A Hibernating Multi-Swarm Optimization Algorithm for Dynamic Environments

Masoud Kamosi
Department of Computer Engineering
Science and Research Branch,
Islamic Azad University
Tehran, Iran
masoud.kamosi@gmail.com

Ali B. Hashemi
Department of Computer Engineering
and Information Technology,
Amirkabir University of Technology
Tehran, Iran
a_hashemi@aut.ac.ir

M.R. Meybodi
Department of Computer Engineering
and Information Technology,
Amirkabir University of Technology
Tehran, Iran
mmeybodi@aut.ac.ir

***Abstract*— many problems in the real world are dynamic in which the environment changes. However, the nature itself provides solutions for adaptation to these changes in order to gain the maximum benefit, i.e. finding the global optimum, at any moment. One of these solutions is hibernation of animals when food is scarce and an animal may use more energy in searching for food than it would receive from consuming the food. In this paper, we applied the idea of hibernation in a multi-swarm optimization algorithm, in which a parent swarm explores the search space and child swarms exploit promising areas found by the parent swarm. In the proposed model, whenever the search efforts of a child swarm for exploiting an area becomes unproductive, the child swarm hibernates. Similar to the nature, which the change of the season awakens hibernating animals, in the proposed model hibernating swarms are awakened upon the detection of a change in the environment. Experimental results on various dynamic environments modeled by the moving peaks benchmark show that the proposed algorithm outperforms other PSO algorithms, including similar particle swarm algorithms for dynamic environments like mQSO, adaptive mQSO, and FMSO.**

Keywords-particle swarm optimization; dynamic environment; multi-swarm optimization

I. INTRODUCTION

The particle swarm optimization algorithm (PSO) is introduced by Kennedy and Eberhart [1]. 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 p and its velocity vector v. In time step t, particle i calculates its new velocity then updates its position according to Equation (1) and Equation (2), respectively.

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

$$p_i(t+1) = p_i(t) + v_i(t+1) \quad (2)$$

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. $pbest_i$ is the best position that particle i has been visited. $gbest$ is the best position found by all particles in the swarm. r_1 and r_2 are uniform random variables in range [0,1].

The standard particle swarm optimization algorithms and its improved variants[2, 3] have been performed well for static environment. Yet, it is shown that PSO, like evolutionary algorithms, must be modified to not only find the optimal solution in a short time, but also to be capable of tracking the solution after a change in the environment occurred. In order to have these capabilities, two important problems should be addressed for designing a particle swarm optimization algorithm for dynamic environments: outdated memory and diversity loss[4]. Outdated memory refers to the condition in which memory of the particles, that is the best location visited in the past and its corresponding fitness, may no longer be valid after a change in the environment [5]. Outdated memory problem is usually solved in one of these two ways: re-evaluating the memory [6] or forgetting the memory [7]. Diversity loss occurs when the swarm converges on a few peaks in the landscape and loses its ability to find new peaks, which is required after the environment changes. There are two approaches to deal with diversity loss problem. In the first approach, a diversity maintenance mechanism runs periodically (or when a change is detected) and re-distributes the particles if the diversity falls below a threshold[8]. In the second approach, diversity is always monitored and as soon as it falls below a threshold, the swarm will be re-diversified.

In multi-swarm approaches to dynamic optimization problems, some swarms may become unproductive. Such swarms, not only do not help to find a better solution, but also consume the precious fitness evaluations. In the previous models, such swarms either are reinitialized [4] or continue their fruitless efforts. In order to prevent this inefficiency, we applied the nature's solution for a similar problem. In nature, when the food becomes scarce and an animal may use more energy in searching for food than it would receive from consuming the food, some animal hibernates until the situation improves, i.e. when the weather is warm enough. In our proposed PSO model, named hibernating multi-swarm optimization (HmSO), a child swarm, whose efforts to find a better local optimum are not productive anymore, hibernates until a change in the environment is detected. Upon detecting a change in the environment, all hibernating child swarms are awakened and begin searching for new peaks. The hibernation of

unproductive child swarms provides the opportunity for the parent swarm, which is responsible for exploring the environment, to create more child swarms, which exploit the promising areas found by the parent swarm. Extensive experiments show that for most tested dynamic multimodal environments, the proposed algorithm outperforms all tested PSO algorithms, i.e. cellular PSO[9, 10], mQSO10(5+5^{q}) [4], adaptive mQSO[11], mPSO[12] and FMSO [13] with which shares the idea of utilizing a parent swarms and child swarms.

