Cellular PSO: A PSO for Dynamic Environments

Ali B. Hashemi and M.R. Meybodi

Computer Engineering and Information Technology Department,
Amirkabir University of Technology, Tehran, Iran
{a_hashemi,mmeybodi}@aut.ac.ir

Abstract. Many optimization problems in real world are dynamic in the sense that the global optimum value and the shape of fitness function may change with time. The task for the optimization algorithm in these environments is to find global optima quickly after the change in environment is detected. In this paper, we propose a new hybrid model of particle swarm optimization and cellular automata which addresses this issue. The main idea behind our approach is to utilized local interactions in cellular automata and split the population of particles into different groups across cells of cellular automata. Each group tries to find an optimum locally which results in finding the global optima. Experimental results show that cellular PSO outperforms mQSO, a well known PSO model in literature, both in accuracy and complexity in a dynamic environment where peaks change in width and height quickly or there are many peaks.

Keywords: Dynamic environments, Particle swarm optimization, Cellular Automata.

1 Introduction

In the real world, many applications are non-stationary problems which need to not only finding the global optimal solution but also keeping trace of its change. Particle swarm optimization algorithms (PSO) have gained popularity in recent years. PSO is a population-based method, a variant of evolutionary algorithms with moving towards the target rather than evolution, through the search space. The basic idea behind this approach is iterative ameliorating of global participant's perception of the target by exchanging local information among them. However due to the static context of PSO usage, some issues arise when using them in dynamic environments. These challenges lie in two aspects: outdated memory due to changing environment and diversity loss due to convergence. Of these two the diversity loss is by far more serious. It has been demonstrated that the time taken for a partially converged swarm to re-diversify, find the shifted peak, and then re-converge is quite deleterious to the performance of PSO [1].

In this paper we address diversity loss problem in adapting PSO to dynamic environments and propose a variant of multi swarm method to solve it. To this aim, embedded cellular automata are utilized to maintain diversity of particles and to scatter them over the search space. Particles in each cell of cellular automata search for a local optimum and broadcast the best solutions to neighborhood. Comparing the best

Z. Cai et al. (Eds.): ISICA 2009, LNCS 5821, pp. 422–433, 2009.

results found in neighboring cells and the best position in their cell, particles in a cell sets their direction towards the best solution found in their neighborhood and their best personal experience. However in moving towards dynamic optimum, preserving particle density in each cell below a threshold is of great concern. Hence upon arriving in a cell, a portion of particles may be reinitialized to keep particle density below this threshold. This mechanism makes the swarms spread out over the highest multiple peaks across the environment, meanwhile helps convergence onto a local optimum in a short time. Extensive experiments show that the proposed cellular PSO results less offline error than mQSO[2] a well known PSO model in literature, in environments which have many peaks or width and height of the peaks change very fast.

The rest of this paper is organized as follows: Section 2 provides a brief introduction on PSO and an overview of the previous works on adapting PSO into dynamic environments. Section 3 introduces cellular automata as the foundations of our approach following by detailed specification of the proposed algorithm in section 4. Section 5 gives out the experimental results of the proposed model along with its comparison to best results gained in previous works. Finally in section 6 we conclude our paper.

2 Related Work

2.1 Particle Swarm Optimization

The particle swarm optimization (PSO) algorithm is introduced by Kennedy and Eberhart [3] based on the social behavior metaphor. The fundament for the development of PSO is hypothesis that a potential solution to an optimization problem is treated as a bird without quality and volume, which is called a particle, flying through a D-dimensional space, adjusting its position in search space according to its own experience and its neighbors.

In PSO, the ith particle is represented as $p_i=(p_{i1}, p_{i2,}\ldots, p_{iD,})$ in the D-dimensional space. The velocity for particle i is represented as $V_i=(v_{i1}, v_{i2,}\ldots, v_{iD,})$, which is usually clamped to a maximum velocity V_{max}, specified by the user. In each time step t, the particles calculate their new velocity then update their position according to eq. (1) and eq. (2) respectively.

$$v_i(t+1) = v_i(t) + c_1 r_1 \left(p_i^{best} - p_i(t) \right) + c_2 r_2 \left(lbest - p_i(t) \right) \tag{1}$$

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

Where c_1 and c_2 are positive acceleration constants used to scale the contribution of cognitive and social components respectively. r_1 and r_2 are uniform random variables in range [0,1]. p_i^{best} is the best personal position of particle i which has been visited during the lifetime of the particle. *lbest* is the local best position that is the best position of all neighboring particles of particle i.

