International Journal of the Physical Sciences Vol. 6(22), pp. 5121-5127, 2 October, 2011
Available online at http://www.academicjournals.org/IJPS
DOI: 10.5897/IJPS11.263
ISSN 1992 - 1950 ©2011 Academic Journals

Full Length Research Paper

An innovative combination of particle swarm optimization, learning automaton and great deluge algorithms for dynamic environments

Hamid Parvin[1]*, Behrouz Minaei[2], Hamid Alinejad-Rokny[3] and Sajjad Ghatei[4]

[1]Islamic Azad University, Nourabad Mamasani Branch, Nourabad, Iran
[2]School of Computer Engineering, Iran University of Science and Technology (IUST), Tehran, Iran
[3]Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran
[4]Islamic Azad University, Nourabad Mamasani Branch, Nourabad, Iran

Accepted 3 August, 2011

Dynamic optimization in which global optima and local optima change over time is always a hot research topic. It has been shown that particle swarm optimization works well when facing dynamic environments. On the other hand, a learning automaton can be considered as an intelligent tool (agent) which can learn what action is the best interacting with its environment. The great deluge algorithm is also a search algorithm applied to optimization problems. All these algorithms have their drawbacks and advantages. This paper explores how one can combine these algorithms to reach better performance in dynamic spaces. Indeed a learning automaton is employed per particle in the swarm to decide whether its particle updates its velocity (and consequently its position) considering the best global particle position, local particle position or a combined position extracted out of global and local particle position. Water level in the deluge algorithm is used in the progress of the algorithm. Experimental results on different dynamic environments modeled by moving peaks benchmark show that the combination of these algorithms outperforms PSO algorithm, fast multi-swarm method (FMSO), a similar particle swarm algorithm for dynamic environments, for all tested environments.

Key words: Particle swarm optimization, great deluge, learning automaton, moving peaks, dynamic environments.

INTRODUCTION

The standard particle swarm optimization (PSO) algorithms have performed well for static environment. Also, it is shown that the original PSO is not able to handle dynamic environments. So researchers turn to new variations of PSO to overcome its inefficiency.

Hu and Eberhart (2002) proposed a re-randomization PSO for optimization in dynamic environments in which some particles are randomly relocated after a change is detected or when the diversity is lost, to prevent losing the diversity. Li and Dam (2003) showed that a grid-like neighborhood structure used in fine grain particle swarm optimization (FGPSO) (Kennedy and Mendes, 2002) can perform better than re-randomization particle swarm optimization (RPSO) in high dimensional dynamic environments by restricting the information sharing and preventing the convergence of particles to the global best position, thereby enhancing population diversity. Janson and Middendorf (2004) proposed hybrid particle swarm optimization (HPSO), a tree-like structure hierarchical PSO, and reported improvements over standard PSO for dynamic environments. They also suggested partitioned hierarchical PSO in which a hierarchy of particles is partitioned into several sub-swarms for a limited number of generations after a change in the environment is detected (2006). Lung and Dumitresc (2007) used two collaborating populations with same size. In their work, one swarm is responsible for preserving the diversity of the particles by using a crowding differential evolutionary algorithm (Thomsen, 2004) while the other keeps track of global optimum with a PSO algorithm.

Li and Yang (2008) proposed a fast multi-swarm method (FMSO) which maintains the diversity through the run. To meet this goal, two types of swarm are used: a parent swarm which maintains the diversity and detects

*Corresponding author. E-mail: parvin@iust.ac.ir.

the promising search area in the whole search space using a fast evolutionary programming algorithm and a group of child swarms which explore the local area for the local optima found by the parent using a fast PSO algorithm. This mechanism makes the child swarms spread out over the highest multiple peaks, as much as possible, and guarantees to converge to a local optimum in a short time. Moreover, Li and Yang (2009) introduced a clustering particle swarm optimizer in which a clustering algorithm partitions the swarm into several sub-swarms each searching for a local optimum.

Liu et al. (2008) introduced compound particle swarm optimization (CPSO) utilizing a new type of particle which helps explore the search space more comprehensively after a change occurred in the environment. In another work, they used composite particles which help to quickly find the promising optima in the search space while maintaining the diversity by a scattering operator (Liu et al., 2010).

