate in an unknown stochastic environment and progressively improve their performance via a learning process. It has been used successfully in many applications such as call admission control in cellular networks [21,22], capacity assignment problems [23], adaptation of back propagation parameter [24], and determination of the number of hidden units for three layers neural networks [25]. In this paper, we have proposed learning automata based

* Corresponding author.
E-mail address: ali.b.hashemi@gmail.com (A.B. Hashemi).

1568-4946/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2009.12.030

algorithms for adaptive parameter selection in PSO. Two classes of algorithms are proposed. In the first class, the same parameters are set for all particles while in the second class each particle adjusts its parameters individually. In addition, two approaches for changing value of parameters were taken. In the first approach, value of a parameter is selected from a finite set while in the second approach, value of a parameter either changes by a fixed amount or remain unchanged. Experimental results showed that proposed algorithms are highly competitive with well-known PSO algorithms in all test functions.

The rest of the paper is organized as follows: Section 2 reviews the parameter selection methods for PSO in literature. Section 3 introduces learning automata. The proposed algorithms are given in Section 4. Experiments settings and results are presented in Section 5. Section 6 concludes the paper.

2. Related work

PSO parameter selection schemes introduced in literature can be classifies in three categories. In the first category, all parameters of PSO are selected empirically. Although this approach may lead to the suboptimal results for a single problem, it would not be helpful as a general approach to solve optimization problems. Algorithms in the second category try to choose time variant parameter values, e.g. adjust particle to do more exploration rather than exploitation at the beginning. Though it helps the swarm to have different behavior during its lifetime, it could vary in different applications. Finally, algorithms in the third category adaptively change PSO parameters over time by looking on the swarm state.

2.1. PSO with constant parameters

In a swarm, acceleration coefficients, c_1 and c_2, known as cognitive and social parameter, respectively, control range of particle movement in a single iteration. Although they are not critical for convergence of PSO, assigning different values to c_1 and c_2 sometimes leads to faster convergence and improved performance [8]. Typically, these are set to a value of 2.0 [1], but experimental results indicate that alternative configurations, depending on the problem, may produce superior performance. An extended study of the acceleration coefficients in the earliest version of PSO, is given in [26].

Clerc [27] introduced a constriction model in which w, c_1 and c_2 has been chosen to insure the convergence by controlling the magnitude of the velocities, in a way similar to the velocity clamping [28]. One of such constriction model is:

$$v_i(t+1) = \chi[v_i(t) + c_1r_1(pbest_i - p_i(t)) + c_2r_2(gbest - p_i(t))] \tag{3}$$

where

$$\chi = \frac{2}{\left|2-\phi-\sqrt{\phi^2-4\phi}\right|}, \qquad \phi = c_1 + c_2, \qquad \phi > 4 \tag{4}$$

Carlisle and Dozier has reported that it might be even better to choose a larger cognitive parameter, c_1, than a social parameter, c_2, but with constraint $c_1 + c_2 \le 4$ [29].

Eberhart and Shi compared the performance of PSO using velocity clamping to one using only the constriction factor [30]. The results indicated that the inclusion of constriction factor increases the rate of convergence. However, when tested on different benchmark problems, the constriction model failed to reach the specified goal for some problems within the allocated number of iterations. Nevertheless, after applying velocity clamping to the constriction model by setting $v_{\max}$ to the length of the search space, performance improved for all test problems.

2.2. PSO with time-varying parameters

Generally, in population based optimization methods, considerably high diversity is necessary during the early stages of the search to allow use of the full range of the search space. On the other hand, during the latter stages of the search, when the algorithm is converging to the optimal solution, fine tuning of the solution is important to find the global optima efficiently. Considering these concerns, Shi and Eberhart [3,31,32] have found a significant improvement in the performance of the PSO method with a linearly varying inertia weight over the generations. In this method, particle velocity is updated according to Eqs. (5) and (6):

$$v_i(t+1) = w(t)v_i(t) + c_1r_1(pbest_i - p_i(t)) + c_2r_2(gbest - p_i(t)) \tag{5}$$

$$w(t) = w(0) - w(n_t)\frac{n_t - t}{n_t} + w(n_t) \tag{6}$$

where n_t is the maximum number of time steps for which the algorithm is executed, and $w(0)$ and $w(n_t)$ are the initial and final value for inertia weight, respectively.

The role of the inertia weight, w in Eq. (5), is considered critical for the convergence of PSO [33]. The inertia weight is used to control the impact of the previous velocity of particle on its current velocity. In addition, the inertia weight regulates the trade off between exploration and exploitation abilities of the swarm. Where a large inertia weight facilitates global exploration and a small one tends to facilitate local exploration. Through empirical studies, Shi and Eberhart [31,32] have observed that the optimal solution can be improved by linearly decreasing the value of inertia weight from 0.9 at the beginning of the search to 0.4 at the end of the search for most problems.

Suganthan suggested that both acceleration coefficient be linearly decreased, but reported no performance improvement using this scheme [8]. Ratnaweera et al. [34] proposed a PSO with time-varying acceleration coefficients (PSO-TAVC), in which c_1 decreases linearly over time while c_2 increases linearly. This strategy focuses on exploring the early stages of the optimization, while encouraging convergence to a good optimum near the end of the optimization process by attracting more particles toward the global best positions. In PSO-TVAC, the values of c_1 and c_2 at time step t are modified according to Eq. (7):

$$c_i(t) = (c_i(n_t) - c_i(0))\frac{t}{n_t} + c_i(0), \quad i = 1, 2 \tag{7}$$

The best reported results were achieved when c_2 start by 0.5 increases linearly to reach 2.5 in the last iteration, and c_1 starts with 2.5 and decreases to 0.5 in the last iteration [34]. In case of no improvement, the velocity is mutated by a mutation probability to the maximum allowed velocity multiplies by the mutation step size. In [34], the mutation step size start with 1 and decrease linearly to 0.1 in the last iteration. In addition, PSO-TVAC uses the time-varying inertia weight as proposed in [31].

Chatterjee and Siarry [4] proposed DAPSO, a PSO model with dynamic adaptation that concentrates on the adjustability of the decreasing rate for the inertia weight. In DAPSO, a set of exponential like curves are used to create a complete set of free parameters for any given problem, to remove the trial and error based approach to determine them for each specific problem.

2.3. PSO with adaptive parameters

Fan and Chang [35] proposed a nonlinear function of decreasing inertia weight model like [4] in which there is no need to know the maximum number of iterations for designing the decreasing inertia weight method. They introduced a new method for updating inertia weight according to Eq. (8):

$$w = (d)^r w_{\text{init}} \tag{8}$$

where w_{init} is the initial value for w, d is the *decrease rate* ranging between 0.1 and 1.0, and r is the dynamic adaptation parameter depending on the following rules for successive adjustment at each time step. In this model when particles find a better solution in the current iteration, it is considered that the swarm may need more exploitation capability. Therefore, w, and consequently "r", should be decreased. Otherwise, if the best solution found in the current iteration is not better than the best solution found in the previous iteration, it is considered that the swarm may need more exploration capability. Hence, w, therefore "r", should be increased. This model has been shown a significant improvement in performance, especially in terms of the solution quality and convergence speed than DAPSO [4].

Tawdross and König [36] has proposed a new approach in PSO in which, like real world, each individual has its own character, which means that each particle has different values for w, c_1, and c_2. In this approach, the velocity of the particle i is computed by Eq. (9) instead of Eq. (1):

$$v_i(t+1) = w_i v_i(t) + c_{1i} r_1 (pbest_i - p_i(t)) + c_{2i} r_2 (gbest - p_i(t)) \tag{9}$$

where w_i, c_{1i}, and c_{2i} are local parameters of the particle i. These local parameters are updated after each iteration by a controller in order to improve fitness of each particle. w_i is initialized by a random number within the range of [0.5,1.0], while c_{1i} and c_{2i} are initialized by a random number within the range of [1.75,2.5]. These local parameters are updated by a simple algorithm in which the particles that are improving will move more toward the global best by increasing c_{2i} as long as they are improving. If a particle is not improving anymore, it will start to move more toward its *pbest* and mutate its velocity with a mutation probability of 0.007. This behavior will start a new search for new local optima in the search space after particles found a local optimum. Like PSO-TAVC [34], velocity of a particle for each dimension will be reinitialize if the particle stops in that dimension. As a result, diversity of particles will increase during the search period, which improves quality of the solution.

Pasupuleti and Battiti introduced GPSO [37], in which a particle's previous velocity is not taken into account for calculating new velocity. In GPSO, the population is attracted by the global best position and each particle is reinitialized with a random velocity if it is stuck close enough to the global best position. Hence, the algorithm proceeds by greedily scouting the local minima whereas basic PSO proceeds by trying to avoid them. In each iteration, a particle either will take a step along the direction towards the global best position or reinitialized if it gets closer than ε to the global best position as follows:

$$\begin{aligned}
&\text{if}(\left\|p_i(t) - gbest\right\| < \varepsilon)\\
&\quad v_i(t+1) = rand(-V_{\max}, V_{\max})\\
&\text{else}\\
&\quad v_i(t+1) = \gamma.rand(0,1).(p_i(t) - gbest)
\end{aligned} \tag{10}$$

The factor γ determines the step size that each particle moves towards the direction of the global best position. Large values of γ will make particle i go over the global best position, thus resulting in oscillations and small values of γ will result in small step sizes, which will lead to slow convergence to the global best position. Therefore, γ is adaptively adjusted by checking the improvement on the global best value found at the end of each iteration according to Eq. (11):

$$\gamma \leftarrow \begin{cases} \max(\gamma - \delta, \gamma_{\min}) & \text{if}(f(g(t)) < f(g(t-1))) \\ \min(\gamma + \delta, \gamma_{\max}) & \text{otherwise} \end{cases} \tag{11}$$

where initial value for γ, $[\gamma_{\min}, \gamma_{\max}]$, and δ are set to 3.0, [2,4] and 0.5, respectively [37].

Yanga et al. proposed DAPSO, a PSO algorithm with dynamic adaptation [38]. In DAPSO, velocity update formula of the particle is modified so that the randomness in updating particle velocity is relatively decreased over time. Moreover, DAPSO introduced two parameters describing the state of the algorithm, the evolution speed factor, which takes the run history of each particle into account and reflects the evolutionary speed of each particle, and aggregation degree factor, which shows the deviation of best solution found in each iteration. The inertia weight of each particle is dynamically adjusted according to the evolution speed and aggregation degree.

3. Learning automata

The automata approach to learning can be considered as the determining of an optimal action from a set of actions. An automaton can be regarded as an abstract object that has finite number of actions. It selects an action from its finite set of actions and applies it to an environment. The environment evaluates the applied action and sends a reinforcement signal to automata. The response from environment is used by learning algorithm of automata to update its internal state. By continuing this process, the automaton learns to select the optimal action, which leads to favorable responses from the environment.

Variable structure learning automata are represented by the sextuple $\langle\beta,\Phi,\alpha,P,G,T\rangle$. β is a set of input actions, Φ is a set of internal states, α is a set of outputs, P denotes state probability vector governing the choice of the state at each stage, G is the output mapping, and T is learning algorithm, which is used to modify the state probability vector.

It is evident that the crucial factor affecting the performance of the variable structure learning automata is the learning algorithm for updating the action probabilities. The linear reward–penalty algorithm ($L_{\text{R–P}}$) is one of the various learning algorithms reported in the literature [39]. Let α_i be the action chosen at time k as a sample realization from distribution $p(k)$. In a P-model environment where $\beta \in \{0,1\}$, an $L_{\text{R–P}}$ scheme for updating probability vector of learning automata with r actions is defined as Eq. (12) when $\beta = 0$ (favorable response) and Eq. (13) when $\beta = 1$ (unfavorable response):

$$p_j(k+1) = \begin{cases} p_j(k) + a(1 - p_j(k)) & \text{if}\, i = j \\ p_j(k)(1 - a) & \text{if}\, i \neq j \end{cases} \tag{12}$$

$$p_j(k+1) = \begin{cases} p_j(k)(1 - b) & \text{if}\, i = j \\ \dfrac{b}{r-1} + (1 - b)p_j(k) & \text{if}\, i \neq j \end{cases} \tag{13}$$

where parameters a and b represent reward and penalty step length, respectively.