2.2 PSO in Dynamic Environment

The application of PSO to dynamic problems has been explored in various literatures. In this section we provide a brief overview of the most relevant works which shapes foundations of our approach through addressing diversity loss in dynamic optimization problems.

The main idea behind multi-swarms approaches is to divide swarms into sub groups and place them on the best peaks found so far, while letting some particles explore for new peaks. Following this idea, Blackwell and Branke [2, 4] has suggested dividing the whole population into several sub swarms incorporating three concepts, namely quantum particles, exclusion and anti-convergence. Quantum particles appear at random positions around the sub-swarm best ensuring a close chase of moving peaks. Exclusion operator regarding local interaction between swarms, upon falling their distance beneath a minimum threshold, reinitializes one with the aim of keeping diversity at a desired level. Moreover taking into account the possibly of existing more peaks than swarms, it is necessary to keep constantly patrolling for new and better peaks. To this end anti-convergence operator has been proposed.

In [5], Blackwell et. al. introduced two variants of the canonical method in [6] addressing self adaptation problem. One starting with a single swarm and adding additional swarms as needed regarding a predefined number of total particles, the other fixing overall number of particles and dynamically distributes particles to swarms, usually starting with many swarms, slowly converging to the required number of swarms to cover all peaks. In both methods, by reducing the number of permanent quantum particles and converting all neutral particles to quantum particles for one iteration only after a change, convergence speed towards local peaks by maximizing its use of neutral particles has been increased.

Lung and Dumitrescu used two collaborating populations of equal size to avoid premature convergence and to efficiently trace the moving optimum [7]. One is responsible for preserving the diversity of the search by using crowding differential evolutionary algorithm while the other keeps track of global optimum with a PSO algorithm. The collaboration mechanism is applied whenever a change is detected in the search space, or if the best individual in second population is too close to the best solution. The search of the second population is restarted by re-initializing it with the positions of individuals in the first population.

In [8] Li and Yang proposed a multi-swarm method which maintains the diversity through the run. To meet this end two type of swarms are used: a parent swarm which maintains the diversity and detects 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 many as possible, and guarantees to converge to a local optimum in a short time.

Du and Li [9] suggest dividing particles into two parts, of which the first uses a standard PSO enhanced a Gaussian local search and the second uses differential mutation acting as a patrol team around first to extend the search area of algorithm, and catch up with the moving optimum. The introduced strategies in these two parts,

respectively enhances the convergence ability of the algorithm and avoids being trapped into the local optimum.

3 Cellular Automata

Cellular automata are mathematical models for systems consisting of large number of simple identical components with local interactions in which space and time are discrete. It is called cellular because it is made up of cells like points in a lattice or like squares of checker boards, and it is called automata because it follows a simple rule[10]. Informally, a d-dimensional CA consists of a d-dimensional lattice of identical cells. The simple components act together to produce complicated patterns of behavior. Each cell can assume a state from a finite set of states. The cells update their states synchronously on discrete steps according to a local rule. The new state of each cell depends on the previous states of a set of cells, including the cell itself, and constitutes its neighborhood (Figure 1). The state of all cells in the lattice is described by a configuration. A configuration can be described as the state of the whole lattice. The rule and the initial configuration of the CA specify the evolution of CA that tells how each configuration is changed in one step. Cellular automata perform complex computations with a high degree of efficiency and robustness. They are especially suitable for modeling natural systems that can be described as massive collections of simple objects interacting locally with each other [11, 12].