The rest of this paper is organized as follows: Section II briefly reviews particle swarm optimization algorithms for dynamic environments introduced in the literature. The proposed algorithm is presented in section III. Section IV presents the experimental results of the proposed algorithm along with comparison with alternative approaches from the literature. Finally, section V concludes this paper.

II. PSO IN DYNAMIC ENVIRONMENTS

Hu and Eberhart proposed re-randomization PSO for optimization in dynamic environments[8] in which some particles randomly are relocated after a change is detected or when the diversity is lost, to prevent losing the diversity. Li and Dam [14] showed that a grid-like neighborhood structure used in FGPSO[15] can perform better than 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 proposed HPSO, a tree-like structure hierarchical PSO [16], 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 [17]. Lung and Dumitresc [18] used two collaborating populations of equal size, one swarm is responsible for preserving the diversity of the particles by using a crowding differential evolutionary algorithm [19] while the other keeps track of global optimum with a PSO algorithm.

Blackwell and Bentley presented a repulsion mechanism in using the analogy of atom particles [20, 21]. In their model, a swarm is comprised of charged and neutral particles. The charged particles repel each other, leading to a cloud of charged particles orbiting a contracting, neutral, PSO nucleus. Moreover, Blackwell et al. extended the idea of charged particles to a quantum model and presented a multi-swarm method [4, 11, 22].

Du and Li [23] suggested an algorithm which divides particles into two parts. The first part uses a standard PSO enhanced by a Gaussian local search to find the global optimum quickly. The second part extends the searching area of the algorithm and patrols around the first part to track the changed global optimum, which possibly escaped from the coverage of the first part. Although their algorithm performs well in the environments with one or two local optima, it cannot find the global optimum when the environment has more local optima.

Li and Yang proposed a fast multi-swarm method (FMSO) which maintains the diversity through the run [13]. 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. Moreover, In [24], the authors 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. [25] introduced compound particle swarm optimization (CPSO) utilizing a new type of particles which helps explore the search space more comprehensively after a change occurred in the environment. In another work, they used composite particles which help quickly find the promising optima in the search space while maintaining the diversity by a scattering operator[26].

Hashemi and Meybodi introduced cellular PSO, a hybrid model of cellular automata and PSO [9, 10]. 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, in [10] 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. [12] 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.

III. THE PROPOSED ALGORITHM

The proposed hibernating multi-swarm optimization algorithm consists of a parent swarm and some child swarms. The parent swarm is responsible for finding promising area in the search space upon which a child swarm is created to exploit the newly found promising area. The proposed algorithm adapts the idea of hibernation of animals to prevent the unproductive search of a child swarm in order to save the precious fitness evaluations for more efficient searches.

In the proposed algorithm, each child swarm is considered as an animal searching for food, i.e. a better solution. As long as it finds a better solution, it remains active and continues searching. Yet, if it cannot find a better solution because its particles have converged to a solution and that solution is not as good as the best solution found by the whole swarm, the child swarm considers its activity unproductive. Therefore, it will hibernate until it can be useful again that is the moment a change is detected in the

environment, similar to the change of the season in the nature, which awaken the hibernating animals. The proposed algorithm, which is called hibernating multi-swarm optimization (HmSO) works as follows.