4. Proposed algorithms

Previous studies have shown that problem based tuning of parameters in PSO is a key factor to find the optimum solution accurately and efficiently. In this section, new learning automata schemes for adaptive parameter selection of PSO are presented. The proposed algorithms are classified in two classes. In the first class, named UAPSO (united or unified adaptive PSO), particles of a swarm share the same parameters, while in the second class, named IAPSO (intelligent or independent adaptive PSO), particles are independent and try to adjust their parameters using their own learning automata.

Proposed PSO algorithms perform like the standard PSO with a new supplementary section at the end of each iteration to select

```
Procedure LearningAutomataBasedAdaptivePSO
begin
    Initialize a D-dimensional PSO
    repeat
        Update particles velocity, position, pbest, gbest successively.
        Parameter Selection:
            I. Select new values for PSO parameters
           II. Update the learning automata.
    until a terminate condition is met
end
```

Fig. 1. Pseudocode of the proposed algorithms.

value of PSO parameters (Fig. 1). This additional section contains two phases, the first phase is for selecting new values of PSO parameters and the second phase is the learning phase. The first phase is studied with two approaches for selecting value of parameters for both classes of proposed model, UAPSO and IAPSO. In the second phase, two different learning phases for each class of the proposed algorithms is used.

4.1. Parameter selection approaches

Two approaches for selecting the value of a parameter is introduced and applied for both classes of proposed algorithms. In the first approach, a learning automaton selects value of a parameter from a permissible range, allowing a parameter to change drastically from one end to other end of its range in two consecutive iterations; hence, this approach is called *adventurous*. Conversely, in the second approach, new value of a parameter either will be the same as its current value or will change by a fixed amount. Since in this approach value of a parameter change gradually, it is called *conservative* approach. The *adventurous* and *conservative* are explained in detail as follows.

4.1.1. Adventurous approach

In the adventurous approach, new value of a parameter is not bind to its previous value and will be selected from a permissible range defined for the parameter. In this approach a permissible range for parameter ξ is discretized into m equally distance values. A learning automaton (LA_{ξ}) with m actions is assigned for parameter ξ so that each action of LA_{ξ} corresponds to one of m permissible values for parameter ξ. The adventurous approach works as follows. At each iteration, LA_{ξ} selects one of its actions, e.g. α_i, then corresponding value of the selected action will be set as the new value for parameter ξ (Fig. 2).

4.1.2. Conservative approach

In the conservative approach, new value of a parameter ξ will not be far from its current value. More precisely, the new value either will be the same as the current value or will be greater than or smaller than the current value by a fixed amount denoted by δ_{ξ}. This approach is modeled by a 3-action learning automata, which its actions corresponds to *increase*, *decrease*, and *no-change*. In order to select new value for parameter ξ, responsible learning automata, LA_{ξ}, select one of its actions, e.g. α_i, then depend on the selected action, value of parameter ξ for next iteration will be set accordingly (Fig. 3).

4.2. UAPSO

This class of proposed PSO algorithms is similar to a conformist society, where every member of the society behaves like everybody else. In UAPSO particles share the same values for PSO parameters, w, c_1, and c_2 which are adaptively set at each iteration using there learning automata LA_w, LA_{c1}, and LA_{c2}, respectively. This algorithm work as follows.

At iteration t, value of each parameter is selected by either the adventurous approach ($UAPSO_{Adv}$) or the conservative approach ($UAPSO_{Con}$). Then the success of the parameter selection is evaluated to update three learning automata. In UAPSO parameter selection is considered "successful" if the fraction of improved particles in the previous iteration is greater than a specified threshold θ. If the parameter selection is successful the learning automata will receive a favorable response and will be rewarded otherwise an unfavorable response will be generated and learning automata will be penalized (Figs. 4 and 5).

```
Let [$\xi_{min}$, $\xi_{max}$] be the permissible range for parameter $\xi$.
Let $LA_{\xi}$ be the learning automata with m actions {$\alpha_1$, $\alpha_2$,..., $\alpha_m$ }
{ $\xi_1$, $\xi_2$,..., $\xi_m$ }←Discretize [$\xi_{min}$ , $\xi_{max}$] into m equally distance value
At time step t:
    Select action of $LA_{\xi}$ and denote it as $\alpha_i$
    $\xi$(t+1) ← $\xi_i$
```

Fig. 2. Pseudocode of the *adventurous* approach for selecting value of parameter ξ.

```
Let [$\xi_{min}$, $\xi_{max}$] be the permissible range for parameter $\xi$
Let $\delta_{\xi}$ be the change step for parameter $\xi$
Let $LA_{\xi}$ be the learning automata with 3 actions {increase, decrease, no-change}
At time step t:
    Select action of $LA_{\xi}$ and denote it as $\alpha$.
    if $\alpha$ == increase then
        $\xi$(t+1) ← $\xi$(t) + $\delta_{\xi}$
    elseif $\alpha$ == decrease then
        $\xi$(t+1) ← $\xi$(t) - $\delta_{\xi}$
    else
        $\xi$(t+1) ← $\xi$(t)
    endif
```

Fig. 3. Pseudocode of the *conservative* approach for selecting value of parameter ξ.

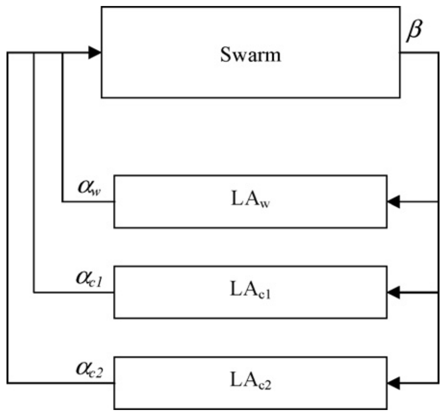


Fig. 4. Three learning automata help selecting parameters of PSO in UAPSO and share one single response to update their state.

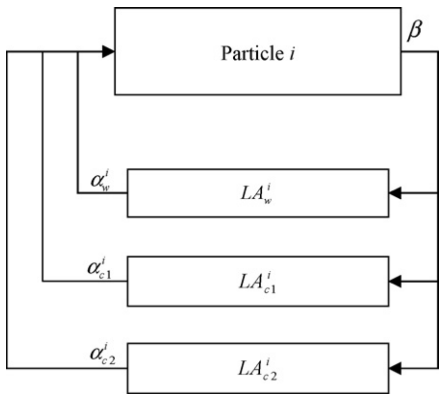


Fig. 6. Three learning automata help selecting parameters of each particle i in IAPSO and share one single response to update their state.

4.3. IAPSO

In this class of proposed algorithms, every individual is independent and can adjust its own parameters. In order to create such individuals, the parameter adaptation process is moved to a lower level, to particles, compared to UAPSO in which this process is done at the swarm level. Therefore, in this class of algorithms, each particle decides independently how to change value of its parameters. To do so, each particle adjusts its parameters, w, c_1, and c_2 with the help of three learning automata as follows.

At iteration t, particle i selects value of its parameter with either the adventurous approach (IAPSO$_{\text{Adv}}$) or the conservative approach (IAPSO$_{\text{Con}}$). Then the success of the parameter selection is evaluated in order to update three learning automata assigned for the particle. In IAPSO, parameter selection for particle i is considered as “successful” when the fitness of the particle i improves. If the parameter selection is successful three learning automata will received a favorable response and will be rewarded otherwise they will be penalized (Figs. 6 and 7).

5. Experimental study

Performance of the proposed PSO models is tested on a number of benchmark functions (Table 1) which have been extensively used in the literature [40]. The benchmark functions include two unimodal functions, Rosenbrock and Sphere and three multimodal functions, Rastrigin, Griewank, and Ackley. The Rastrigin function has many local optima around the global optima, and no correlation among its variables. The Ackley function is a multimodal at low resolution. The Griewank function is the only function, which introduces correlation between its variables. Table 1 shows the values that have been used for the dimension of these functions, feasible bounds, the range of the corresponding initial position of the particles, and the goal for each function that has to be achieved by the algorithms [7,29,30,41–44].

5.1. Experiments settings

5.1.1. Population initialization

Gehlhaar and Fogel [45] have shown that the typical uniform initialization can give false impressions of relative performance. This problem escalates specially in the functions with optima at or near the center of search space, which include all benchmark functions used in this study. Consequently, they suggested initializing in regions that expressly do not include the optima. Therefore, the asymmetric initialization already used by other researchers [43,46,47], adopted for the experiments and particles are deliberately initialized the in regions that do not include the global optimum.

```
Procedure UAPSO
begin
  Initialize a D-dimensional PSO with N particles
  Let LA_w, LA_c1, LA_c2 be the learning automata responsible for parameters w, c1,
  and c2 correspondingly
  Let θ be the success threshold for the swarm.
  Let $N_{improved}(t)$ be the number of particles which their fitness has been
  improved since iteration t-1
  repeat
     Update particles velocity, position, pbest, gbest successively.
     Select new values for swarm parameters using Adventurous (Fig. 2) or
     Conservative approach (Fig. 3)
     if $\frac{N_{improved}(t)}{N} \geq \theta$ then
          reward selected actions of LA_w, LA_c1, LA_c2 according to eq.(12).
     else
          penalize selected actions of LA_w, LA_c1, LA_c2 according to eq. (13).
     end if
  until a termination condition is met
end
```

Fig. 5. Pseudocode of UAPSO.

Table 1
Benchmark functions and their parameters.

Name	Equation	Dimension	Feasible Bounds	Initial Range	Goal
Rosenbrock	$f_{\text{Rosenbrock}} = \sum_{i=1}^{n-1}(100(x_{i+1}-x_i^2)^2+(x_i-1)^2)$	30	$[-30, 30]^n$	$[15, 30]^n$	100
Sphere	$f_{\text{Sphere}} = \sum_{i=1}^{n} x_i^2$	30	$[-5.12, 5.12]^n$	$[50, 100]^n$	0.01
Rastrigin	$f_{\text{Rastrigin}} = 10n + \sum_{i=1}^{n} x_i^2 - 10\cos 2\pi x_i$	30	$[-5.12, 5.12]^n$	$[2.56, 5.12]^n$	100
Ackley	$f_{\text{Ackley}} = 20 + e - 20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right)$	30	$[-32, 32]^n$	$[16, 32]^n$	0.1
Griewank	$f_{\text{Griewangk}} = 1 + \sum_{i=1}^{n}\frac{x_i^2}{4000} - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)$	30	$[-600, 600]^n$	$[300, 600]^n$	0.1

The particles are initialized with a random velocity where the values in every dimension are randomly chosen according to a uniform distribution over the length of search space. During a run of the PSO, position of a particle is not restricted to the defined boundaries, but when a particle passes the defined boundaries of a function (Table 1), its position will not be evaluated. Therefore the global best and particle best are always in the feasible boundaries of the search space [43].

5.1.2. *Population size*

Eberhart and Shi [30] showed that population size has almost no significant effect on the performance of the PSO. However, van den Bergh and Engelbrecht [41] suggested that even though there is a slight improvement of the optimal value with increasing swarm size, it increases the number of function evaluations to converge to an error limit. In the literature, usually the number of particles is in the range 20–60. Therefore, in this study the swarm size is set to 40 particles, like [34], for all experiments.

5.1.3. *PSO algorithms configuration*

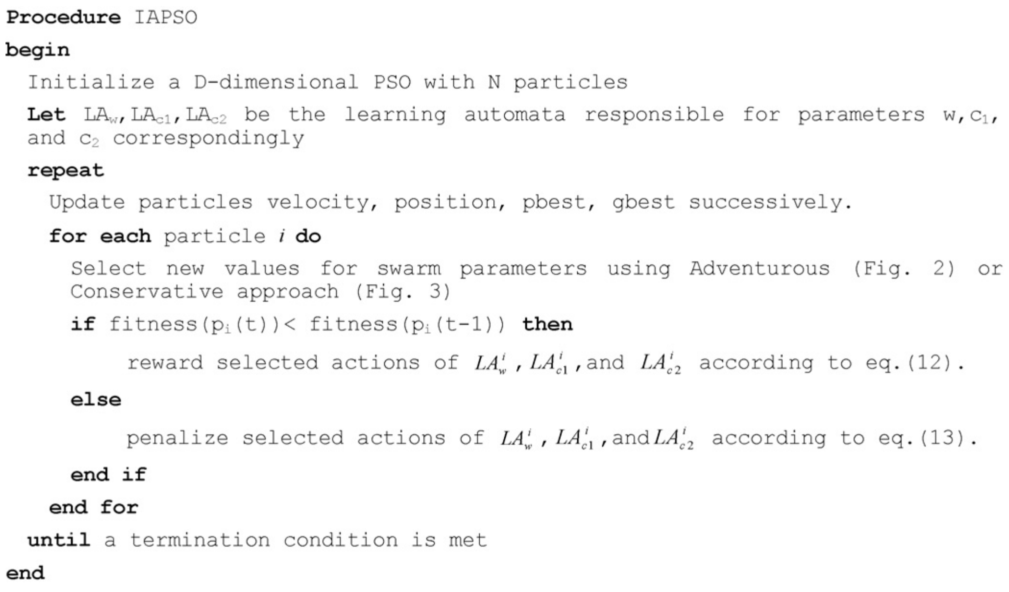
All experiments are carried out 300 times for 4×10^5 function evaluations, or until the best particle's fitness reaches the goal specified in Table 1, depending on the type of experiment being performed. For all algorithms, the average fitness error and 95% confidence interval are reported. In addition, in each experiment, result of the best performing algorithm(s) is highlighted. Where the results of the best algorithms are not statistically different, all of them are highlighted.