Hashemi and Meybodi (2009) introduced cellular PSO, a hybrid model of cellular automata and PSO. In cellular PSO, a cellular automaton partitions the search space into cells. At any time, in some cells of the cellular automaton, a group of particles search for a local optimum using their best personal experiences and the best solution found in their neighborhood cells. To prevent losing the diversity, a limit on the number of particles in each cell is imposed. Furthermore, to track the changes in the environment (Hashemi and Meybodi, 2009), particles in cellular PSO change their role to quantum particles and perform a random search around the previously found optima for a few iterations after a change is detected in the environment.

Kamosi et al. (2010) proposed some variations of PSO that can perform well for dynamic environments. In their work, they proposed a multi-swarm algorithm for dynamic environments which address the diversity loss problem by introducing two types of swarm: a parent swarm, which explores the search space to find promising area containing local optima and several non-overlapping child swarms, each of which is responsible for exploiting a promising area found by the parent swarm.

This paper explores how to combine the PSO, learning automaton (LA) and great deluge algorithms to eliminate their weaknesses in order to reach better performance in dynamic spaces. A learning automaton is employed per particle in PSO to decide whether its particle updates its velocity (and consequently its position) considering the best global particle position, local particle position or a combined position extracted out of global particle position and local particle position. Water level in the deluge algorithm is used in the progress of the algorithm to prevent it from premature convergence of the algorithm.

RELATED WORKS

A learning automaton (LA) is an adaptive decision-making unit situated in a random environment that learns the optimal action through repeated interactions with its environment. The actions are chosen according to a specific probability distribution which is updated based on the response the automaton obtains from the environment by performing a particular action.

Given a finite number of actions that can be performed in a random environment, when a specific action is takes place, the environment provides a random response which is either favorable or unfavorable. The objective in the design of the automaton is to determine how the choice of the action at any stage should be guided by past actions and responses.

The particle swarm optimization algorithm (PSO) was introduced by Kennedy and Eberhart (1995) in their work. In PSO, a potential solution for a problem is considered as a bird, which is called a particle, flies through a D-dimensional space and adjusts its position according to its own experience and other particles'. In PSO, a particle is represented by its position vector *x* and its velocity vector *v*.

The great deluge algorithm (GD) was introduced by Dueck (1993), but unfortunately was not widely useful in succeeding years. This local search meta-heuristic is different to its predecessors (for example, consider hill-climbing or simulated annealing) in the acceptance of a candidate solution from a neighborhood. The GD algorithm accepts all solutions, for which absolute values of the cost function are less than or equal to the current boundary value, called "level". The local search starts with the initial value of "level" equal to an initial cost function and during the search its value is monotonically reduced. A decrement of the reduction (defined by the user) appears as a single algorithmic parameter.

PROPOSED COMBINATION FRAMEWORK OF PSO, LA AND GD ALGORITHMS

To get rid of premature convergence of PSO algorithm, there is need for the use a mechanism that produces perturbation in the population. A very promising method is to turn to multi-swarm mechanisms. In the multi-swarm mechanism, there must be a parent swarm which is responsible for finding promising area in the search space and also some child swarms which are created to exploit the new found promising area (Blackwell and Branke, 2006, 2004; Kennedy et al., 1995; Blackwell et al., 2008).

This paper differently deals with the premature problem. It uses a PSO in which each particle uses a learning automaton based on which particle decides how to update its velocity. Indeed, each automaton learns how to behave in predefined situations. These situations contain variance of the best fitness of local optima and the distance of the particle to its local and global optima. The learning of automaton is based on feedbacks received from the environment. A feedback per each particle is set according to its fitness and the previous position's fitness. It means that if the previous position's fitness is higher than the current position's, the action taken on the previous position to reach current position punishes, else it is rewarded.

Figure 1 depicts the proposed method flowchart for a particle P_i ($i \in 1..N$) in the swarm. In Figure 1, A_i stands for automaton related to ith particle whose position is denoted by x_i. X_g stands for global best position so far and X^i_l stands for best position found by P_i so far. As it is obvious from Figure 1, the automaton A_i makes a decision for particle *i* on how to update its velocity. Based on the position of ith particle, position of other particles, position of global best solution X_g and position of local best solution X_l, the automaton decides the action *a*. Based on the action *a*, the next velocity of the current particle denoted by V'_i is calculated.