Algorithm 1 Hibernating multi-Swarm Optimization (HmSO)

```
Initialization the parent swarm
repeat
  if a change is detected in the environment then
    for each particle i in the parent swarm do
        pbest_i = p_i
    end-for
    gbest = arg max_{p_i ∈ parent swarm} (f(p_i))
    for each child swarm c do
      if child swarm c is hibernating then
       awaken child swarm c
      end-if
      for each particle i in child swarm c do
        p_i = a random position in a hyper-sphere
        with radius r_s centered at cbest_c
        pbest_i = p_i
      end-for
      cbest_c = arg max_{p_i ∈ child swarm c} (f(p_i))
    end-for

  else
    //Update the parent swarm
    for each particle i in the parent swarm do
      Update particle's velocity and position
      according to (3) and (2), respectively
      Update pbest_i
      for each swarm c do
        if distance(p_i, cbest_c) < r then
          if f(p_i) > f(cbest_c) then
              cbest_c = p_i
          end-if
          Reinitialize particle i
        end-if
      end-for
    end-for
    cbest_parent = arg max_{p_i ∈ parent swarm} (f(p_i))

    if cbest_parent has been improved then
      //create a new child swarm n around the cbest_parent
      cbest_n = cbest_parent
      for each particle i in the parent swarm do
        if distance(p_i, cbest_n) < r then
          if |child swarm_n| < π then
            copy particle i to the child swarm n
          end-if
          Initialize particle i
        end-if
      end-for
      while |child swarm_n| < π
          Create a new particle p_j in the child
          swarm n
          p_j = a random position in a hyper-
          sphere with radius r/3 centered at
          cbest_n
      end-while
    end-if

    //Update child swarms
    for each child swarm c do
      for each particle i in child swarm c do
        Update particle's velocity and position
        according to (3) and (2), respectively
        Update pbest_i
      end-for
      cbest_c = arg max_{p_i ∈ child swarm c} (f(p_i))
      if radius_c < r_conv and f(cbest_c) < f(gbest)-ξ then
        hibernate child swarm c
      end-if
    end-for
    // check child swarms collision
    for each pair of child swarms (k, l), k• l do
     if distance(cbest_k, cbest_l) < r_excl then
       if f(cbest_k) > f(cbest_l) then
         Destroy child swarm l
       else
         Destroy child swarm k
       end-if
     end-if
    end-for
  end-if
until a termination condition is met
```

After initializing the parent swarm, particles of the parent swarm begin searching in the search space. At each iteration, velocity and position of a particle i in the parent swarm is updated utilizing its best personal position ($pbest_i$) and the best position found by the parent swarm ($cbest_{parent}$) according to (3) and (2), respectively. After updating the position of the particle i, if the fitness of the new position of particle i is better than its best personal position ($pbest_i$), $pbest_i$ will be updated to the new position. Likewise, the best position found in the parent swarm ($cbest_{parent}$) will be updated. Afterwards, the distance between particle i and the attractor of each child swarm c, i.e. the best position found by a child swarm c ($cbest_c$), is calculated. If the distance between particle i and the attractor of a child swarm c is less than r, the attractor of the child swarm c will be updated to the position of particle i and particle i will be reinitialized.

When all particles in the parent swarm are updated, if the best position found in the parent swarm ($cbest_{parent}$) is improved, a new child swarm will be created with $cbest_{parent}$ as its attractor. If there are m particles in the parent swarm whose distances to the attractor of the newly created child swarm are less than r, these particles will be moved to the new child swarm. At the same time, these particles will be replaced by m new initialized particles in the parent swarm. If the number of particles moved to the newly created child swarm (m) is less than the number of particles determined for a child swarm (π), π-m particles for the child swarm will be created and initialized in a hyper sphere with radius r/3 centered at the child swarm's attractor. Afterwards, all particles in every child swarm c update their velocity and position according to (3) and (2), respectively. Then, the personal best position (*pbest*) for all child particles and the child swarm's best position, i.e. the child swarm's attractor $cbest_c$, will be updated.

$$v_i(t+1) = w v_i(t) + c_1 r_1 (pbest_i - p_i(t)) + c_2 r_2 (cbest_c - p_i(t)) \quad (3)$$

At each iteration, if a child swarm c converges to a local optimum, i.e. its radius (the maximum distance between every two particles in a swarm) becomes less than a specified threshold r_{conv} and at the same time the difference between the fitness value of the best position found by the child swarm ($cbest_c$) and the fitness value of the best position found by the whole swarm (*gbest*) is less than a specified threshold ξ, the child swarm c will hibernate until a change in the environment is detected.

Moreover, since the search of two swarm in the same area is not efficient, at the end of each iteration every two child swarms are checked whether they are searching in the same area or not. Two child swarms are searching in the same area (or colliding), if the Euclidian distance between their attractors is less than a specified threshold r_{excl}. In this case, the worse child swarm whose attractor is less fit will be removed.