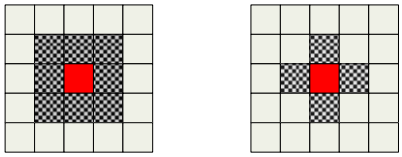

Fig. 1. Neighborhood in a 2-D cellular automata: a. Moore neighborhood b. von Neumann Neighborhood

4 Proposed Model

In previous multi-swarm methods [2, 5, 8] in order to prevent sitting of two or more swarm on a peak, every pair of swarms have to calculate their distance so that if they were too close, the worse swarm reinitialize. In the proposed model we omit the burden of exhaustive distance calculations by cellular automata into the search space as shown in Fig 2. Hence, we call our model “Cellular PSO”. Although the term "Cellular PSO" was first used in [13], it was used as a synonym of a local best PSO with a Von Neumann neighborhood.

In our model, D-dimensional cellular automata with C^D equal cells are used in a D-dimensional environment. Therefore, a particle in search space can be assigned to one cell in cellular automata. This concept is used to preserve diversity in the search space. We define *particle density* of a cell c at time t, $\rho_c(t)$ to be the number of particles which are positioned in boundaries of cell c at the specified time t. By keeping

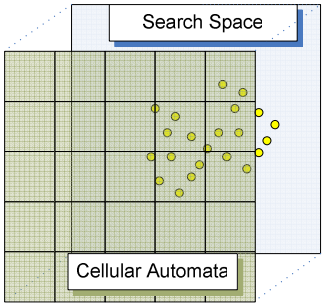

Fig. 2. Embedding cellular automata in a 2-D Search Space

the particle density of all cells bellow a specified threshold θ, we prevent all particles from converging to one cell. Therefore, only a portion of all particles can investigate a sub-space while there are particles available to search other parts of the space. Moreover, when particle density of cell i rise up beyond the acceptable threshold θ, some of the particles in cell i are randomly selected to be reinitialized in the cellular automata as follows.

First, status of a selected particle is changed to *inactive*. Then a randomly selected cell is assigned as destination of the particle. Distance of current cell and the destination cell is calculated and is set as the hop count for the selected particle. Afterwards at the next iteration, neighbors of cell i read this particle's information as part of state of cell i and add this inactive particle to their state while decreasing the particle's hop counter. This will continue until the destination cell receives this particle as part of one of its neighbors' state. At this time, particle will be activated and will begin searching in the new neighborhood.

Moreover, the proposed model only uses local interaction between cells of cellular automata (*lbest* PSO model) to maintain better diversity among the cells. $lbest_i$ for all particles in cell i is defined the best position found in neighborhood of cell i, including cell i.

In the proposed model, state of cell i is consists of following information.

1. Best position found in cell i. ($cbest_i$)
2. Best position found in cell i and neighborhood of cell i since last environment change. ($lbest_i$)
3. Information of all particles currently exists in cell i, including particle position, velocity, state, destination, and hop count. Where particle state can be active or inactive. When a particle is inactive it does no function evaluation. Particle's destination and hop count are only meaningful for an inactive particle which is headed to another cell as mentioned above.

A change is detected by monitoring of *lbest* as in [14, 15]. After detecting an environment change, memory of all cell including $cbest_i$ and $lbest_i$ and all particles' history is reset.

At the beginning, N particles are initialized into cellular automata randomly. Then at iteration t cell i updates its state according to its current state and it neighboring cells' with the following rule.

1. $lbest_i$ is updated by reading state of its neighbors, e.g. cell j, according to *update lbest* procedure in Fig. 3.

```
Procedure updatelbest
begin
    for All cell j, cell_j is a neighbour of cell i do
       if  f(lbest_i) < f(cbest_j) then
          lbest_i ← cbest_j
       end if
       for all p_k ∈ P_j do
       if (p_k is inactive) then
        p_k.hop ← p_k.hop-1;
       endif
       if (p_k.hop > 0) then
        add p_k to P_i
       else if ( p_k.hop == 0 and p_k.dst=i) then
        activate p_k
        add p_k to P_i
       endif
      endfor
    endfor
end
```