For adventurous algorithms, UAPSO$_{\text{Adv}}$ and IAPSO$_{\text{Adv}}$, the c_1 and c_2 are bounded in [0,4]. While for conservative algorithms the c_1 and c_2 are bounded in [0,3]. In all experiments, the permissible range for inertial weight w is set to [0,1]. Moreover, in the proposed algorithms, all learning automata use $L_{\text{R-P}}$ learning algorithm with $a = 0.01$ and $b = 0.01$.

Each of the two classes of proposed algorithms, UAPSO and IAPSO, has adopted both adventurous and conservative approach for the parameter value adjustment. Therefore, there are four algorithms to be studied. In the section, first the effect of parameters of each proposed algorithm is studied, then the comparison results of proposed algorithm and well-known PSO algorithms in the literature are presented and discussed.

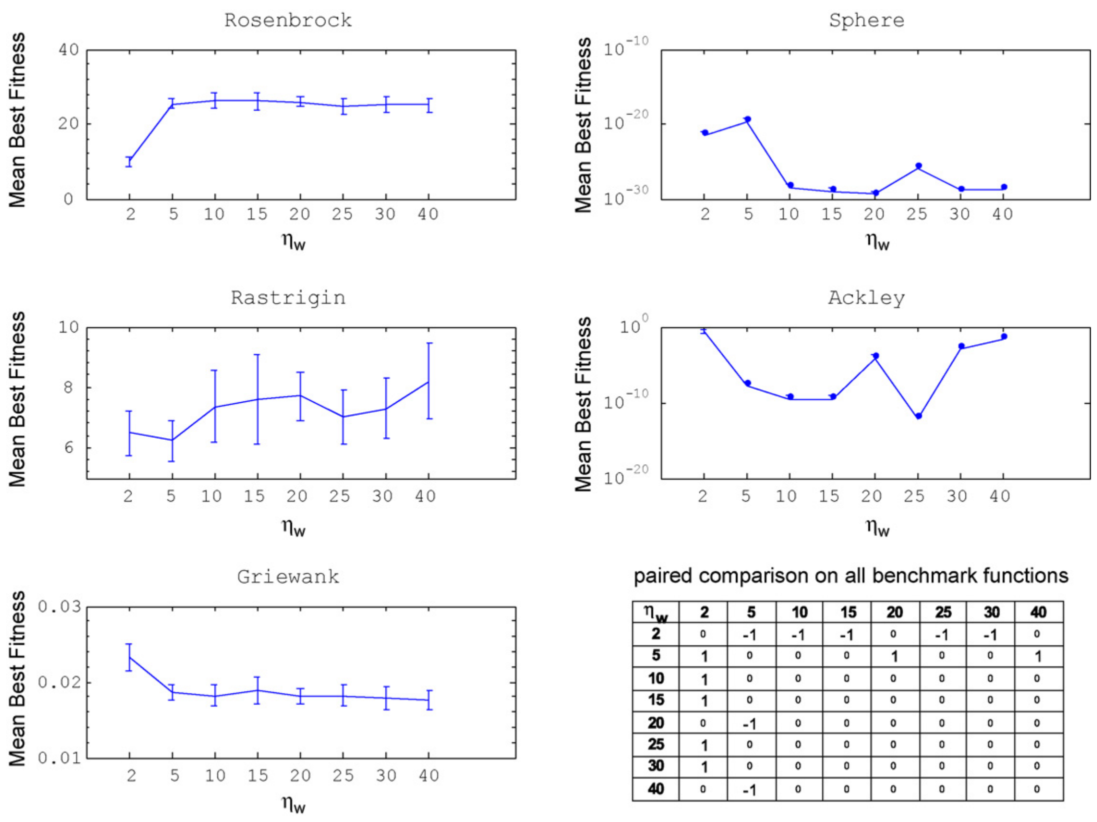
5.2. *Study of the parameters of the proposed algorithms*

In experiments 1–10, we have studied effect of each parameter of the four proposed algorithm, i.e. UAPSO$_{\text{Adv}}$, UAPSO$_{\text{Con}}$, IAPSO$_{\text{Con}}$,



```
Procedure IAPSO
begin
  Initialize a D-dimensional PSO with N particles
  Let LA_w, LA_c1, LA_c2 be the learning automata responsible for parameters w, c_1,
  and c_2 correspondingly
  repeat
    Update particles velocity, position, pbest, gbest successively.
    for each particle i do
      Select new values for swarm parameters using Adventurous (Fig. 2) or
      Conservative approach (Fig. 3)
      if fitness(p_i(t)) < fitness(p_i(t-1)) then
         reward selected actions of $LA_w^i$, $LA_{c1}^i$, and $LA_{c2}^i$ according to eq.(12).
      else
         penalize selected actions of $LA_w^i$, $LA_{c1}^i$, and $LA_{c2}^i$ according to eq.(13).
      end if
    end for
  until a termination condition is met
end
```

Fig. 7. Pseudocode of the IAPSO.



paired comparison on all benchmark functions

η_w	2	5	10	15	20	25	30	40
2	0	-1	-1	-1	0	-1	-1	0
5	1	0	0	0	1	0	0	1
10	1	0	0	0	0	0	0	0
15	1	0	0	0	0	0	0	0
20	0	-1	0	0	0	0	0	0
25	1	0	0	0	0	0	0	0
30	1	0	0	0	0	0	0	0
40	0	-1	0	0	0	0	0	0

Fig. 8. Effect of η_w on fitness error for adventurous UAPSO.

and IAPSO$_{Adv}$. Effect of each parameter for each function has been reported separately and concluded by the results of the multiple comparison tests over all function as a table. The multiple comparison test has been applied using Holm–Bonferroni procedure [43]. Since the results of all paired comparison tests produce enormous amount of data, summary of the comparison tests are provided to compare every pair of PSO algorithms. This summary for every two algorithms PSO$_i$ and PSO$_j$, is the number of functions in which PSO$_i$ outperforms PSO$_j$ minus the number of functions PSO$_i$ is outperformed by PSO$_j$. Therefore, if the result of multiple comparison test for PSO$_i$ and PSO$_j$ is a positive number, it indicates that PSO$_i$ performs better than PSO$_j$ for most of the test function and vice versa.

5.2.1. *Experiment 1: effect of η_w on fitness error for adventurous UAPSO*

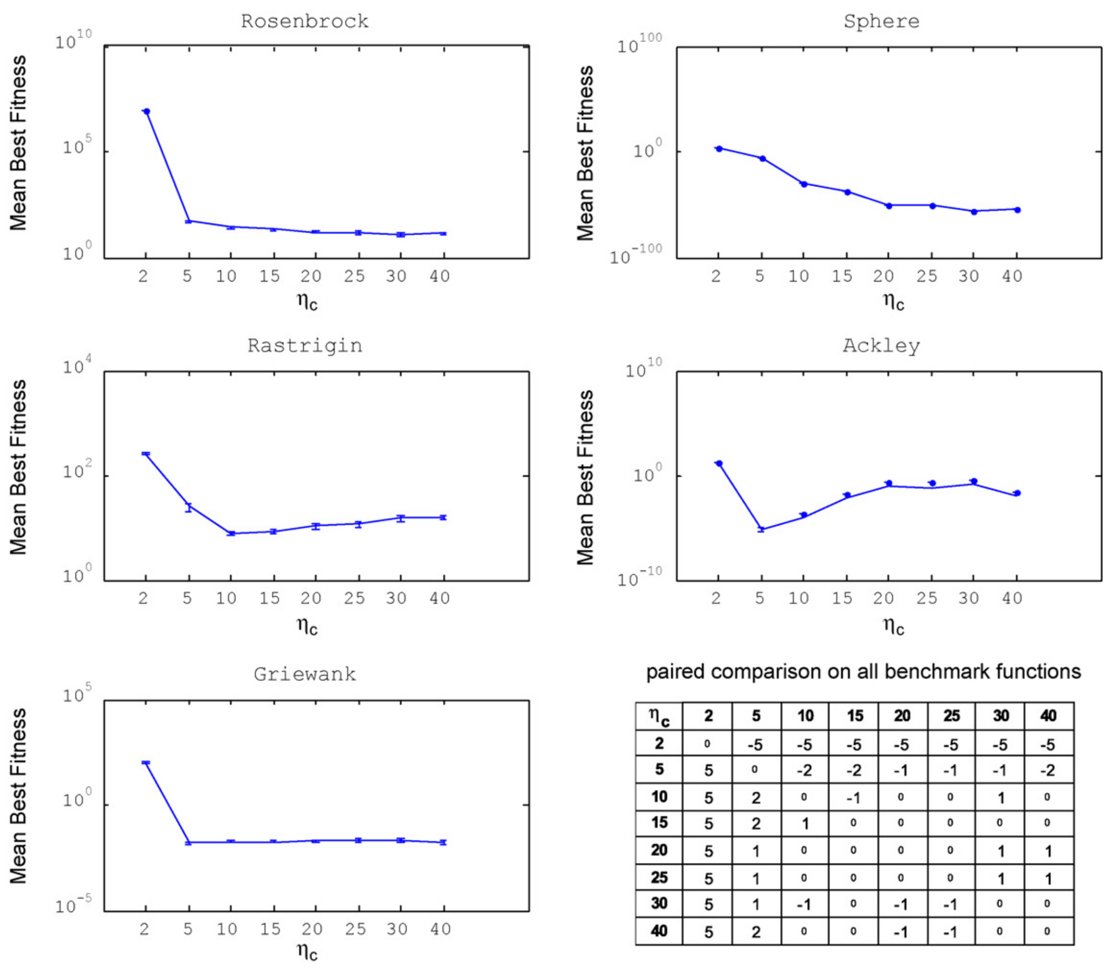
This experiment is conducted to study the effect of η_w, number of discrete value for the inertia weight, on fitness error of UAPSO$_{Adv}$. In order to perform this experiment, fitness error for different values of η_w with fixed value for $\eta_c = 10$ is observed. Fig. 8 shows the results of this experiment on all benchmark functions concluded the comparison table of UAPSO$_{Adv}$ with different η_w. From this figure, we observe that when there are only two values for η_w, UAPSO$_{Adv}$ performs the worst for all function except Rosenbrock and Rastrigin functions. By increasing η_w, fitness error for Sphere and Ackley functions will decrease. However, for Ackley function fitness begins increasing when η_w increases to a value greater than 20. In addition, for Rosenbrock, Rastrigin, and Griewank functions, fitness does not change significantly if η_w is set to a greater value than 5. Hence, as presented in the comparison table, UAPSO$_{Adv}$ with η_w equal to or greater than 5 do not performs significantly different.

5.2.2. *Experiment 2: effect of η_c on fitness error for adventurous UAPSO*

This experiment is conducted to study the effect of η_c, number of discrete value for the cognitive c_1 and social c_2 factor. In order to perform this experiment, fitness error for different values of η_c in UAPSO$_{Adv}$ with fixed value for $\eta_w = 20$ is observed. Fig. 9 shows the results of this experiment on all benchmark functions concluded by the comparison table of UAPSO$_{Adv}$ with different η_c. From this figure, we conclude that if range of c_1 and c_2 are discretized into two values ($\eta_c = 2$) UAPSO$_{Adv}$ cannot perform well for any of the test functions. However, by increasing η_c up to 10 values, fitness error for all functions decreases. This phenomenon continues for unimodal functions. But for multimodal functions, by increasing η_c fitness either increases, for Rastrigin and Ackley, or does not change, for Griewank. The comparison table concludes this experiment and shows that UAPSO$_{Adv}$ with a moderate value for η_c, where η_c is between 10 and 25, performs the best.