The proposed algorithm detects a change in the environment by reevaluating *gbest*[7, 8]. If the new fitness value is different, a change in the environment is detected. Upon detecting a change in the environment, the following steps are performed in parallel. The particles in the parent swarm re-evaluate their positions and reset their best personal position (*pbest*) to their current position. All hibernating child swarms are awakened. Then, the particles in each child swarm change their behaviors in the following iteration after the change in the environment and perform a random local search around the best position of their swarm. A particle performs a random search by setting its positions to a random location in a hyper-sphere with radius r_s centered at its swarm's best position. Then, every child particle resets its best personal positions (*pbest*) to its new position and the child swarms attractors will be updated.

IV. EXPERIMENTAL STUDY

In this section, we first describe moving peaks benchmark [27] on which the proposed algorithm is evaluated. Then, experimental settings are described. Finally, experimental results of the proposed algorithm are presented and compared with alternative approaches from the literature.

A. Moving peaks benchmark

Moving peaks benchmark [27] is widely used in the literature to evaluate the performance of optimization algorithms in dynamic environments [28]. In this benchmark, there are some peaks in a multi-dimensional space, where the height, width, and position of each peak alter when the environment changes. Unless stated otherwise, the parameters of the moving peaks benchmark are set to the values presented in Table 1.

In order to measure the efficiency of the algorithms, offline error that is the average fitness of the best position found by the swarm at every point in time (4), is used [29].

$$offline\_error = \frac{1}{T}\sum_{t=1}^{T}\left(Fitness(swarm_{best}(t))\right) \quad (4)$$

where T is the maximum iteration, and $swarm_{best}(t)$ is best position solution by the whole swarm at iteration t.

TABLE 1. DEFAULT SETTINGS OF MOVING PEAKS BENCHMARK

Parameter	Value
number of peaks m	10
frequency of change f	every 5000 FEs
height severity	7.0
width severity	1.0
peak shape	cone
shift length s	1.0
number of dimensions D	5
cone height range H	[30.0, 70.0]
cone width range W	[1, 12]
cone standard height I	50.0
search space range A	[0, 100]

B. Experimental Settings

For the proposed algorithm the acceleration coefficients c_1 and c_2 are set to 1.496180 and the inertial weight w is set to 0.729844 [30]. 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 (r_{excl}) and the radius of random search (r_s) are set to 30.0, 30.0, and 0.5, respectively. The threshold r_{conv} and ξ are set to 1.0 and 5.0, respectively. Moreover, the initial velocity of the parent particles and child particles on each dimension are set to a random value in [-50,50] and [-10,10], respectively.

TABLE 2. DEFAULT PARAMETER SETTINGS OF HMSO

Parameter	Value
r	30
r_{excl}	30
r_s	0.5
ξ	5
Size of the parent swarm	5
Size of each child swarm	10

HmSO is compared with mPSO[12], cellular PSO[9, 10], mQSO10(5+5^q) [4], adaptive mQSO[11], and FMSO [13]. For mQSO we adapted a configuration 10(5+5^q) which creates a 10 swarms with 5 neutral (standard) particles and 5 quantum particles with r_{cloud}=0.5 and r_{excl}= r_{conv} =31.5, as suggested in [4, 11]. For adaptive mQSO, we considered unlimited number of sub-swarms, each contains 5 standard particles and one quantum particle[11]. 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[13]. For cellular PSO, a 5-Dimensional cellular automaton with 10^5 cells and Moore neighborhood with radius of two cells is 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_s) and the cell capacity θ are set to 0.5 and 10, respectively.

C. Experimental Results

For all algorithms, we reported the average offline error with 95% confidence interval for 100 runs. Offline error of HmSO, mPSO[12], cellular PSO[9, 10], mQSO10(5+5^q) [4], adaptive mQSO[11], and FMSO [13] for different dynamic environment is presented in table III to table VII. For each environment, the offline errors of the algorithms are compared statistically and the result of the best performing algorithm(s) with 95% confidence is printed in bold. When

the offline errors of the best two (or more) algorithms are not statistically different, all are printed in bold.

As depicted in the table III to table VII, the proposed HmSO outperforms other tested PSO algorithms, including FMSO, for almost all environments. Only in the uni-peak environments and when the environment changes very quickly (f=500) and contains less than 10 peaks (table III), adaptive mQSO results in less offline error than the proposed algorithms. The reason for the good performance of the proposed algorithm is that HmSO finds better solutions faster than other algorithms after a change occurs in the environment, especially at the early iterations (Fig. 1).