After calculating the next position of particle P_i denoted by x'_i, if the fitness reduces (in minimization problems) rather than previous position x_i, the state-action is rewarded, else it is punished. Then, according to the punish-reward signal, the automaton is updated.

After the great deluge we accept the relocation of ith particle if

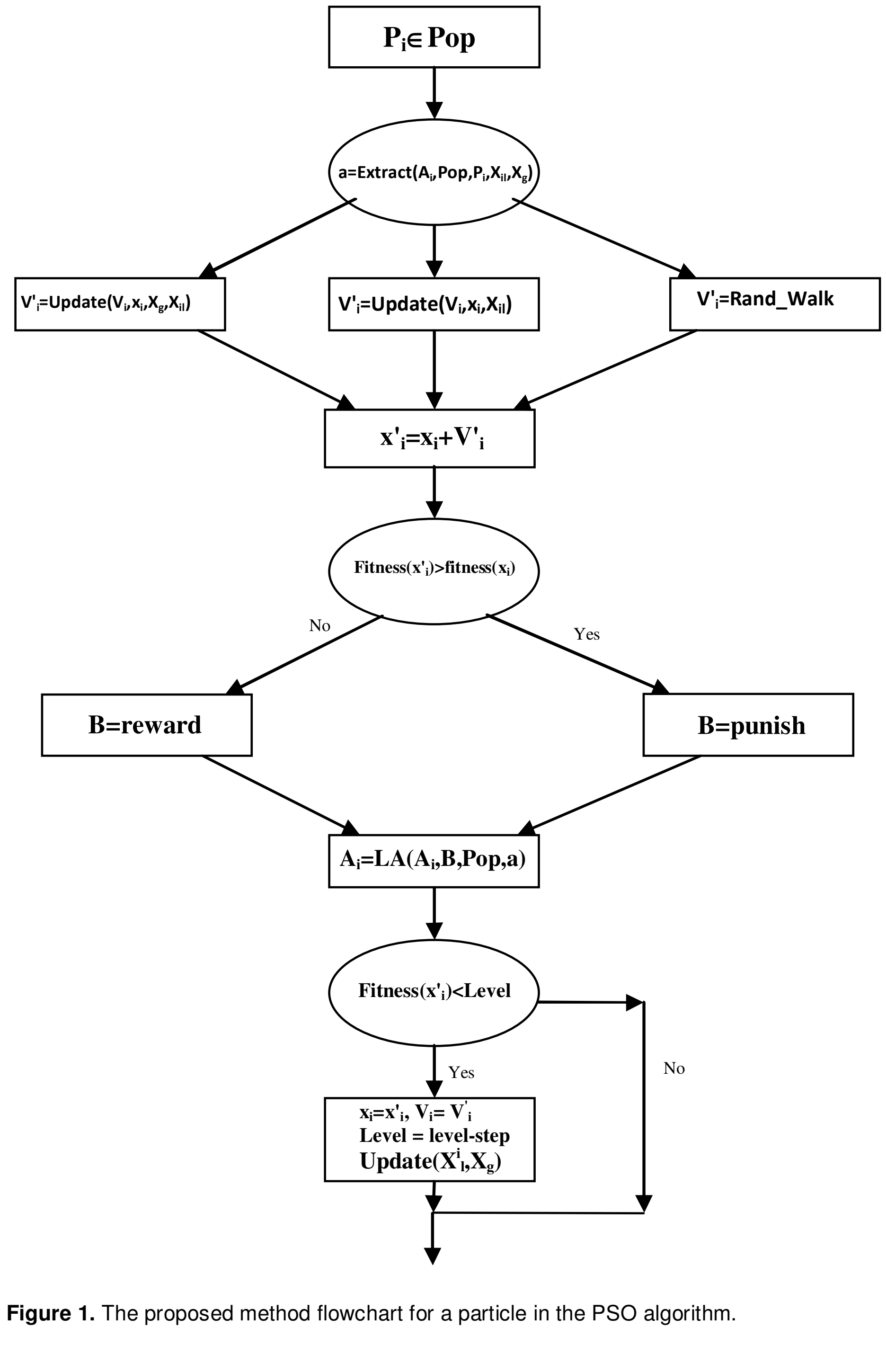

Figure 1. The proposed method flowchart for a particle in the PSO algorithm.

the fitness of its new position is better than an acceptance level.