Fig. 3. Update $lbest_i$

```
Procedure updateParticles
begin
  for all active particles p_j then
    update particle velocity according to eq. 3
    update particle position according to eq. 2
    if  cellof(p_j(t+1) ≠ i then
        deactivate p_j
        p_j.dst ← cellof(p_j(t+1)
        p_j.hop ← distance(i, p_j.dst)
        p_j.initialize ← false
    end if
  end for
end
```

Fig. 4. Update particle procedure for cell i

2. All particles p_k in cell i updates their velocity according to eq. (3)
$$v_k(t+1) = w\, v_k(t) + c_1 r_1 \left(pbest_k - p_k(t) \right) + c_2 r_2 \left(lbest_i - p_k(t) \right) \quad (3)$$
3. Particle's next position is calculated by eq. (2). If p_k is leaving cell i to one of its neighbors e.g. cell j where $i \neq j$, then cell c_i will set the state of particle p_k to inactive. Then at time $t+1$ when cell j receives state of cell i, it will activate particle p_k and will add it to its state. (Fig. 4)
4. After updating position of all particles, cell i checks its particle density. If the particle density in cell i, $\rho_i(t)$, is above the specified threshold θ, some particles should scatter among the cellular automata. Therefore, θ-$\rho_i(t)$ particles in cell i are randomly selected to be reinitialized in cellular automata as mentioned above. This will decrease particle density below the threshold θ while increasing diversity of particles in search space. (Fig. 5)

```
Procedure ParticleDensityControl
begin
  if ρ_i > θ then
    R←select θ-ρ_i(t)  particles randomly;
    for all particle p_i∈R do
      p_i(t+1).statue ← inactive
      p_i(t+1).destinationCell ← a random cell of CA
      p_i(t+1).hop ←distance of p_i(t+1).destinationCell,c
      p_i(t+1).init ← true
    end for
  end if
end
```

Fig. 5. Density control procedure for cell i

5 Experimental Study

5.1 Dynamic Test Function

Branke [16] introduced a dynamic benchmark problem, called moving peaks benchmark 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 [8, 17].

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

number of fitness evaluations. *H* and *W* denote range of height and width of peaks which will change after a change in environment by *height severity* 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.

Table 1. Parameters of 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

5.2 Experimental Settings

In this study we have tested our proposed model on MPB and compared the results with mQSO [2]. In [2] Blackwell and Brank have shown that mQSO results less offline error than standard PSO with re-initialization [15] and Hierarchical Swarms [18]. Therefore we omit the results of those algorithms and only compare cellular PSO with mQSO 10(5+5^q) and mQSO10 (10+0^q) which have the parameters reported in [2] (r_{excel}=31.5, r_{conv}=0.0).

For Cellular PSO (CPSO), based on the previous experimental work[19], the acceleration constants C_1 and C_2 were both set to 1.496180, and the inertia weight *w* was 0.729844. We set maximum velocity for a particle equal to the neighborhood distance on the cellular automata. Hence, a particle will not move further than neighboring cells. The size of swarm has been set to 100 particles for all the experiments respectively. In cellular PSO, cellular automata with 10^D cells in a D-Dimensional space have been used. In the cellular automata neighborhood has been defined as Moore neighborhood with a neighborhood of 1 cell for D=2,3, and 2 cells for D=4,5. Moreover, the maximum allowed particle density (θ), was set equally in all cells to 10 particles per cell.

5.3 Experimental Results

Average offline error and 95% confidence interval in 100 runs for different frequency of environment change and different number of peaks for both CPSO, mQSO 10(5+5^q), and mQSO 10(10+0^q) algorithms are presented in Table 2 -Table 7. For each environment, offline error of the all algorithms have been compares

statistically (with error 0.05) and the result(s) of the algorithm(s) performing significantly better than others is printed in bold.

In the fastest changing environment, where the peaks change every 500 iterations (Table 2), cellular PSO performs significantly better than mQSO for different number of peaks. This superiority also exists in the environment here the peaks change every 1000 iterations (Table 3), for different number of peaks but for the 10 peaks environment in which there is no significant difference between offline error for CPSO and mQSO 10(5+5^q).