TABLE III. OFFLINE ERROR ±STANDARD ERROR FOR f=500

m	HmSO	mPSO	CellularPSO	mQSO	Adaptive mQSO	FMSO
1	8.53±0.49	8.71±0.48	22.37±3.8	36.52±3.2	**5.08±0.27**	27.58±0.9
5	7.40±0.31	6.69±0.26	14.20±0.6	13.50±0.8	**5.14±0.09**	19.45±0.4
10	7.56±0.27	7.19±0.23	13.55±0.5	11.18±0.4	**6.20±0.11**	18.26±0.3
20	7.81±0.20	8.01±0.19	12.77±0.3	10.54±0.2	**6.94±0.18**	17.34±0.3
30	8.33±0.18	8.43±0.17	12.55±0.4	10.37±0.2	**7.23±0.16**	16.39±0.4
40	8.45±0.18	8.62±0.18	12.33±0.3	10.32±0.2	**7.43±0.17**	15.34±0.4
50	8.83±0.17	8.76±0.18	12.19±0.3	10.33±0.2	**7.49±0.09**	15.54±0.2
100	8.85±0.16	8.91±0.17	11.38±0.2	9.93±0.21	**7.29±0.15**	12.87±0.6
200	8.85±0.16	8.88±0.14	11.34±0.2	9.67±0.20	**6.82±0.14**	11.52±0.6

TABLE IV. OFFLINE ERROR ±STANDARD ERROR FOR f=1000

m	HmSO	mPSO	CellularPSO	mQSO	Adaptive mQSO	FMSO
1	4.46±0.26	4.44±0.24	7.97±0.54	19.1±1.5	**2.68±0.14**	14.42±0.4
5	4.27±0.08	3.93±0.16	6.16±0.31	7.59±0.39	**3.22±0.07**	10.59±0.2
10	4.61±0.07	4.57±0.18	5.94±0.23	6.33±0.23	**4.11±0.08**	10.40±0.1
20	**4.66±0.12**	4.97±0.13	6.13±0.17	6.46±0.20	**4.75±0.14**	10.33±0.1
30	**4.83±0.09**	5.15±0.12	6.23±0.15	6.51±0.16	4.98±0.10	10.06±0.1
40	**4.82±0.09**	5.17±0.10	6.27±0.15	6.43±0.18	5.10±0.11	9.85±0.1
50	**4.96±0.03**	5.33±0.10	6.26±0.16	6.67±0.16	5.12±0.05	9.54±0.1
100	**5.14±0.08**	5.60±0.09	6.27±0.14	6.34±0.12	**5.03±0.09**	8.77±0.0
200	5.25±0.08	5.78±0.09	6.01±0.11	6.13±0.12	**4.65±0.09**	8.06±0.0

TABLE V OFFLINE ERROR ±STANDARD ERROR FOR f=2500

m	HmSO	mPSO	CellularPSO	mQSO	Adaptive mQSO	FMSO
1	1.75±0.10	1.79±0.10	4.57±0.31	7.79±0.72	**1.09±0.06**	6.29±0.20
5	1.92±0.11	2.04±0.12	3.15±0.21	3.53±0.18	**1.58±0.13**	5.03±0.12
10	**2.39±0.16**	2.66±0.16	3.09±0.16	3.20±0.14	**2.33±0.11**	5.09±0.09
20	**2.46±0.09**	3.07±0.11	3.60±0.13	3.83±0.12	2.84±0.09	5.32±0.08
30	**2.57±0.05**	3.15±0.08	3.88±0.12	4.03±0.12	3.13±0.09	5.22±0.08
40	**2.56±0.06**	3.17±0.07	4.17±0.12	3.90±0.11	3.23±0.08	5.09±0.06
50	**2.65±0.05**	3.26±0.07	4.25±0.12	3.95±0.10	3.24±0.07	4.99±0.06
100	**2.72±0.04**	3.31±0.05	4.25±0.13	3.81±0.10	3.20±0.06	4.60±0.05
200	**2.81±0.04**	3.36±0.05	4.20±0.09	3.66±0.07	3.00±0.05	4.34±0.04