This flowchart is accomplished per particle. One generation is completed after repeating the flowchart for each particle. The whole process is repeated for a predefined generation number.

In the proposed method, the states of learning automaton A_j is a triple denoted by (*var, dis^j_1* and *dis^j_2*), where *var, dis^j_1* and *dis^j_2* are linguistic variables belonging to {0,1,2}. For calculating *var* first maximum distance between two arbitrary selected X^i_l ($i \in 1..N$) is defined as *max_disqp*.

$$\max\_disqp = \max(\|X^q_l - X^p_l\|) \quad q,p \in 1..N \tag{1}$$

Then, the normalized variance of X^i_l ($i \in 1..N$) denoted by *nvar* is defined as follows:

$$n\,\mathrm{var} = Var(X_l^q)/\max\_disqp \quad q \in 1..N \qquad (2)$$

Now *var* is calculated according the following equations.

$$\mathrm{var} = \begin{cases} 0 & n\,\mathrm{var} < v_1 \\ 1 & v_1 \le n\,\mathrm{var} < v_2 \\ 2 & v_2 \le n\,\mathrm{var} \end{cases} \qquad (3)$$

where v_1 and v_2 are two user-specified thresholds. For calculating dis_1^j, *max_disp* first is defined as following equation.

$$\max\_disp = \max(\left\| X_g - X_l^p \right\|) \quad p \in 1..N \qquad (4)$$

Then the normalized distance between X_l^j and X_g denoted by $ndis_1^j$ is defined as follows:

$$ndis_1^j = (X_g - X_l^j)/\max\_disp \qquad (5)$$

Now dis_1^j is calculated according the following equations.

$$dis_1^j = \begin{cases} 0 & ndis_1^j < d_{1,1} \\ 1 & d_{1,1} \le ndis_1^j < d_{1,2} \\ 2 & d_{1,2} \le ndis_1^j \end{cases} \qquad (6)$$

where d_{11} and d_{12} are two user-specified thresholds. And finally, for calculating dis_2^j, *max_disq* first is defined as follow:

$$\max\_disq = \max(\left\| X_p - X_l^p \right\|) \quad p \in 1..N \qquad (7)$$

Then the normalized distance between X_l^j and x_j denoted by $ndis_2^j$ is defined as follows:

$$ndis_2^j = (X_p - X_l^p)/\max\_disq \qquad (8)$$

Now dis_2^j is calculated according the following equation.

$$dis_2^j = \begin{cases} 0 & ndis_2^j < d_{2,1} \\ 1 & d_{2,1} \le ndis_2^j < d_{2,2} \\ 2 & d_{2,2} \le ndis_2^j \end{cases} \qquad (9)$$

where d_{21} and d_{22} are two user-specified thresholds. Pseudo code of the proposed algorithm is presented in the Algorithm 1. In this code, r_1 and r_2 are both 0.5. Also all w_1, w_2 and w_3 are 0.33.

```
counter=0
max_fitness=maximum of possible fitness in an arbitrary problem setting
level= max_fitness
Repeat
  counter=counter+1
  step = exp( -1*counter)*max_fitness
  each particle i
  Update particle position $x_i$ According to one of the three below equations
       For each particle i
            compute situation according to equation 3, 6 and 9
         a=make_decide(Automata_i, situation)
         if(a=0)
              $v_i(t+1) = w_1 v_i(t) + w_2(x_i^l(t) - x_i(t)) + w_3(x^g(t) - x_i(t))$
         elseif(a=1)
              $v_i(t+1) = r_1 v_i(t) + r_2(x_i^l(t) - x_i(t))$
         elseif(a=2)
              $v_i(t+1) = random$
         End if
       End for
  $x_i' = x_i + v_i$
  if(f(x_i')> f(x_i))
     punish(Automata_i,situation,a)
  else
     reward(Automata_i,situation,a))
  End if
  if(f(x_i')<level)
     $x_i = x_i'$
  End if
  if(f(xi')<f(xg))
     $x_g = x_i'$
  End if
  if(f(x_i')<f(x^l_i))
     $x^l_i = x_i$
  End if
  level=level+step
Until termination criterion reached
```

Algorithm 1. Pseudo code of the proposed algorithm.

Table 1. Setting of parameters in moving peaks benchmark.