By decreasing the dynamicity of the environment (increasing f), a trend in difference of CPSO and mQSO can be seen in which CPSO breaks out to perform the same as or even worse than mQSO 10(5+5^q) where the number of peaks is around 10 (Table **3** through Table 6). This trend, which can be due to specific tuning of mQSO 10(5+5^q) for 10 peaks environments [2], is describe as follows.

The trend begins in environment with f=1000 (Table **3**) where CPSO performs significantly better than mQSO 10(5+5^q) for all number of peaks but the 10 peaks environment where there is just no significant difference between offline error of CPSO and mQSO 10(5+5^q). This pattern expands in the environment with f=2500 (Table 4) where not only for the environments with 30 or less peaks CPSO and mQSO 10(5+5^q) do not have any significant difference in offline error, but also for the 10 peaks environment mQSO 10(5+5^q) results significantly less offline error than CPSO. Almost the same pattern repeats for the environment with f=5000 (Table 5), but with not only mQSO 10(5+5^q) performs better than CPSO in the 10 peaks environment but also it expands this superiority for the single peak environment. The increasing trend can be better seen in the least changing tested environment where f=10000 (Table 6) in which CPSO can only compete with mQSO 10(5+5^q), in terms of offline error, for environments which have 30 peaks or more.

Furthermore, for environments that positions of the peaks also change with s=1 (Table 7 and Table 8), CPSO still outperforms mQSO in low dimensional highly dynamic environments (D=2 and f=500) or when the environment has few peaks. But when either the dimension or number of peaks increases, mQSO 10(5+5^q) performs significantly better than cellular PSO.

Although introducing new parameters in cellular PSO, e.g. size of cellular automata and particle density threshold, might be considered a drawback for the proposed model, the experimental results that cellular PSO can outperform mQSO[2] in various environments without changing value of its the parameters.

Table 2. Offline error for different number of peaks (f =500 and s=0)

m	CPSO	mQSO 10(10+0^q)	mQSO 10(5+5^q)
1	**9.78±0.79**	35.22±3.10	24.97±2.61
5	**1.91±0.18**	5.83±0.86	3.91±0.56
10	**1.44±0.13**	2.56±0.17	1.89±0.17
20	**2.40±0.06**	3.39±0.05	2.94±0.04
30	**2.90±0.08**	3.84±0.05	3.50±0.05
40	**3.12±0.07**	4.09±0.05	3.81±0.05
50	**3.31±0.08**	4.29±0.05	3.92±0.05
100	**3.57±0.06**	4.59±0.04	4.31±0.05
200	**3.75±0.07**	4.72±0.05	4.45±0.05

Table 3. Offline error for different number of peaks (f=1000 and s=0)

m	CPSO	mQSO 10(10+0^q)	mQSO 10(5+5^q)
1	**7.37±0.53**	14.56±1.49	12.25±1.42
5	**1.56±0.17**	3.18±0.43	2.68±0.23
10	**1.33±0.11**	1.77±0.09	**1.29±0.07**
20	**2.22±0.08**	2.64±0.05	2.46±0.06
30	**2.74±0.08**	3.20±0.05	3.05±0.06
40	**2.89±0.08**	3.42±0.05	3.36±0.06
50	**2.98±0.07**	3.58±0.06	3.45±0.06
100	**3.27±0.08**	3.95±0.06	3.78±0.06
200	**3.43±0.08**	4.03±0.06	3.92±0.07

Table 4. Offline error for different number of peaks (f=2500 and s=0)

m	CPSO	mQSO 10(10+0^q)	mQSO 10(5+5^q)
1	**6.10±0.55**	6.49±0.59	**5.43±0.58**
5	**1.17±0.18**	1.87±0.21	**1.38±0.11**
10	1.08±0.09	1.48±0.10	**0.90±0.06**
20	**2.19±0.11**	2.39±0.07	**2.21±0.08**
30	**2.62±0.11**	2.87±0.09	**2.71±0.08**
40	**2.77±0.12**	3.07±0.08	2.92±0.08
50	**2.94±0.09**	3.28±0.08	3.20±0.10
100	**3.08±0.10**	3.66±0.09	3.56±0.09
200	**3.22±0.11**	3.77±0.10	3.66±0.10