TABLE VI. OFFLINE ERROR ±STANDARD ERROR FOR f=5000

m	HmSO	mPSO	CellularPSO	mQSO	Adaptive mQSO	FMSO
1	0.87±0.05	0.90±0.05	2.79±0.19	4.06±0.40	**0.55±0.02**	3.44±0.11
5	1.18±0.04	1.21±0.12	1.94±0.18	1.98±0.10	**1.00±0.04**	2.94±0.07
10	**1.42±0.04**	1.61±0.12	1.93±0.08	1.93±0.09	**1.43±0.04**	3.11±0.06
20	**1.50±0.06**	2.05±0.08	2.73±0.12	2.59±0.11	1.95±0.05	3.36±0.06
30	**1.65±0.04**	2.18±0.06	3.08±0.11	2.84±0.08	2.15±0.05	3.28±0.05
40	**1.65±0.05**	2.24±0.06	3.28±0.11	2.74±0.08	2.28±0.04	3.26±0.04
50	**1.66±0.02**	2.34±0.06	3.34±0.07	2.74±0.07	2.28±0.02	3.22±0.05
100	**1.68±0.03**	2.32±0.04	3.48±0.11	2.61±0.06	2.31±0.03	3.06±0.04
200	**1.71±0.02**	2.34±0.03	3.37±0.08	2.45±0.05	2.11±0.03	2.84±0.03

TABLE VII. OFFLINE ERROR ±STANDARD ERROR FOR f=10000

m	HmSO	mPSO	CellularPSO	mQSO	Adaptive mQSO	FMSO
1	0.45±0.02	0.44±0.02	1.63±0.12	2.24±0.19	**0.27±0.01**	1.90±0.06
5	0.71±0.12	0.72±0.08	1.20±0.15	1.07±0.07	**0.54±0.09**	1.75±0.06
10	**0.94±0.09**	1.05±0.10	1.19±0.09	1.19±0.09	**0.86±0.04**	1.91±0.04
20	**1.02±0.06**	1.37±0.06	2.13±0.10	1.86±0.09	1.24±0.06	2.16±0.04
30	**1.13±0.04**	1.50±0.06	2.59±0.12	2.09±0.08	1.43±0.05	2.18±0.04
40	**1.09±0.04**	1.53±0.05	2.72±0.09	2.02±0.06	1.57±0.05	2.21±0.03
50	**1.10±0.03**	1.62±0.04	2.91±0.11	2.03±0.07	1.58±0.02	2.60±0.08
100	**1.08±0.02**	1.62±0.03	3.02±0.11	1.94±0.05	1.65±0.04	2.20±0.03
200	**1.07±0.02**	1.64±0.02	2.94±0.10	1.79±0.04	1.50±0.03	2.00±0.02

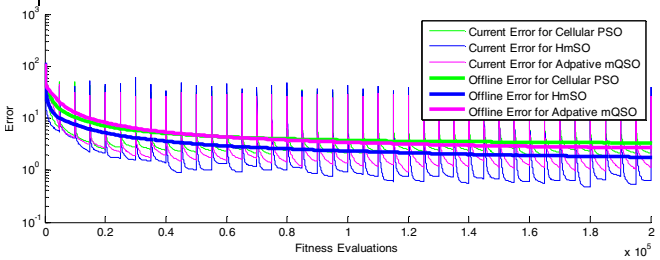

Figure 1. Convergence of offline error over time for a dynamic environment with 50 peaks and f=5000.

D. *Effect of varying the convergence threshold r_{conv}*

This experiment examines the effect of the convergence threshold (r_{conv}) on offline error. As depicted in Fig. 2 and Fig. 3, the optimal value for r_{conv} for different environments is near 1.0. Either increasing or decreasing the size of r_{conv} from its optimal value (r_{conv} =1) increases offline error monotonically. The reason is that setting r_{conv} to a too large value causes early hibernation of the child swarm and prevents the child swarms to exploit an area enough. Hence, the algorithm looses its exploitation capability and the offline error increases. On the other hand, if r_{conv} is set to a too small value, the algorithm loses its hibernation capability. Therefore, the converged child swarms remain active and consume the expensive fitness evaluations.

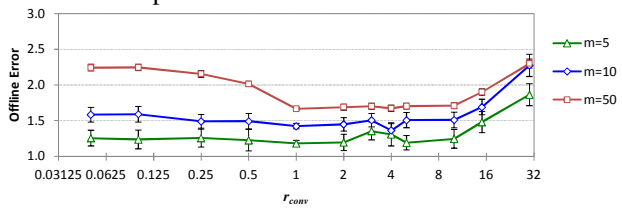

Figure 2. The effect of convergence threshold r_{conv} on offline error for environments with different number of peaks, f=5000.