5.2.3. *Experiment 3: effect of θ on fitness error for adventurous UAPSO*

This experiment is conducted to study the effect of θ, the success threshold, on the performance of UAPSO$_{Adv}$. In order to perform this experiment, fitness error for different values of θ in UAPSO$_{Adv}$ with fixed value for $\eta_w = 20$ and $\eta_c = 10$ is observed. Fig. 10 shows the results of this experiment on all benchmark functions concluded by the comparison table of UAPSO$_{Adv}$ with different value for θ. From Fig. 10 it can be observed that rigorous success threshold, large value for θ, or relaxed success threshold, small value for θ, increases fitness error. The comparison table supports this hypothesis and show that when θ is around 50% the proposed algorithm perform the best.



η_c	2	5	10	15	20	25	30	40
2	0	-5	-5	-5	-5	-5	-5	-5
5	5	0	-2	-2	-1	-1	-1	-2
10	5	2	0	-1	0	0	1	0
15	5	2	1	0	0	0	0	0
20	5	1	0	0	0	0	1	1
25	5	1	0	0	0	0	1	1
30	5	1	-1	0	-1	-1	0	0
40	5	2	0	0	-1	-1	0	0

Fig. 9. Effect of η_c on fitness error for adventurous UAPSO.

5.2.4. *Experiment 4: effect of δ_w on fitness error for conservative UAPSO*

This experiment is conducted to study the effect of the threshold δ_w, change step of w, on the performance of UAPSO$_{\text{Con}}$. In order to perform this experiment, fitness error for different values of δ_w in UAPSO$_{\text{Con}}$ with fixed value for $\delta_c = 1.0$ is observed. Fig. 11 shows the results of this experiment on all benchmark functions concluded by the comparison table of UAPSO$_{\text{Con}}$ with different value for δ_w. From Fig. 11 it can be observed that for unimodal functions, increasing δ_w from 0.2 to 0.5 causes a temporary decrease in fitness error. Nevertheless, by increasing δ_w to a greater value than 0.5, fitness error will increase. The effect of δ_w for multimodal function is different, where increasing δ_w decreases fitness error. This could be because of the effect of large w that causes more exploration, which helps the particles escape from local minima.

5.2.5. *Experiment 5: effect of δ_c on fitness error for conservative UAPSO*

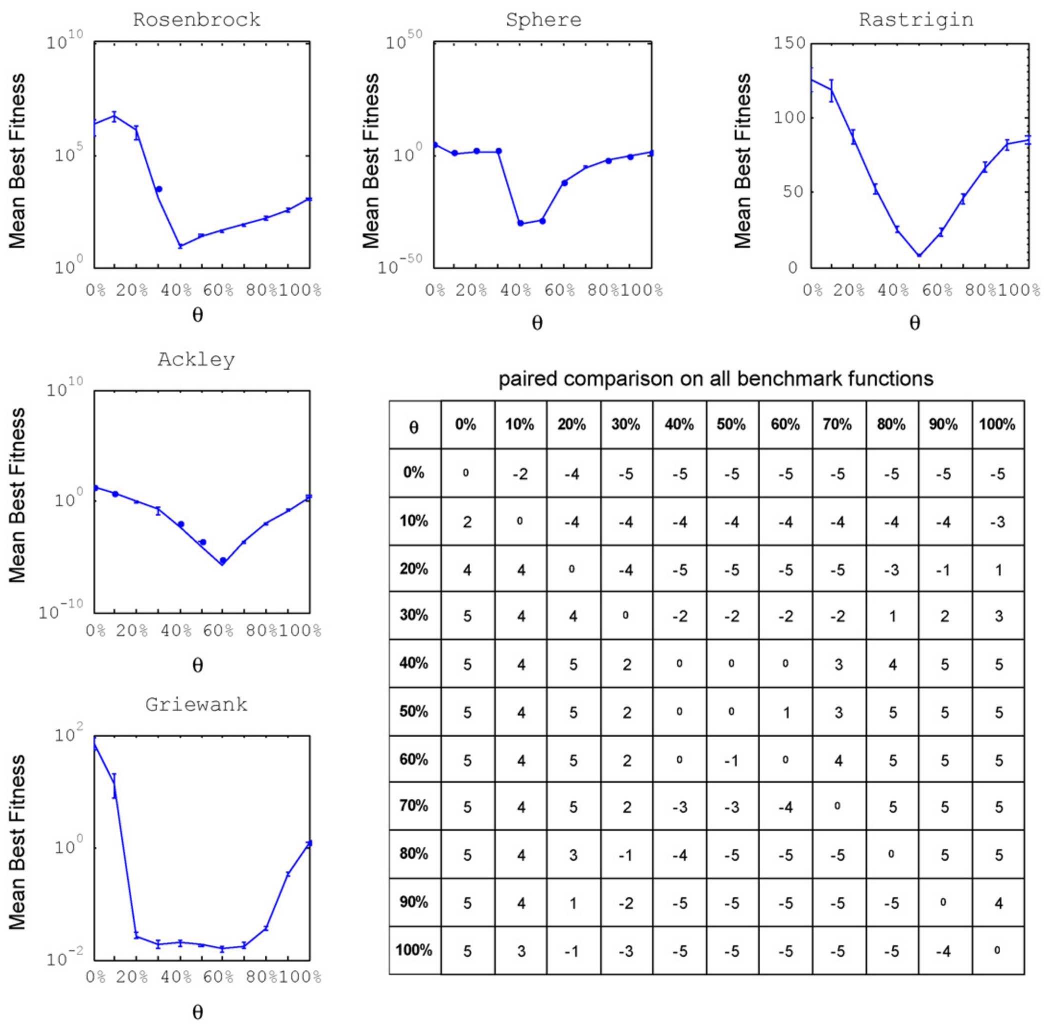
This experiment is conducted to study the effect of the threshold δ_c on performance of UAPSO$_{\text{Con}}$. In order to perform this experiment, fitness error for different values of δ_c in UAPSO$_{\text{Con}}$ with fixed value for $\delta_w = 1$ is observed. Fig. 12 shows the results of this experiment on all benchmark functions concluded by the comparison table of UAPSO$_{\text{Con}}$ with different value for δ_c. From this figure, it can be observed that increasing δ_c up to 1.0 decreases fitness error for all functions. By increasing δ_c from 1.0, fitness error in Sphere, Rosenbrock, and Ackley increases. For Rastrigin increasing δ_c to 1.5 will decrease in fitness error, but setting δ_c to 2.0 causes a significant increase in fitness error. Fitness error for Griewank function shows a different behavior to change of δ_c and decreases slightly by increasing δ_c.

5.2.6. *Experiment 6: effect of θ on fitness error for conservative UAPSO*

This experiment is conducted to study the effect of θ, the success threshold, on the performance of UAPSO$_{\text{Con}}$. In order to perform this experiment, fitness error for different values of θ in UAPSO$_{\text{Con}}$ with fixed value for $\delta_w = \delta_c = 1$ is observed. Fig. 13 shows the results of this experiment on all benchmark functions concluded by the comparison table of UAPSO$_{\text{Con}}$ with different value for θ. Fitness error for each function including with the comparison table show that changing value of θ does not have any significant effect on any of the benchmark functions.

5.2.7. *Experiment 7: effect of η_w on fitness error for adventurous IAPSO*

This experiment is conducted to study the effect of η_w, i.e. number of discretized value for the inertia weight, on fitness error. In order to perform this experiment, fitness error for different values of η_w in IAPSO$_{\text{Adv}}$ with fixed value for $\eta_c = 10$ is observed. Fig. 14 shows the result of this experiment on all benchmark functions concluded by the comparison table of IAPSO$_{\text{Adv}}$ with different value for η_w. From this figure, we observe that not only different values for η_w do not have a significant effect on fitness error



θ	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0%	0	-2	-4	-5	-5	-5	-5	-5	-5	-5	-5
10%	2	0	-4	-4	-4	-4	-4	-4	-4	-4	-3
20%	4	4	0	-4	-5	-5	-5	-5	-3	-1	1
30%	5	4	4	0	-2	-2	-2	-2	1	2	3
40%	5	4	5	2	0	0	0	3	4	5	5
50%	5	4	5	2	0	0	1	3	5	5	5
60%	5	4	5	2	0	-1	0	4	5	5	5
70%	5	4	5	2	-3	-3	-4	0	5	5	5
80%	5	4	3	-1	-4	-5	-5	-5	0	5	5
90%	5	4	1	-2	-5	-5	-5	-5	-5	0	4
100%	5	3	-1	-3	-5	-5	-5	-5	-5	-4	0

Fig. 10. Effect of θ on fitness error for adventurous UAPSO.

of IAPSO$_{Adv}$ for different functions, but also the overall comparison of IAPSO$_{Adv}$ with different η_w do not show a superiority for a specific η_w.

5.2.8. Experiment 8: effect of η_c on fitness error for adventurous IAPSO

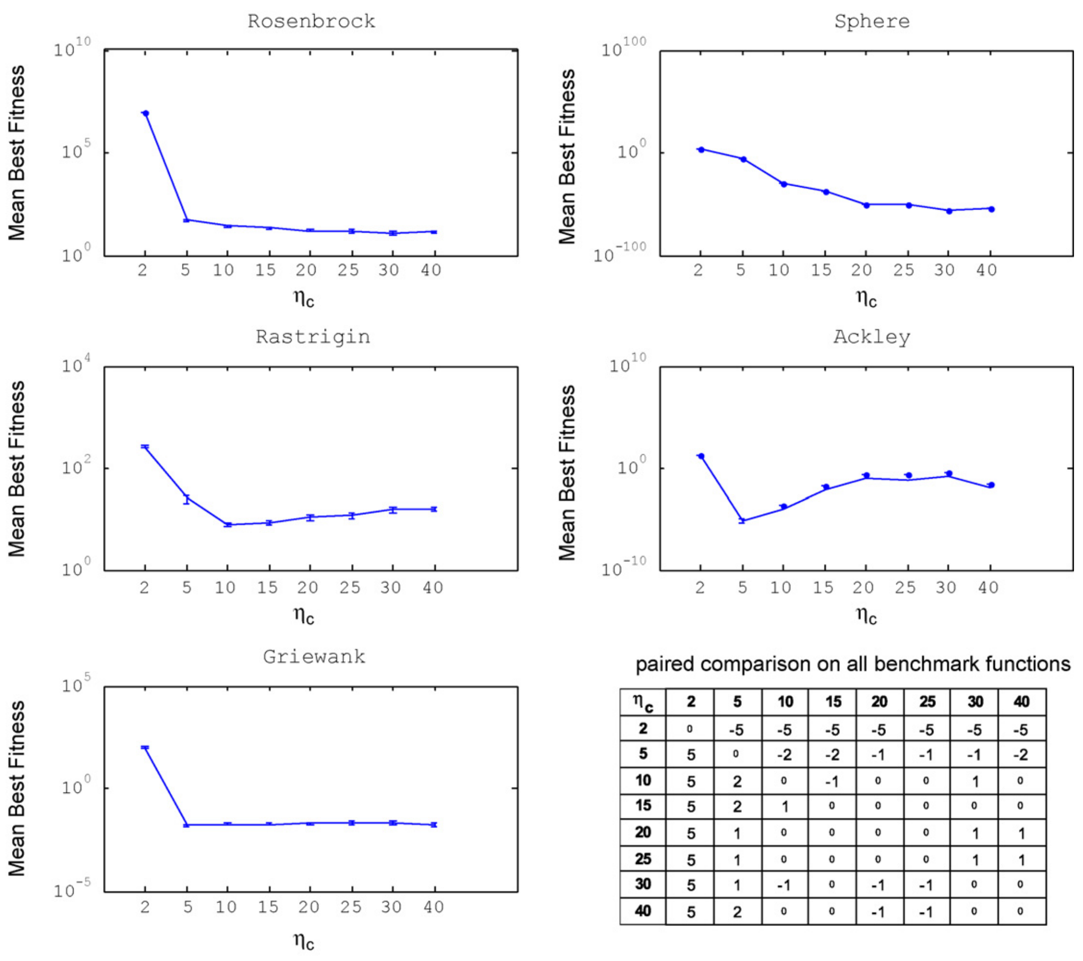
This experiment is conducted to study the effect of number of partitions of the inertia weight, η_c, on fitness error. In order to perform this experiment, fitness error for different values of η_c in IAPSO$_{Adv}$ with fixed value for $\eta_w = 5$ is observed. Fig. 15 shows the results of this experiment on all benchmark functions concluded by the comparison table of IAPSO$_{Adv}$ with different value for η_c. From this figure, we observe that for Rosenbrock, Sphere, and Rastrigin functions, fitness error decreases by increasing η_c. Nevertheless, for Ackley and Griewank fitness error increases slightly as η_c increases. However, since more functions performs better with large values of η_c and the increase in offline error for the Ackley and Griewank is of little importance, the overall comparison for all functions shows that IAPSO$_{Adv}$ performs if η_c is greater than 10.