Table 5. Offline error for different number of peaks (f=5000 and s=0)

m	CPSO	mQSO 10(10+0^q)	mQSO 10(5+5^q)
1	5.23±0.47	3.36±0.31	**2.92±0.30**
5	**1.09±0.22**	1.25±0.13	**0.97±0.08**
10	1.14±0.13	1.54±0.13	**0.85±0.08**
20	**2.20±0.12**	**2.23±0.11**	**2.09±0.12**
30	**2.67±0.13**	**2.67±0.10**	**2.62±0.13**
40	**2.70±0.13**	2.98±0.13	2.99±0.13
50	**2.77±0.13**	3.13±0.13	3.02±0.13
100	**2.97±0.14**	3.58±0.13	3.41±0.14
200	**3.14±0.12**	3.52±0.11	3.48±0.13

Table 6. Offline error for different number of peaks (f=10000 and s=0)

m	CPSO	mQSO 10(10+0^q)	mQSO 10(5+5^q)
1	4.18±0.39	**1.59±0.16**	**1.49±0.15**
5	0.85±0.16	0.67±0.10	**0.53±0.06**
10	1.10±0.18	0.90±0.08	**0.57±0.07**
20	2.26±0.18	**1.98±0.09**	**1.86±0.11**
30	**2.57±0.17**	**2.49±0.11**	**2.48±0.13**
40	**2.72±0.18**	**2.88±0.12**	**2.87±0.13**
50	**2.89±0.16**	3.05±0.12	**2.89±0.13**
100	**2.90±0.16**	3.28±0.13	3.27±0.13
200	**3.17±0.17**	3.68±0.13	**3.34±0.13**

Table 7. Offline error for *CPSO* for different number of peaks and dimensions, f=500 and s=1

m \ D	2	3	4	5
1	**4.58±0.16**	**7.43±0.33**	**9.23±0.34**	**11.41±0.68**
5	**3.45±0.10**	**4.40±0.15**	**6.43±0.19**	**6.83±0.18**
10	**3.42±0.06**	**4.29±0.08**	6.34±0.12	6.81±0.17
20	**3.44±0.03**	**5.11±0.05**	6.50±0.08	7.73±0.07
30	**3.42±0.03**	**5.46±0.07**	6.78±0.06	8.39±0.11
40	**3.35±0.03**	5.65±0.07	6.89±0.05	8.64±0.11
50	**3.29±0.03**	5.63±0.07	6.96±0.05	8.63±0.11
100	**3.03±0.02**	5.41±0.05	**6.87±0.05**	8.93±0.12
200	**2.76±0.01**	5.00±0.04	**6.61±0.04**	8.68±0.11

Table 8. Offline error for *mQSO10(5+5^q)* for different number of peaks and dimensions, f=500 and, s=1

m \ D	2	3	4	5
1	9.01±0.24	17.43±1.98	22.81±3.20	28.38±1.89
5	5.15±0.12	6.96±0.45	7.57±0.57	8.80±0.45
10	4.63±0.06	5.59±0.20	**5.50±0.21**	**5.74±0.08**
20	4.36±0.03	5.27±0.07	**6.01±0.06**	**6.80±0.04**
30	4.18±0.03	**5.37±0.07**	**6.49±0.07**	**7.28±0.04**
40	4.04±0.02	**5.49±0.06**	**6.66±0.08**	**7.52±0.04**
50	3.92±0.02	**5.48±0.07**	**6.82±0.06**	**7.67±0.04**
100	3.51±0.01	**5.21±0.05**	**6.92±0.04**	**7.92±0.04**
200	3.09±0.01	**4.73±0.04**	6.76±0.04	**7.91±0.04**

6 Conclusions

In this paper, we have proposed a new particle swarm optimization algorithm called cellular PSO which tackles dynamic environments. Cellular PSO combines the power of particle swarm optimization with cellular automata concepts by embedding cellular automata into the search space and keeping particle density in each cell bellow a specified threshold. With local information exchange between cells, particles in each neighborhood look for a peak together.