Parameter	Value
Number of peaks *m*	10
F	Every 5000 evaluations
Height severity	7.0
Width severity	1.0
Peak shape	cone
Shift length *s*	{0.0}
Number of dimensions *D*	5
A	[0, 100]
H	[30.0 , 70.0]
W	[1, 12]
I	50.0

RESULTS AND DISCUSSION

We have implemented all algorithms by Matlab 2008a and have run all algorithms on a PC with a core 2 duo processor 2 GHz and a 3 G Bytes RAM memory.

Branke (1999) introduced a dynamic benchmark problem, called moving peaks benchmark (MPB) problem. In this problem, there are some peaks in a multi-dimensional space, where the height, width and position of each peak change during the environment change. This function is widely used as a benchmark for dynamic environments in literature (Li et al., 2008; Moser, 2007).

The default parameter setting of MPB used in the experiments is presented in Table 1. In MPB, shift length (s) is the radius of peak movement after an environment change, *m* is the number of peaks, *f* is the frequency of environment change as the number of fitness

Table 2. Offline error ± standard error for f = 500.

f = 1000	Proposed algorithm	Multi swarm PSO (Kamosi et al., 2010)	Cellular PSO	FMSO	mQSO 10
1	6.12 ± 0.22	2.90 ± 0.18*	6.77 ± 0.38	14.42 ± 0.4	18.6 ± 1.6
5	5.66 ± 0.20	3.35 ± 0.18*	5.30 ± 0.32	10.59 ± 0.2	6.56 ± 0.38
10	5.88 ± 0.16	3.94 ± 0.08*	5.15 ± 0.13	10.40 ± 0.1	5.71 ± 0.22
20	5.36 ± 0.16	4.33 ± 0.12*	5.23 ± 0.18	10.33 ± 0.1	5.85 ± 0.15
30	5.37 ± 0.16	4.41 ± 0.11*	5.33 ± 0.16	10.06 ± 0.1	5.81 ± 0.15
40	4.45 ± 0.11*	4.52 ± 0.09	5.61 ± 0.16	9.85 ± 0.11	5.70 ± 0.14
50	4.49 ± 0.15*	4.57 ± 0.08	5.55 ± 0.14	9.54 ± 0.11	5.87 ± 0.13
100	3.79 ± 0.09*	4.77 ± 0.08	5.57 ± 0.12	8.77 ± 0.09	5.83 ± 0.13
200	3.93 ± 0.10*	4.76 ± 0.07	5.50 ± 0.12	8.06 ± 0.07	5.54 ± 0.11

* Result of the best performing algorithm(s) with 95% confidence.

evaluations. *H* and *W* denote range of height and width of peaks which will change after a change in environment by height and width severity, respectively. *I* is the initial heights for all peaks. Parameter A denotes minimum and maximum value on all dimensions. For evaluating the efficiency of the algorithms, we use the offline error measure, the average deviation of the best individual from the optimum in all iterations.

In the proposed method, the acceleration coefficients c_1 and c_2 are set to 2.8 and 1.3 and the inertial weight w is set to mean of c_1 and c_2 (2.05). The number of particles in the swarm is set to 20 particles. d_{11}, d_{21}, d_{12}, d_{22}, v_1 and v_2 are set to 0.4, 0.4, 0.6, 0.6, 0.4 and 0.6, respectively. The proposed algorithm is compared with multi-swarm PSO (Kamosi et al., 2010), mQSO (Blackwell and Branke, 2006), FMSO (Li and Yang, 2008) and cellular PSO (Hashemi and Meybodi, 2009). In multi-swarm PSO, the acceleration coefficients c_1 and c_2 are set to 1.496180 and the inertial weight w is set to 0.729844. The number of particles in the parent swarm and the child swarms (π) are set to 5 and 10 particles, respectively. The radius of the child swarms (r), the minimum allowed distance between two child swarm (rexcl) and the radius of quantum particles (rs) are set to 30.0, 30.0 and 0.5, respectively. For mQSO, we adapted a configuration 10 (5 + 5q) which creates 10 swarms with 5 neutral (standard) particles and 5 quantum particles with rcloud = 0.5 and rexcl = rconv = 31.5, as suggested (Blackwell and Branke, 2006). For FMSO, there are at most 10 child swarms, each has a radius of 25.0. The size of the parent and the child swarms are set to 100 and 10 particles, respectively (Li and Yang, 2008). For cellular PSO, a 5-Dimensional cellular automaton with 105 cells and Moore neighborhood with radius of two cells embedded into the search space. The maximum velocity of particles is set to the neighborhood radius of the cellular automaton and the radius for the random local search (r) is set to 0.5 for all experiments. The cell capacity θ is set to 10 particles for every cell (Hashemi and Meybodi, 2009).