5.2.9. Experiment 9: effect of δ_w on fitness error for conservative IAPSO

This experiment is conducted to study the effect of the threshold δ_w on performance of IAPSO$_{Con}$. In order to perform this experiment, fitness error for different values of δ_w in IAPSO$_{Con}$ with fixed value for $\delta_c = 1.5$ is observed. Fig. 16 shows the results of this experiment on all benchmark functions concluded by the comparison table of IAPSO$_{Con}$ with different value for δ_w. From this figure, it can be observed that for the unimodal functions, increasing δ_w results a significant increase on fitness error. Nevertheless, in multimodal functions except Rastrigin, fitness error slightly decreases by increasing δ_w. For Rastrigin function fitness error increases when δ_w changes from 0.1 to 0.5, but increasing δ_w from 0.5 to 1.0 causes a significant decrease in fitness.

5.2.10. Experiment 10: effect of δ_c on fitness error for conservative IAPSO

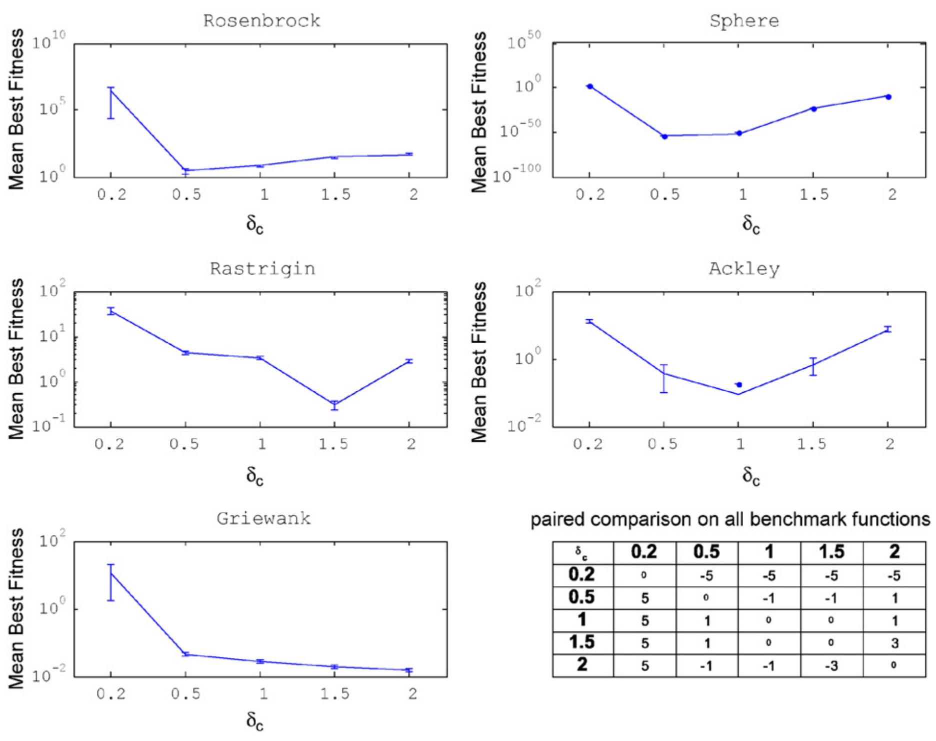
This experiment is conducted to study the effect of the threshold δ_c on performance of IAPSO$_{Con}$. In order to perform this experiment, fitness error for different values of δ_c in IAPSO$_{Con}$ with fixed value for $\delta_w = 1.0$ is observed. Fig. 17 shows the results of this experiment on all benchmark functions concluded by the comparison table of IAPSO$_{Con}$ with different value for δ_c. From this figure, it can be observed that for unimodal function, increasing δ_c from 0.1 to 1 slightly decreases fitness error. But when δ_c increases from 0.5 to 2 fitness error increases significantly. For Rastrigin setting δ_c to 1 results the worst fitness error and by either increasing or decreasing δ_w from 1, fitness error improves. For Ackley function, fitness error slightly improves by increasing δ_w to 1.5, and then begins increasing by changing δ_w to 2.0. Fitness error for Griewank functions improves as δ_w increase.



paired comparison on all benchmark functions

η_c	2	5	10	15	20	25	30	40
2	0	-5	-5	-5	-5	-5	-5	-5
5	5	0	-2	-2	-1	-1	-1	-2
10	5	2	0	-1	0	0	1	0
15	5	2	1	0	0	0	0	0
20	5	1	0	0	0	0	1	1
25	5	1	0	0	0	0	1	1
30	5	1	-1	0	-1	-1	0	0
40	5	2	0	0	-1	-1	0	0

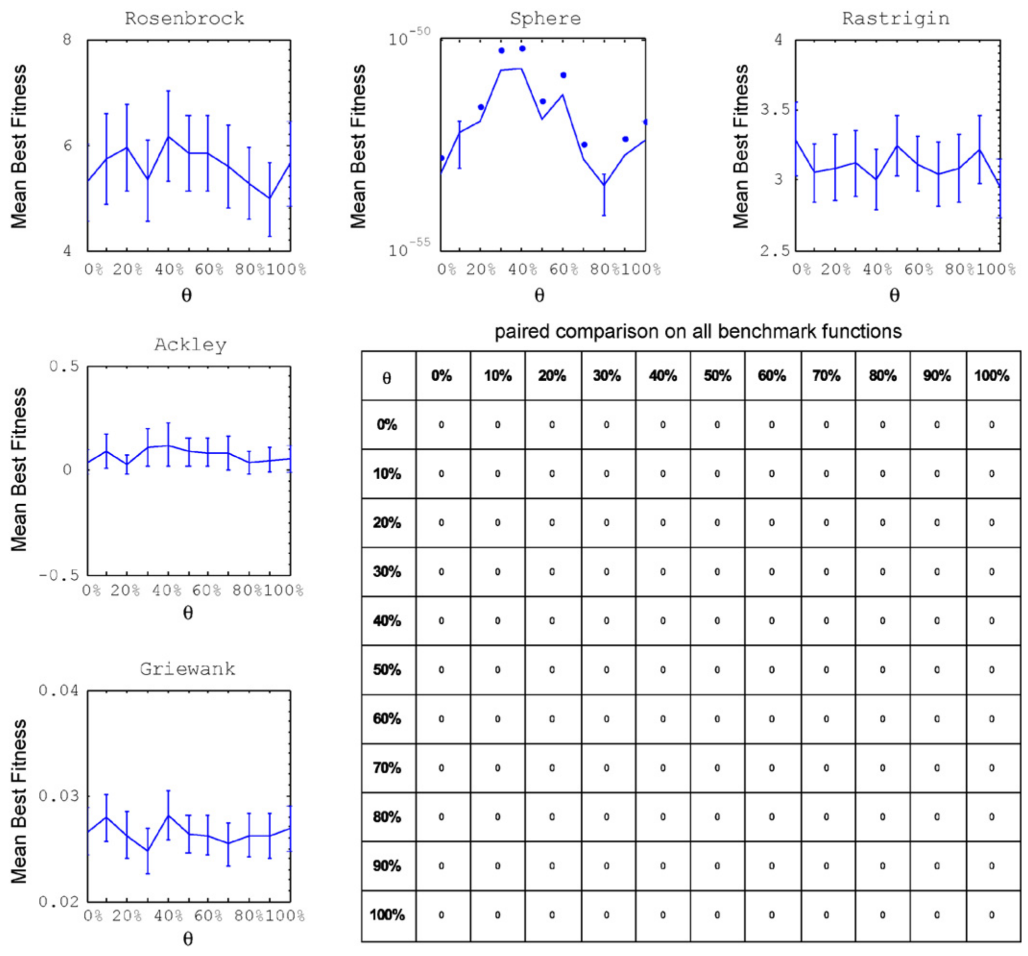
Fig. 11. Effect of δ_w on fitness error for conservative UAPSO.



paired comparison on all benchmark functions

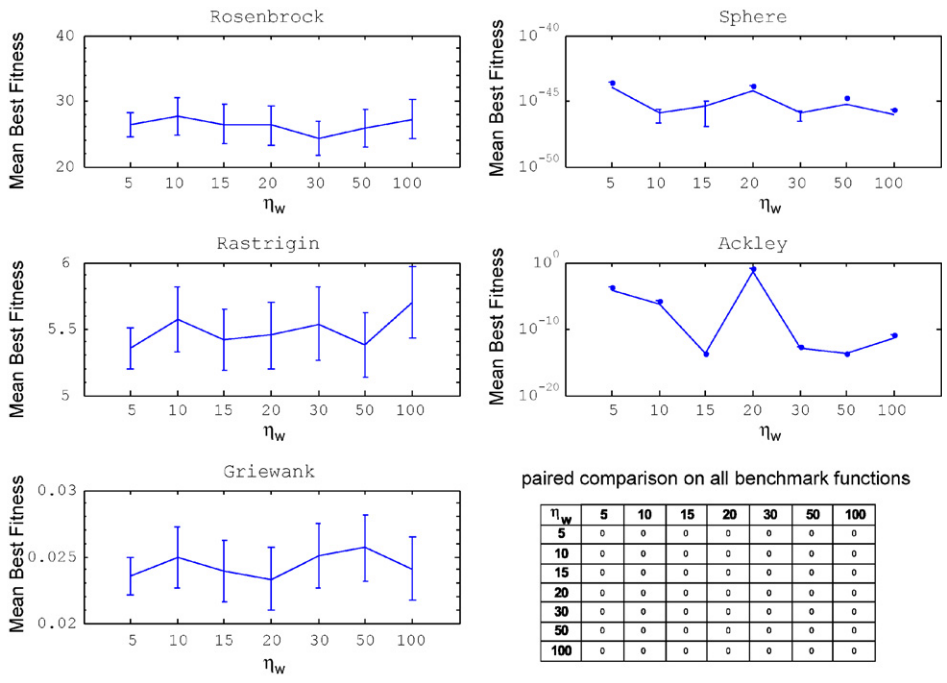
δ_c	0.2	0.5	1	1.5	2
0.2	0	-5	-5	-5	-5
0.5	5	0	-1	-1	1
1	5	1	0	0	1
1.5	5	1	0	0	3
2	5	-1	-1	-3	0

Fig. 12. Effect of δ_c on fitness error for conservative UAPSO.