Compared to other well known approaches, our proposed model results more accurate solutions in highly dynamic environments, modeled by MPB[16], where peaks change in width and height. Moreover, in the environments where positions of the peaks also change, cellular PSO results less offline error than mQSO where either number of dimensions or number of peaks are small. In addition, experimental results show that this superiority can be achieved without changing value of parameters of cellular PSO. Furthermore, cellular PSO requires less computational effort since unlike mQSO [2] it does not need to compute distance of every pair of swarms on every iteration.

References

1. Blackwell, T.M.: Particle Swarms and Population Diversity. Soft Computing - A Fusion of Foundations, Methodologies and Applications 9, 793–802 (2005)
2. Blackwell, T., Branke, J.: Multiswarms, Exclusion, and Anti-Convergence in Dynamic Environments. IEEE Transactions on Evolutionary Computation 10, 459–472 (2006)
3. Kennedy, J., Eberhart, R.C.: Particle Swarm Optimization. In: IEEE International Conference on Neural Networks, Piscataway, NJ, vol. IV, pp. 1942–1948 (1995)
4. Blackwell, T., Branke, J.: Multi-Swarm Optimization in Dynamic Environments. Applications of Evolutionary Computing, 489–500 (2004)
5. Blackwell, T., Branke, J., Li, X.: Particle Swarms for Dynamic Optimization Problems. Swarm Intelligence, 193–217 (2008)
6. Blackwell, T.: Particle Swarm Optimization in Dynamic Environments. In: Evolutionary Computatation in Dynamic and Uncertain Environments, pp. 29–49 (2007)
7. Lung, R.I., Dumitrescu, D.: A Collaborative Model for Tracking Optima in Dynamic Environments. In: IEEE Congress on Evolutionary Computation, pp. 564–567 (2007)
8. Li, C., Yang, S.: Fast Multi-Swarm Optimization for Dynamic Optimization Problems. In: Fourth International Conference on Natural Computation, Jinan, Shandong, China, vol. 7, pp. 624–628 (2008)
9. Du, W., Li, B.: Multi-Strategy Ensemble Particle Swarm Optimization for Dynamic Optimization. Information Sciences: an International Journal 178, 3096–3109 (2008)
10. Fredkin, E.: Digital Mechanics: An Informational Process Based on Reversible Universal Cellular Automata. Physica D45, 254–270 (1990)
11. Mitchell, M.: Computation in Cellular Automata: A Selected Review. In: Gramss, T., Bornholdt, S., Gross, M., Mitchell, M., Pellizzari, T. (eds.) Nonstandard Computation, pp. 95–140 (1996)
12. Wolfram, S., Packard, N.H.: Two-Dimensional Cellular Automata. Journal of Statistical Physics 38, 901–946 (1985)
13. Waintraub, M., Pereira, C.M.N.A., Schirru, R.: The Cellular Particle Swarm Optimization Algorithm. In: International Nuclear Atlantic Conference, Santos, SP, Brazil (2007)
14. Carlisle, A., Dozier, G.: Adapting Particle Swarm Optimization to Dynamic Environments. In: International Conference on Artificial Intelligence, Las Vegas, NV, USA, vol. 1, pp. 429–434 (2000)
15. Hu, X., Eberhart, R.C.: Adaptive Particle Swarm Optimization: Detection and Response to Dynamic Systems. In: IEEE Congress on Evolutionary Computation, Honolulu, HI, USA, vol. 2, pp. 1666–1670 (2002)
16. Branke, J.: Memory Enhanced Evolutionary Algorithms for Changing Optimization Problems. In: 1999 Congress on Evolutionary Computation, Washington D.C., USA, vol. 3, pp. 1875–1882 (1999)
17. Moser, I.: All Currently Known Publications on Approaches Which Solve the Moving Peaks Problem. Swinburne University of Technology, Melbourne (2007)
18. Janson, S., Middendorf, M.: A Hierarchical Particle Swarm Optimizer for Dynamic Optimization Problems. Applications of Evolutionary Computing, 513–524 (2004)
19. van den Bergh, F.: An Analysis of Particle Swarm Optimizers. Department of Computer Science, PhD. University of Pretoria, Pretoria, South Africa (2002)