As depicted in Tables 2 to 5, the proposed algorithm outperforms other tested PSO algorithms when the number of peaks increases.

For all algorithms, we reported the average offline error and 95% confidence interval for 100 runs. Offline error of the proposed algorithm, mQSO 10 (5 + 5q) (Blackwell and Branke, 2006), FMSO (Li and Yang, 2008), cellular PSO (Hashemi and Meybodi, 2009) and multi-swarm PSO (Kamosi et al., 2010) for different dynamic environment is presented in Tables 2 and 3.

Also to show the effect of the dimension of the problem over the efficacy of the algorithm, the results of offline error of the proposed algorithm is given in Table 6.

The convergence of the proposed method compares with the best method proposed so far to deal with dynamic moving peak problem is shown in Figure 2.

Conclusion

In this paper, new PSO algorithm is proposed for dynamic environment. In the proposed PSO, there is one learning automaton per particle in which it learns for each particle, how to act during the evolution. To prevent redundant search in the same area, the LA belonging to particle P_i, which is denoted by L_i, learns the relationship between the variance of the solutions, normalized distance between the position x_i and position of its local optima and normalized distance between the position of its local optima and global optima, and the behavior of the particles. Indeed the proposed PSO is a kind of indirect niching method. In addition, the deluge water level is employed during the evolution.

Results of the experiments show that for many tested dynamic environments, the proposed algorithm out-performs all competent tested PSO algorithms.

Table 3. Offline error ± standard error for f = 1000.

f = 5000	Proposed algorithm	Multi swarm PSO (Kamosi et al., 2010)	Cellular PSO	FMSO	mQSO 10
1	2.54 ± 0.19	0.56 ± 0.04*	2.55 ± 0.12	3.44 ± 0.11	3.82 ± 0.35
5	1.49 ± 0.11	1.06 ± 0.06*	1.68 ± 0.11	2.94 ± 0.07	1.90 ± 0.08
10	1.44 ± 0.10*	1.51 ± 0.04	1.78 ± 0.05	3.11 ± 0.06	1.91 ± 0.08
20	1.85 ± 0.11*	1.89 ± 0.04	2.61 ± 0.07	3.36 ± 0.06	2.56 ± 0.10
30	2.00 ± 0.09*	2.03 ± 0.06	2.93 ± 0.08	3.28 ± 0.05	2.68 ± 0.10
40	2.02 ± 0.08*	2.04 ± 0.06	3.14 ± 0.08	3.26 ± 0.04	2.65 ± 0.08
50	2.03 ± 0.08*	2.08 ± 0.02	3.26 ± 0.08	3.22 ± 0.05	2.63 ± 0.08
100	2.23 ± 0.04	2.14 ± 0.02*	3.41 ± 0.07	3.06 ± 0.04	2.52 ± 0.06
1	2.54 ± 0.19	0.56 ± 0.04*	2.55 ± 0.12	3.44 ± 0.11	3.82 ± 0.35

* Result of the best performing algorithm(s) with 95% confidence.

Table 4. Offline error ± standard error for f = 5000.

f = 10000	Proposed algorithm	Multi swarm PSO (Kamosi et al., 2010)	Cellular PSO	FMSO	mQSO 10
1	1.52 ± 0.17	0.27 ± 0.02*	1.53 ± 0.12	1.90 ± 0.06	1.90 ± 0.18
5	0.88 ± 0.11	0.70 ± 0.10*	0.92 ± 0.10	1.75 ± 0.06	1.03 ± 0.06
10	0.91 ± 0.06*	0.97 ± 0.04	1.19 ± 0.07	1.91 ± 0.04	1.10 ± 0.07
20	1.32 ± 0.07*	1.34 ± 0.08	2.20 ± 0.10	2.16 ± 0.04	1.84 ± 0.08
30	1.35 ± 0.05*	1.43 ± 0.05	2.60 ± 0.13	2.18 ± 0.04	2.00 ± 0.09
40	1.27 ± 0.04*	1.47 ± 0.06	2.73 ± 0.11	2.21 ± 0.03	1.99 ± 0.07
50	1.30 ± 0.03*	1.47 ± 0.04	2.84 ± 0.12	2.60 ± 0.08	1.99 ± 0.07
100	1.32 ± 0.03*	1.50 ± 0.03	2.93 ± 0.09	2.20 ± 0.03	1.85 ± 0.05
200	1.51 ± 0.02	1.48 ± 0.02*	2.88 ± 0.07	2.00 ± 0.02	1.71 ± 0.04