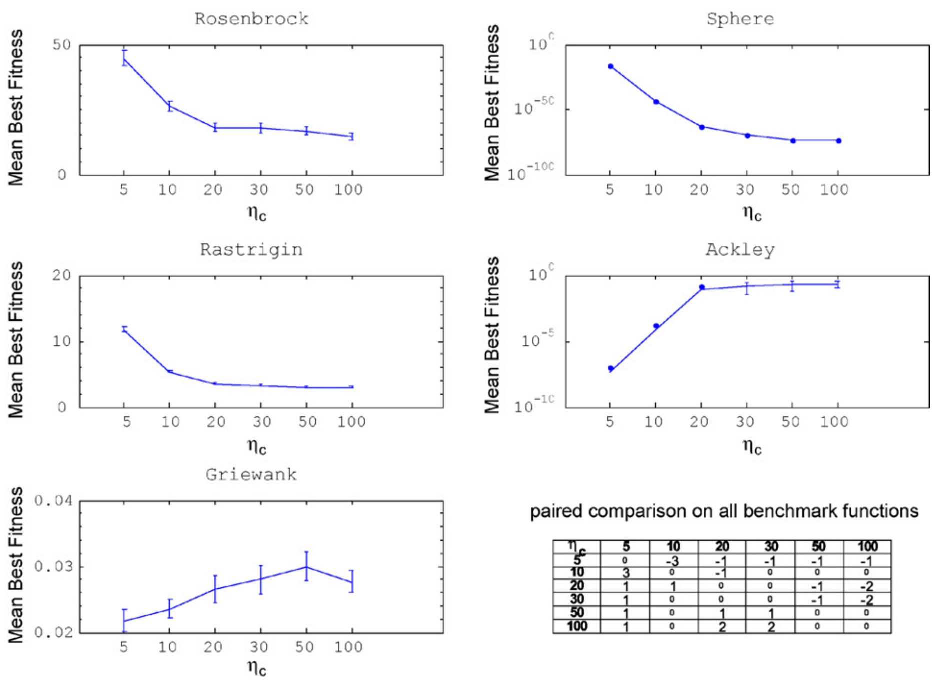
θ	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0%	0	0	0	0	0	0	0	0	0	0	0
10%	0	0	0	0	0	0	0	0	0	0	0
20%	0	0	0	0	0	0	0	0	0	0	0
30%	0	0	0	0	0	0	0	0	0	0	0
40%	0	0	0	0	0	0	0	0	0	0	0
50%	0	0	0	0	0	0	0	0	0	0	0
60%	0	0	0	0	0	0	0	0	0	0	0
70%	0	0	0	0	0	0	0	0	0	0	0
80%	0	0	0	0	0	0	0	0	0	0	0
90%	0	0	0	0	0	0	0	0	0	0	0
100%	0	0	0	0	0	0	0	0	0	0	0

Fig. 13. Effect of θ on fitness error for conservative UAPSO.



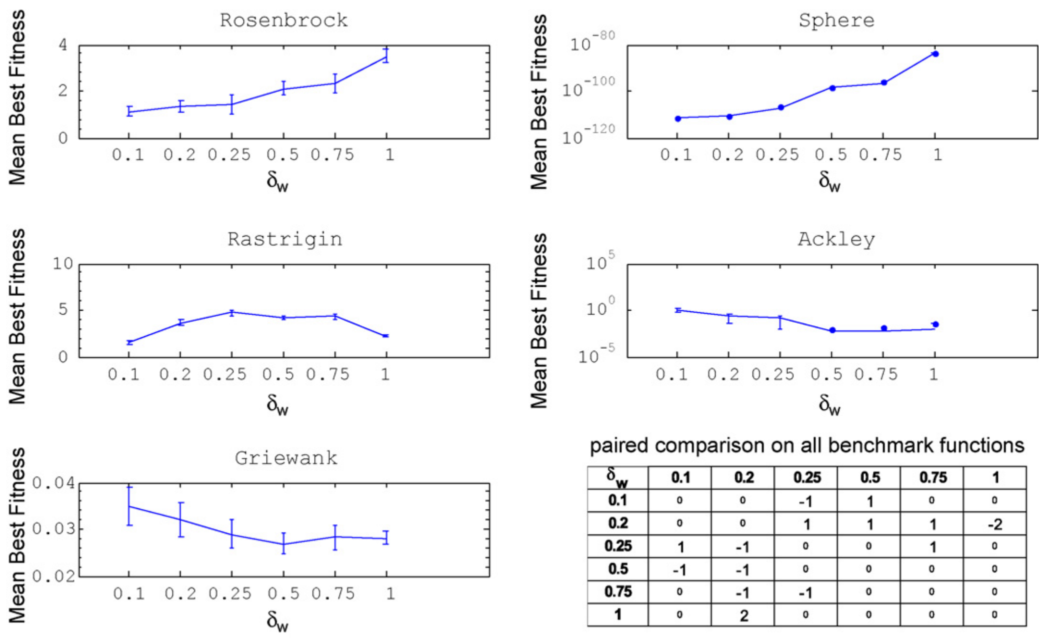
η_w	5	10	15	20	30	50	100
5	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0
100	0	0	0	0	0	0	0

Fig. 14. Effect of η_w on fitness error for adventurous IAPSO.



η_c	5	10	20	30	50	100
5	0	-3	-1	-1	-1	-1
10	3	0	-1	0	0	0
20	1	1	0	0	-1	-2
30	1	0	0	0	-1	-2
50	1	0	1	1	0	0
100	1	0	2	2	0	0

Fig. 15. Effect of η_c on fitness error for adventurous IAPSO.



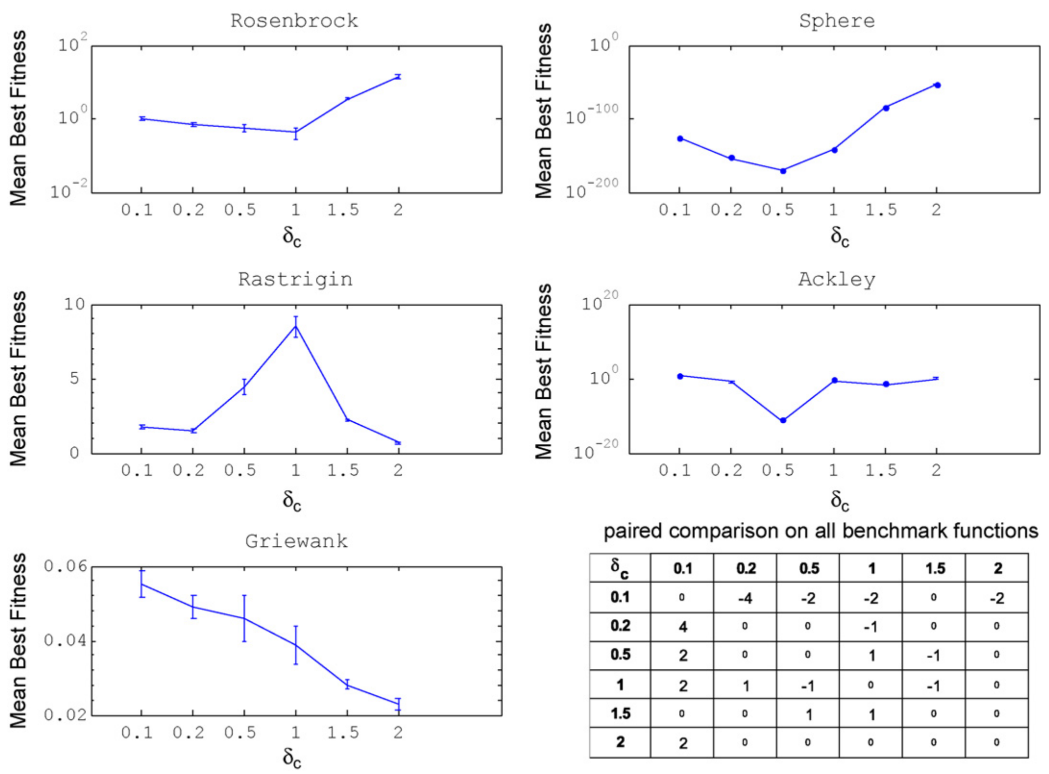
δ_w	0.1	0.2	0.25	0.5	0.75	1
0.1	0	0	-1	1	0	0
0.2	0	0	1	1	1	-2
0.25	1	-1	0	0	1	0
0.5	-1	-1	0	0	0	0
0.75	0	-1	-1	0	0	0
1	0	2	0	0	0	0

Fig. 16. Effect of δ_w on fitness error for conservative IAPSO.

5.3. *Comparison with other PSO algorithms*

In this section, proposed algorithms are compared to well-known PSO algorithms. To do so, for each proposed algorithm a set of parameter value is selected based on the previous experiments. Then four proposed algorithms are compared with the following well-known PSO algorithms: SPSO [30], PSOIW [3], GPSO [37], PSO-TVAC [34], PSOLP [36], DCPSO [48], and DAPSO [49]. The algorithms, which are compared in this section, have been configured as follows.

- UAPSO$_{\mathrm{Adv}}$: $\theta = 50\%$, $\eta_w = 20$, $\eta_c = 10$
- UAPSO$_{\mathrm{Con}}$: $\theta = 60\%$, $\delta_w = 1$, $\delta_c = 1$
- IAPSO$_{\mathrm{Adv}}$: $\eta_w = 5$, $\eta_c = 10$
- IAPSO$_{\mathrm{Con}}$: $\delta_w = 1$, $\delta_c = 1.5$
- SPSO: Standard PSO with $w = 0.72$ and $c_1 = c_2 = 1.49$ as suggested in [27,30]
- PSOIW: PSO with a linearly varying inertia weight PSO [3] $w(0) = 0.9$ and $w(n_t) = 0.4$ [31]
- GPSO: gregarious particle swarm optimizer: $\gamma_{\mathrm{init}} = 3.0$, $\gamma \in [2,4]$, $\delta = 0.5$, and $\varepsilon = 10^{-8}$ [37]
- PSO-TVAC: PSO with time-varying acceleration coefficients: $c_1(0) = 2.5$, $c_1(n_t) = 0.5$, $c_2(0) = 0.5$ $c_2(n_t) = 2.5$, $w(0) = 0.9$, and $w(n_t) = 0.4$ [34]



δ_c	0.1	0.2	0.5	1	1.5	2
0.1	0	-4	-2	-2	0	-2
0.2	4	0	0	-1	0	0
0.5	2	0	0	1	-1	0
1	2	1	-1	0	-1	0
1.5	0	0	1	1	0	0
2	2	0	0	0	0	0

Fig. 17. Effect of δ_c on fitness error for conservative IAPSO.

- PSO-LP: PSO with local parameter: $\alpha_1 = 1.05$, $\alpha_2 = 0.975$, $\beta_1 = 1.07$, $\beta_2 = 0.833$, and *MutationProb* = 0.007 [36]
- DCPSO: $c_1 = c_2 = 2.0$ and $K_p = 1.4$ [48]
- DAPSO: dynamic adaptation PSO: $\alpha = 0.8$, $\beta = 0.4$, and $w(0) = 1.0$ [38]

Proposed algorithms are compared to these PSO algorithms in many aspects. First, a comparison over mean best fitness is presented. Then the results of algorithms from success and convergence speed in reaching a specified goal for each function are presented and compared.

5.3.1. Experiment 11: comparison of mean best fitness

This experiment is conducted to compare mean best fitness of the proposed algorithms with famous PSO algorithms. The mean best fitness for 1,000 runs for over 10,000 iterations is presented in Table 2. However, in order to present a complete comparison for function *T*-tests for each pair of PSO algorithm have been performed. Since the results of *t*-test produces enormous amount of data, we have summarized these comparisons in a ranked list for each function with the following procedure.

First, mean fitness of all PSO algorithms are sorted ascendingly. Then the first PSO with smallest mean error value creates the first. Then all subsequent PSO algorithms, which are not statistically different from the creator of current rank, are added to this rank group. Afterward, the second ranked group is created with the first unranked PSO algorithm. This procedure continues until all PSO algorithms are ranked. The result is a ranked list, in which mean fitness of the PSO algorithms which share the same rank are not statistically different from the least fitness in that rank.

Although none of the proposed algorithms has the least fitness in any function, the proposed algorithms are highly competitive with the PSO algorithms in the literature. Moreover from Table 2 it can be observed that:

1. IAPSO$_{\text{Con}}$ is the best proposed algorithm for the Rosenbrock function. However, it could not rank better than GPSO and PSOLP.
2. All proposed algorithms share the third rank for Sphere function with PSO-TVAC. GPSO performs worse than all proposed algorithm for this function.
3. Both IAPSO algorithms and UAPSO$_{\text{Adv}}$ share the first rank for Ackley function with GPSO. UAPSO$_{\text{Con}}$ with PSO-TVAC follow them in the second rank for this function.
4. UAPSO$_{\text{Adv}}$ performs better than all other adaptive PSO for Griewank function and holds the second rank after PSOIW.
5. Both conservative algorithms perform better than or as well as PSO-TVAC for Rosenbrock, Rastrigin, and Ackley functions.
6. Among the proposed algorithms, IAPSO$_{\text{Con}}$ holds the best average rank.