* Result of the best performing algorithm(s) with 95% confidence.

Table 5. Offline error ± standard error for f = 10000.

Peaks versus dimension	1	2	3	4	5
1	0.38 ± 0.01	0.62 ± 0.04	1.00 ± 0.03	1.19 ± 0.02	1.52 ± 0.17
5	0.13 ± 0.02	0.56 ± 0.03	0.93 ± 0.03	0.76 ± 0.02	0.88 ± 0.11
10	0.26 ± 0.05	0.33 ± 0.01	0.84 ± 0.05	1.01 ± 0.01	0.91 ± 0.06
20	0.13 ± 0.02	0.55 ± 0.04	0.88 ± 0.04	1.20 ± 0.03	1.32 ± 0.07
30	0.16 ± 0.03	0.54 ± 0.18	0.60 ± 0.03	1.23 ± 0.18	1.35 ± 0.05
40	0.15 ± 0.02	0.52 ± 0.11	0.81 ± 0.05	1.15 ± 0.04	1.27 ± 0.04
50	0.16 ± 0.05	0.69 ± 0.35	0.76 ± 0.06	0.93 ± 0.04	1.30 ± 0.03
100	0.08 ± 0.02	0.57 ± 0.14	0.73 ± 0.04	1.01 ± 0.03	1.32 ± 0.03
200	0.04 ± 0.01	0.66 ± 0.11	0.81 ± 0.15	1.14 ± 0.05	1.51 ± 0.02

Table 6. Offline error ± standard error of the proposed algorithm for f = 10000 through different dimension.

Peaks versus dimension	1	2	3	4	5
1	0.38 ± 0.01	0.62 ± 0.04	1.00 ± 0.03	1.19 ± 0.02	1.52 ± 0.17
5	0.13 ± 0.02	0.56 ± 0.03	0.93 ± 0.03	0.76 ± 0.02	0.88 ± 0.11
10	0.26 ± 0.05	0.33 ± 0.01	0.84 ± 0.05	1.01 ± 0.01	0.91 ± 0.06
20	0.13 ± 0.02	0.55 ± 0.04	0.88 ± 0.04	1.20 ± 0.03	1.32 ± 0.07
30	0.16 ± 0.03	0.54 ± 0.18	0.60 ± 0.03	1.23 ± 0.18	1.35 ± 0.05
40	0.15 ± 0.02	0.52 ± 0.11	0.81 ± 0.05	1.15 ± 0.04	1.27 ± 0.04
50	0.16 ± 0.05	0.69 ± 0.35	0.76 ± 0.06	0.93 ± 0.04	1.30 ± 0.03
100	0.08 ± 0.02	0.57 ± 0.14	0.73 ± 0.04	1.01 ± 0.03	1.32 ± 0.03
200	0.04 ± 0.01	0.66 ± 0.11	0.81 ± 0.15	1.14 ± 0.05	1.51 ± 0.02

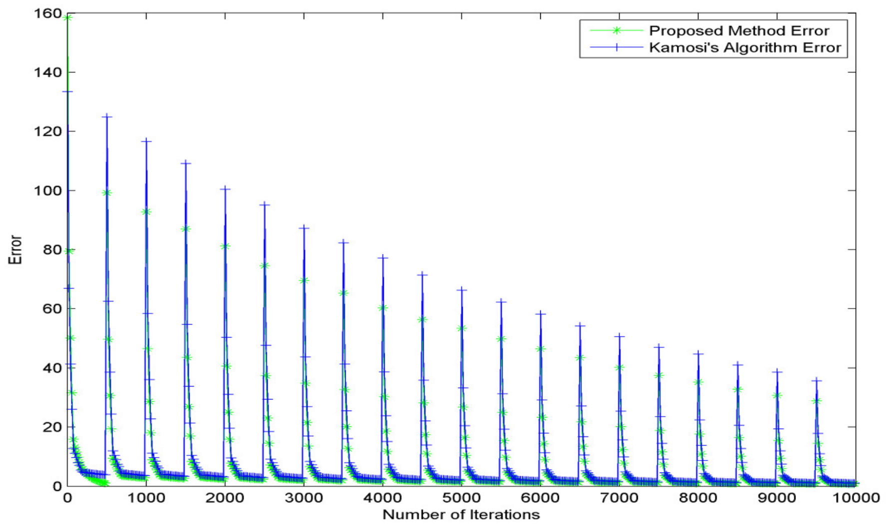

Figure 2. Convergence of the proposed method when compared with the best proposed method with frequency 500 and peak number equal.

For further study, researchers are advised to turn to employ more actions which can be learned by learning automata during exploration phase of particle swarm optimization algorithm.

REFERENCES

Blackwell T, Branke J (2004). Multi-Swarm Optimization in Dynamic Environments. Appl. Evolut. Comput., pp. 489–500.

Blackwell T, Branke J (2006). Multiswarms, Exclusion, and Anti-Convergence in Dynamic Environments. IEEE Trans. Evolut. Computat., 10: 459–472.

Blackwell T, Branke J, Li X (2008). Particle Swarms for Dynamic Optimization Problems. Swarm Intell., pp.193–217.

Branke J (1999). Memory Enhanced Evolutionary Algorithms for Changing Optimization Problems. Congress on Evolutionary Computation, 3: 1875–1882.

Dueck G (1993). New Optimization Heuristics. The Great Deluge Algorithm and the Record-to-Record Travel. J. Comput. Phys., 104: 86-92.

Hashemi AB, Meybodi MR (2009). Cellular PSO: A PSO for Dynamic Environments. Advances in Computation and Intelligence, pp.422–433.

Hu X, Eberhart RC (2002). Adaptive particle swarm optimization: detection and response to dynamic systems. IEEE Congress on Evolutionary Computation, 2: 1666–1670.

Janson S, Middendorf M (2004). A Hierarchical Particle Swarm Optimizer for Dynamic Optimization Problems. Appl. Evolut. Comput., pp.513–524.

Janson S, Middendorf M (2006). A hierarchical particle swarm optimizer for noisy and dynamic environments. Gen. Program. Evolvable. Mach., 7: 329–354.

Kamosi M, Hashemi AB, Meybodi MR (2010). A New Particle Swarm Optimization Algorithm for Dynamic Environments. SEMCCO, pp.129-138.

Kennedy J, Eberhart RC (1995). Particle Swarm Optimization. IEEE International Conference on Neural Networks, 5: 1942-1948.

Kennedy J, Mendes R (2002). Population structure and particle swarm performance. Evolutionary Computation Congress, pp.1671–1676.

Li C, Yang S (2008). Fast Multi-Swarm Optimization for Dynamic Optimization Problems. Fourth IEEE International Conference on Natural Computing, 7: 624–628.

Li C, Yang S (2009). A clustering particle swarm optimizer for dynamic optimization. IEEE Congress on Evolutionary Computation, pp. 439–446.

Li X, Dam KH (2003). Comparing particle swarms for tracking extrema in dynamic environments. IEEE Congress on Evolutionary Computation, pp.1772–1779.

Liu L, Wang D, Yang S (2008). Compound Particle Swarm Optimization in Dynamic Environments. Appl. Evolut. Comput., pp. 616–625.

Liu L, Yang S, Wang D (2010). Particle Swarm Optimization with Composite Particles in Dynamic Environments. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, pp. 1–15.

Lung RI, Dumitrescu D (2007). A collaborative model for tracking optima in dynamic environments. IEEE Congress on Evolutionary Computation, pp.564–567.

Moser I (2007). All Currently Known Publications on Approaches Which Solve the Moving Peaks Problem. Swinburne University of Technology, Melbourne.

Thomsen R (2004). Multimodal optimization using crowding-based differential evolution. IEEE Congress on Evolutionary Computation, pp. 1382–1389.