In order to compare these PSO algorithms over all test functions, Holm–Bonferroni multiple comparison tests has been applied [43]. Result of multiple comparison tests for every pair of PSO algorithms PSO$_i$, PSO$_j$ shows number of functions for which PSO$_i$ outperforms PSO$_j$, denoted by b_{ij}, and number of functions for which PSO$_i$ performs worse than PSO$_j$, denoted by w_{ij}. In Table 3 results of comparison tests is present in the format of $(b, -w)$ over subtraction of b and w. If subtraction of b and w is a positive number, it means that PSO$_i$ outperforms PSO$_j$ in most of the test function and vice versa. From this table we can observe that:

1. IAPSO$_{\text{Con}}$ is the only algorithm which no PSO algorithm could perform better than.
2. IAPSO$_{\text{Con}}$ is the best algorithm among the proposed algorithm and outperforms the other proposed algorithm.
3. UAPSO and IAPSO$_{\text{Adv}}$, are not superior to each other considering all functions.
4. UAPSO and IAPSO$_{\text{Con}}$ outperform PSO-TVAC for most of the test functions and IAPSO$_{\text{Adv}}$ does not perform worse than PSO-TVAC.

Table 2
Mean best fitness of PSO algorithms and their ranks.

Algorithm	Function									
	Mean fitness					Mean fitness rank				
	Rosenbrock	Sphere	Rastrigin	Ackley	Griewank	Rosenbrock	Sphere	Rastrigin	Ackley	Griewank
UAPSO$_{Adv}$	25.82 ± 1.39	3.6E−30 ± 5.3E−30	7.7E+00 ± 7.9E−01	7.52E−5 ± **1.4E−4**	1.81E−2 ± 1.0E−3	6	3	5	1	2
UAPSO$_{Con}$	5.82 ± 0.71	4.9E−52 ± 9.3E−52	3.1E+00 ± 2.0E−01	8.51E−2 ± 7.1E−2	2.62E−2 ± 1.9E−3	4	3	3	2	5
IAPSO$_{Adv}$	26.30 ± 1.77	1.1E−44 ± 1.6E−44	5.4E+00 ± 1.5E−01	6.40E−5 ± **1.3E−4**	2.35E−2 ± 1.4E−3	6	3	4	1	4
IAPSO$_{Con}$	3.50 ± 0.30	2.3E−84 ± 2.8E−84	2.2E+00 ± 8.3E−02	1.11E−2 ± **1.9E−2**	2.81E−2 ± 1.4E−3	3	3	2	1	5
PSOIW	8.19 ± 0.83	2.5E−243 ± 0.0E+00	3.2E+01 ± 4.9E−01	1.65E+0 ± 2.9E−1	1.57E−2 ± **9.6E−4**	5	2	8	3	1
PSO-TVAC	7.89 ± 0.81	2.0E−190 ± 0.0E+00	2.9E+01 ± 3.8E−01	6.96E−2 ± 4.9E−2	1.99E−2 ± 1.1E−3	5	3	7	2	3
GPSO	0.17 ± **0.04**	8.6E−18 ± 1.5E−19	2.5E−15 ± **1.6E−16**	2.74E−9 ± **2.9E−11**	5.86E−2 ± 2.4E−3	1	4	1	1	8
DCPSO	70.72 ± 6.72	1.1E−05 ± 1.8E−06	2.5E+01 ± 1.0E+00	4.78E+0 ± 9.6E−1	3.19E−2 ± 3.8E−3	7	5	6	4	6
PSO-LP	1.32 ± 0.47	7.3E−292 ± **0.0E+00**	1.1E+02 ± 3.4E+00	1.98E+1 ± 7.8E−3	4.76E−2 ± 6.8E−3	2	1	10	5	7
DAPSO	714.423 ± 49.1	5.0E+00 ± 1.9E−01	5.1E+01 ± 9.6E−01	4.26E+0 ± 3.8E−1	1.03E+0 ± 3.0E−3	8	6	9	4	9

Table 3
Pair comparison of mean best fitness.

Algorithm	Function									
	UAPSO$_{Adv}$	UAPSO$_{Con}$	IAPSO$_{Adv}$	IAPSO$_{Con}$	PSOIW	PSO-TVAC	GPSO	DCPSO	PSO-LP	DAPSO
UAPSO$_{Adv}$	(0,0) **0**	(2,−2) **0**	(1,−1) **0**	(1,−2) **−1**	(2,−2) **0**	(3,−1) **2**	(2,−2) **0**	(5,0) **5**	(3,−1) **2**	(5,0) **5**
UAPSO$_{Con}$	(2,−2) **0**	(0,0) **0**	(2,−2) **0**	(0,−2) **−2**	(3,−1) **2**	(2,−1) **1**	(2,−3) **−1**	(5,0) **5**	(3,−1) **2**	(5,0) **5**
IAPSO$_{Adv}$	(1,−1) **0**	(2,−2) **0**	(0,0) **0**	(1,−2) **−1**	(2,−2) **0**	(2,−2) **0**	(2,−2) **0**	(5,0) **5**	(3,−1) **2**	(5,0) **5**
IAPSO$_{Con}$	(2,−1) **1**	(2,0) **2**	(2,−1) **1**	(0,0) **0**	(3,−1) **2**	(3,−1) **2**	(2,−2) **0**	(5,0) **5**	(3,−1) **2**	(5,0) **5**
PSOIW	(2,−2) **0**	(1,−3) **−2**	(2,−2) **0**	(1,−3) **−2**	(0,0) **0**	(2,−2) **0**	(2,−3) **−1**	(4,−1) **3**	(3,−2) **1**	(5,0) **5**
PSO-TVAC	(1,−3) **−2**	(1,−2) **−1**	(2,−2) **0**	(1,−3) **−2**	(2,−2) **0**	(0,0) **0**	(2,−3) **−1**	(4,−1) **3**	(3,−2) **1**	(5,0) **5**
GPSO	(2,−2) **0**	(3,−2) **1**	(2,−2) **0**	(2,−2) **0**	(3,−2) **1**	(3,−2) **1**	(0,0) **0**	(4,−1) **3**	(3,−2) **1**	(5,0) **5**
DCPSO	(0,−5) **−5**	(0,−5) **−5**	(0,−5) **−5**	(0,−5) **−5**	(1,−4) **−3**	(1,−4) **−3**	(1,−4) **−3**	(0,0) **0**	(3,−2) **1**	(4,0) **4**
PSO-LP	(1,−3) **−2**	(1,−3) **−2**	(1,−3) **−2**	(1,−3) **−2**	(2,−3) **−1**	(2,−3) **−1**	(2,−3) **−1**	(2,−3) **−1**	(0,0) **0**	(3,−2) **1**
DAPSO	(0,−5) **−5**	(0,−5) **−5**	(0,−5) **−5**	(0,−5) **−5**	(0,−5) **−5**	(0,−5) **−5**	(0,−5) **−5**	(0,−4) **−4**	(2,−3) **−1**	(0,0) **0**

Table 4
Mean success rate.

Function Algorithm	Rosenbrock	Sphere	Rastrigin	Ackley	Griewank
UAPSO$_{Adv}$	94.1%	**100.0%**	92.9%	99.0%	99.4%
UAPSO$_{Con}$	99.5%	**100.0%**	**100.0%**	21.2%	97.0%
IAPSO$_{Adv}$	90.0%	**100.0%**	**100.0%**	98.0%	98.0%
IAPSO$_{Con}$	99.0%	**100.0%**	**100.0%**	18.9%	96.5%
PSOIW	99.8%	**100.0%**	**100.0%**	75.8%	**99.8%**
PSO-TVAC	99.9%	**100.0%**	**100.0%**	78.1%	99.2%
GPSO	**100.0%**	**100.0%**	**100.0%**	**100.0%**	79.1%

Table 5
Mean number of iterations to reach goal for successful runs.

Function Algorithm	Rosenbrock	Sphere	Rastrigin	Ackley	Griewank
UAPSO$_{Adv}$	55,703 ± 2,329	38,907 ± 1,014	31,393 ± 1,297	**44,213** ± **931**	36,261 ± 880
UAPSO$_{Con}$	39,077 ± 1,498	28,033 ± 309	28,462 ± 682	99,448 ± 7,357	24,230 ± 355
IAPSO$_{Adv}$	47,838 ± 1,815	22,931 ± 147	20,947 ± 713	**43,285** ± **527**	22,779 ± 502
IAPSO$_{Con}$	34,556 ± 1,745	18,804 ± 107	25,830 ± 414	132,126 ± 5,235	**17,063** ± **381**
PSOIW	42,792 ± 1,307	38,187 ± 143	14,571 ± 242	33,535 ± 367	36,385 ± 156
PSO-TVAC	28,736 ± 988	22,193 ± 100	**11,433** ± **206**	**22,249** ± **221**	21,177 ± 156
GPSO	**13,352** ± **780**	**5,659** ± **58**	17,144 ± 514	90,460 ± 1,624	22,863 ± 2,189

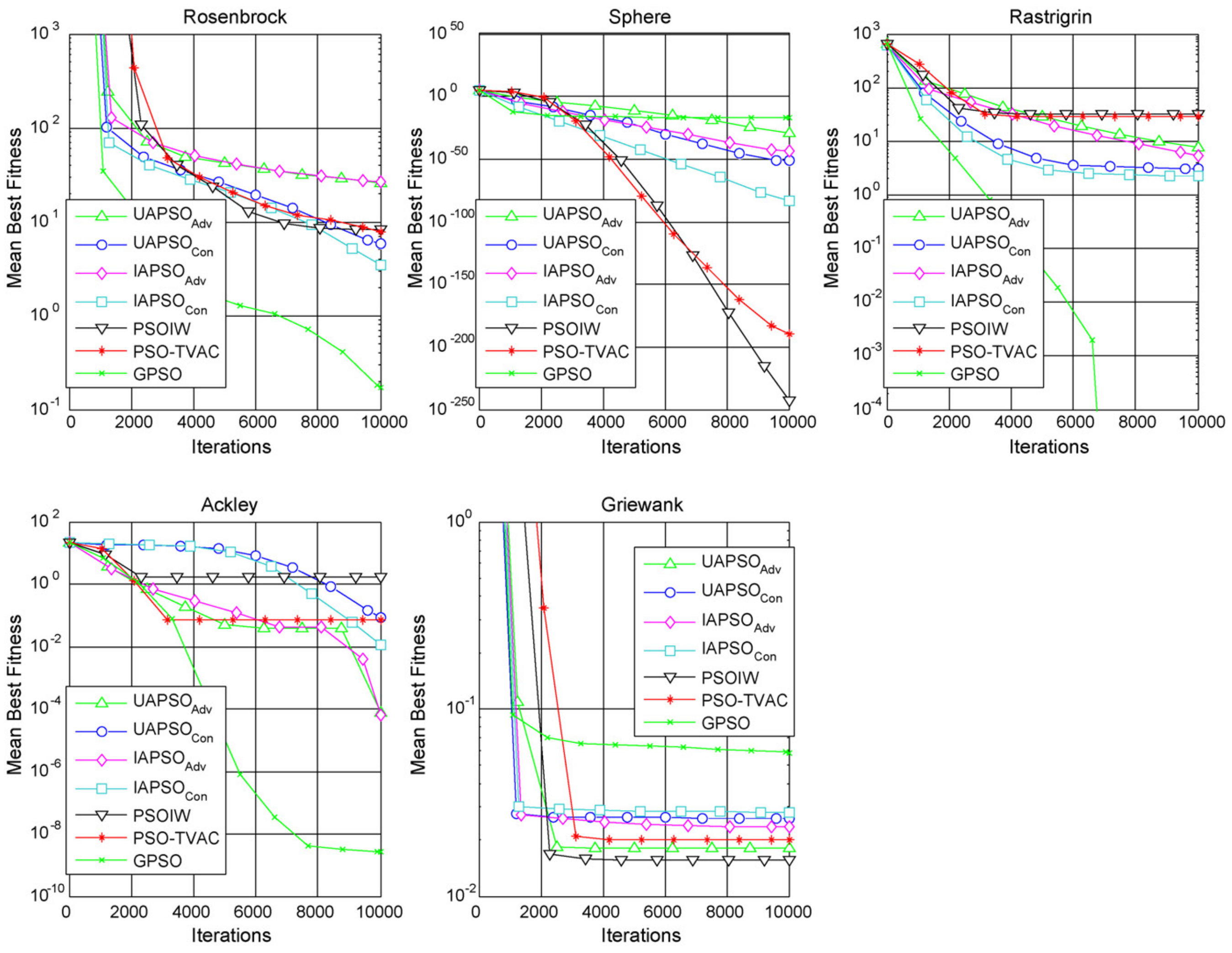


Fig. 18. Mean best fitness over time.

5. Although GPSO performs very well in most of the tested functions, it cannot outperform any of the IAPSO and $UAPSO_{Adv}$.
6. Although PSOIW could outperform the conservative algorithms in one function, the conservative algorithms are better that PSOIW for three functions. However, either of the adventurous algorithms and PSOIW outperforms the other one in two functions, hence, none of them has superiority over the other.
7. All proposed algorithms are superior to DCPSO, PSO-LP, and DAPSO in most functions.

5.3.2. *Experiment 12: comparison of convergence to global optimum*

In this experiment, first, success rate of each algorithm in reaching the specified goal for each function is studied. Success rate of a PSO is defined as the percentage of successful runs of the PSO. A successful run is a run in which the PSO reaches the specified goal of the function. Second, we measure the speed of convergence to the goal for successful runs of each function in terms of number of iterations required to reach the specified goal. The results presented in this section are average of 1000 runs of the algorithms over 5000 iterations.

The success rates of the PSO algorithms are presented in Table 4. It can be observed that:

1. Although proposed algorithms are more than 90% successful for Rosenbrock function, the conservative algorithms are more successful.
2. All algorithms reach the specified goal for Sphere function in all runs.
3. All algorithms except $UAPSO_{Adv}$ can reach the specified goal for Rastrigin function in all runs. However, $UAPSO_{Adv}$ were successful in 92% of runs.
4. The proposed adventurous algorithms are the second most successful algorithms for Ackley function after GPSO.
5. The conservative algorithms are not successful in most experiments for Ackley function.
6. All the proposed algorithms are successful in more than 96% of runs for Griewank function. Moreover, $UAPSO_{Adv}$ is the second most successful algorithm for Griewank function.

From Table 5 it can be observed that:

1. $IAPSO_{Con}$ is the second fastest algorithm for Sphere function after GPSO.
2. Among the successful algorithms for Ackley function, the proposed adventurous algorithms are twice as fast as the second fastest algorithm, GPSO.
3. $IAPSO_{Adv}$ is the fastest algorithms for Griewank function.
4. Although all proposed algorithms are successful for Rosenbrock and Rastrigin function, they are slower than GPSO and PSO-TVAC (Fig. 18).

6. Conclusion

In this paper, we studied the ability of learning automata based schemes to adjust parameters of PSO. Two classes of algorithms are proposed. In the first class, the same parameters are set for all particles while in the second class each particle adjusts its parameters individually. In addition, for both classes of proposed algorithms, two approaches for changing value of parameters were taken. In the first approach, named adventurous, value of a parameter is selected from a finite set while in the second approach, named conservative, value of a parameter either changes by a fixed amount or remain the same value. Experimental results showed that proposed algorithms are highly competitive with well-known PSO algorithms such as SPSO, PSOIW, DCPSO, DAPSO, PSOLP, PSO-TAVC, and GPSO in all test functions. Although the proposed algorithms could not perform very well on each function separately, comparison of PSO algorithms on all functions showed that they could outperform all PSO algorithms in terms of number of function in which they are superior. Moreover, proposed algorithms could reach the goal of highly multi modal functions, Ackley and Griewank, faster than most of other PSO algorithms with the same success rate. Furthermore, studies the effect of parameters of proposed algorithms demonstrate that proposed algorithms are able to perform very well with the same parameters for different functions.

References

[1] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: IEEE International Conference on Neural Networks, Piscataway, NJ, USA, 1995, pp. 1942–1948.
[2] B. Niu, Y. Zhu, K. Hu, S. Li, X. He, A novel particle swarm optimizer using optimal foraging theory, Computational Intelligence and Bioinformatics (2006) 61–71.
[3] Y. Shi, R.C. Eberhart, A modified particle swarm optimizer, in: IEEE International Conference on Evolutionary Computation, Anchorage, AK, USA, 1998, pp. 69–73.
[4] A. Chatterjee, P. Siarry, Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization, Computers and Operations Research 33 (2006) 859–871.
[5] M. Clerc, J. Kennedy, The particle swarm—explosion, stability, and convergence in a multidimensional complex space, IEEE Transactions on Evolutionary Computation 6 (2002) 58–73.
[6] J Kennedy, Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance, in: Congress on Evolutionary Computation, Piscataway, NJ, USA, 1999, pp. 1931–1938.
[7] J. Kennedy, R. Mendes, Population structure and particle swarm performance, in: Evolutionary Computation Congress, Honolulu, Hawaii, USA, 2002, pp. 1671–1676.
[8] P.N. Suganthan, Particle swarm optimiser with neighbourhood operator, in: Congress of Evolutionary Computation, Washington, DC, USA, 1999, pp. 1958–1962.
[9] X. Hu, R.C. Eberhart, Multiobjective optimization using dynamic neighborhood particle swarm optimization, in: Congress on Evolutionary Computation, Honolulu, Hawaii, USA, 2002, pp. 1677–1681.
[10] W.J. Zhang, X.F. Xie, DEPSO: hybrid particle swarm with differential evolution operator, in: IEEE International Conference on Systems, Man, and Cybernetics, Washington, DC, USA, 2003, pp. 3816–3821.
[11] C.F. Juang, A hybrid of genetic algorithm and particle swarm optimization for recurrent network design, IEEE Transactions on Systems, Man and Cybernetics, Part B 34 (2004) 997–1006.
[12] X.H. Shi, Y.C. Liang, H.P. Lee, C. Lu, L.M. Wang, An improved GA and a novel PSO-GA-based hybrid algorithm, Information Processing Letters 93 (2005) 255–261.
[13] S. He, Q.H. Wu, J.Y. Wen, J.R. Saunders, R.C. Paton, A particle swarm optimizer with passive congregation, Biosystems 78 (2004) 135–147.
[14] X.F. Xie, W. Zhang, Z. Yang, Hybrid particle swarm optimizer with mass extinction, in: International Conference on Communication, Circuits and Systems, Chengdu, China, 2002, pp. 1170–1173.
[15] M. Løvbjerg, T.K. Rasmussen, T. Krink, Hybrid particle swarm optimiser with breeding and subpopulations, in: Third Genetic and Evolutionary Computation Conference, San Francisco, CA, USA, 2001, pp. 469–476.
[16] R. Poli, C.D. Chio, W.B. Langdon, Exploring extended particle swarms: a genetic programming approach, in: 2005 Conference on Genetic and Evolutionary Computation, ACM, Washington, DC, USA, 2005, pp. 169–176.
[17] M.S. Arumugam, M.V.C. Rao, On the improved performances of the particle swarm optimization algorithms with adaptive parameters, cross-over operators and root mean square (RMS) variants for computing optimal control of a class of hybrid systems, Applied Soft Computing 8 (2008) 324–336.
[18] B Niu, Y.L. Zhu, X.X. He, Construction of fuzzy models for dynamic systems using multi-population cooperative particle swarm optimizer, in: Fuzzy Systems and Knowledge Discovery, Springer-Verlag, 2005, pp. 987–1000.
[19] B. Niu, Y.L. Zhu, X.X. He, Multi-population cooperative particle swarm optimization, in: Advances in Artificial Life, Springer-Verlag, 2005, pp. 874–883.
[20] F. van den Bergh, A. Engelbrecht, A cooperative approach to particle swarm optimization, IEEE Transactions on Evolutionary Computation 8 (2004) 225–239.
[21] H. Beigy, M.R. Meybodi, An adaptive call admission algorithm for cellular networks, Journal of Computer and Electrical Engineering 31 (2005) 132–151.
[22] H. Beigy, M.R. Meybodi, Call admission control in cellular mobile networks: a learning automata approach, in: EurAsia-ICT 2002: Information and Communication Technology, 2002, pp. 450–457.
[23] B.J. Oommen, T.D. Roberts, Continuous learning automata solutions to the capacity assignment problem, IEEE Transactions on Computers 49 (2000) 608–620.
[24] M.R. Meybodi, H. Beigy, A note on learning automata based schemes for adaptation of bp parameters, Journal of Neurocomputing 48 (2002) 957–974.
[25] H. Beigy, M.R. Meybodi, A learning automata based algorithm for determination of the number of hidden units for three layers neural networks, International Journal of Systems Science 40 (2009) 101–118.

[26] J. Kennedy, The behavior of particles, in: 7th International Conference on Evolutionary Programming VII, San Diego, CA, USA, 1998, pp. 581–589.
[27] M. Clerc, The swarm and the queen: toward a deterministic and adaptive particle swarm optimization, in: Congress on Evolutionary Computation, Washington, DC, USA, 1999, pp. 1951–1957.
[28] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, 1995, pp. 39–43.
[29] A. Carlisle, G. Dozier, An off-the-shelf PSO, in: Workshop on Particle Swarm Optimization, Indianapolis, IN, USA, 2001, pp. 1–6.
[30] R.C. Eberhart, Y. Shi, Comparing inertia weights and constriction factors in particle swarm optimization, in: Congress on Evolutionary Computation, San Diego, CA, USA, 2000, pp. 84–88.
[31] Y. Shi, R.C. Eberhart, Parameter selection in particle swarm optimization, in: 7th International Conference on Evolutionary Programming VII, 1998, pp. 591–600.
[32] Y. Shi, R.C. Eberhart, Empirical study of particle swarm optimization, in: IEEE Congress on Evolutionary Computation, Carmel, IN, USA, 1999, pp. 101–106.
[33] A. Abraham, H. Guo, H. Liu, Swarm intelligence: foundations, perspectives and applications, in: A. Abraham, C. Grosan, V. Ramos (Eds.), Swarm Intelligence: Foundations, Perspectives and Applications, Studies in Computational Intelligence, Springer, Germany, 2006, pp. 3–25.
[34] A. Ratnaweera, S.K. Halgamuge, H.C. Watson, Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients, IEEE Transactions on Evolutionary Computation 8 (2004) 240–255.
[35] S.-K. Fan, J.-M. Chang, A modified particle swarm optimizer using an adaptive dynamic weight scheme, in: Digital Human Modeling, 2007, pp. 56–65.
[36] P. Tawdross, A. Konig, Local parameters particle swarm optimization, in: Sixth International Conference on Hybrid Intelligent Systems (HIS'06), Auckland, New Zealand, 2006, pp. 52–55.
[37] S. Pasupuleti, R. Battiti, The gregarious particle swarm optimizer (G-PSO), in: 8th annual conference on Genetic and evolutionary computation, ACM, Seattle, WA, USA, 2006, pp. 67–74.
[38] X. Yanga, J. Yuana, J. Yuana, H. Maob, A modified particle swarm optimizer with dynamic adaptation, Applied Mathematics and Computation 189 (2007) 1205–1213.
[39] M. Thathachar, P. Sastry, Varieties of learning automata: an overview, IEEE Transactions on Systems, Man and Cybernetics 32 (2002) 711–722.
[40] S. Janson, M. Middendorf, A hierarchical particle swarm optimizer and its adaptive variant, IEEE Transactions on Systems, Man and Cybernetics, Part B 35 (2005) 1272–1282.
[41] F. van den Bergh, A.P. Engelbrecht, Effects of swarm size on cooperative particle swarm optimizers, in: Genetic and Evolutionary Computation Conference, San Francisco, CA, USA, 2001, pp. 892–899.
[42] R. Mendes, J. Kennedy, J. Neves, The fully informed particle swarm: simpler, maybe better, IEEE Transactions of Evolutionary Computation 8 (2004) 204–210.
[43] D Bratton, J. Kennedy, Defining a standard for particle swarm optimization, in: IEEE Swarm Intelligence Symposium, Honolulu, Hawaii, USA, 2007, pp. 120–127.
[44] R. Mendes, J. Kennedy, J. Neves, Avoiding the pitfalls of local optima: how topologies can save the day, in: 12th Conference Intelligent Systems Application to Power Systems (ISAP2003), Lemnos, Greece, 2003.
[45] D.B. Fogel, H.-G. Beyer, A note on the empirical evaluation of intermediate recombination, Evolutionary Computation 3 (1995) 491–495.
[46] P.J. Angeline, Using selection to improve particle swarm optimization, in: IEEE International Conference on Evolutionary Computation, Anchorage, AK, USA, 1998, pp. 84–89.
[47] P. Angeline, Evolutionary optimization versus particle swarm optimization: philosophy and performance differences, Evolutionary Programming VII (1998) 601–610.
[48] J. Jie, J. Zeng, C. Han, Adaptive particle swarm optimization with feedback control of diversity, Computational Intelligence and Bioinformatics (2006) 81–92.
[49] X. Yang, J. Yuan, J. Yuan, H. Mao, A modified particle swarm optimizer with dynamic adaptation, Applied Mathematics and Computation 189 (2007) 1205–1213.