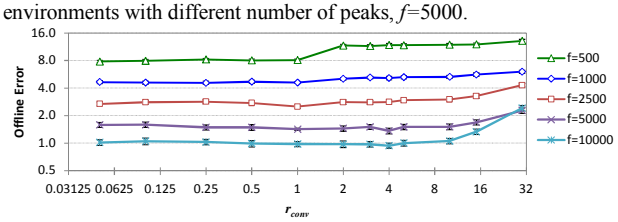

Figure 3. The effect of convergence threshold r_{conv} on offline error for environments with different frequency of change, m=10.

E. *Effect of varying the parametet ξ*

This experiment examines the effect of the parameter ξ, i.e. the hibernation threshold for the fitness of the best position of a child swarm, on offline error. As depicted in Fig. 4 and Fig. 5, the optimal value for ξ for different environments is about 5.0. Either increasing or decreasing

the value of ξ from its optimal value (ξ =5.0) increases offline error monotonically. This is because too large values for ξ prevents the algorithm to exploit a possible better solution than *gbest*. Conversely, setting ξ to a too small value causes late hibernation (or no hibernation), hence it results in losing the hibernation capability of the algorithm.

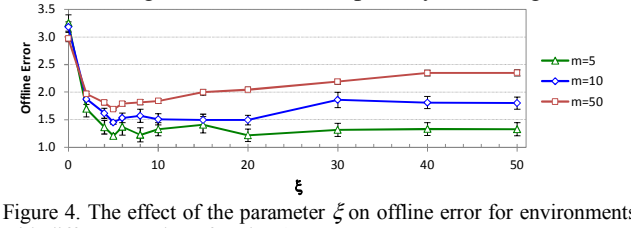

Figure 4. The effect of the parameter ξ on offline error for environments with different number of peaks, f=5000.

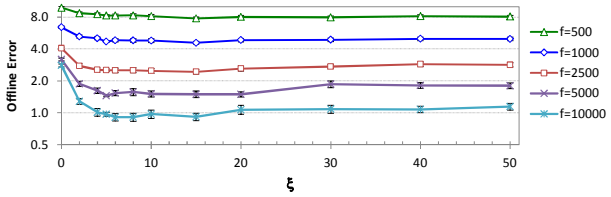

Figure 5. The effect of the parameter ξ on offline error for environments with different frequency of change, m=10.

F. *Effect of varying the size of the parent swarm*

This experiment examines the effect of the size of the parent swarm, i.e. the number of particles in the parent swarm, on offline error. The visualized results in Fig. 6 and Fig. 7 depicts that when there are many peaks in the environment (m=50 or m=100) or when the environment changes slowly (f=5000 or f=10000) the size of the parent swarm affect the offline error less significantly. For other environments, offline error escalates by increasing the size of the parent swarm. However, when there are about five particles in the parent swarm, HmSO results in the least offline error in all tested environments (Fig. 6 and Fig. 7) .

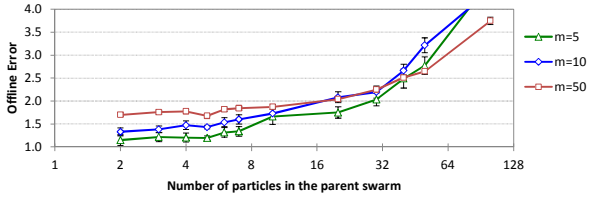

Figure 6. The effect of the size of the parent swarm on offline error for environments with different number of peaks, f=5000.

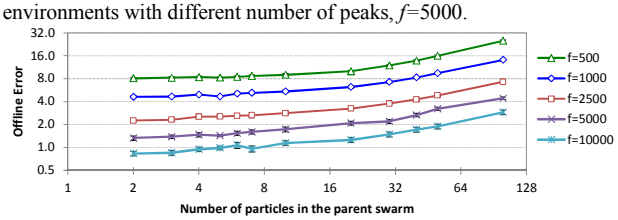

Figure 7. The effect of the size of the parent swarm on offline error for environments with different frequency of change, m=10.

G. *Effect of varying the size of the child swarms*

This experiment examines the effect of the size of the child swarms (π), i.e. the number of particles in each child swarm, on offline error. As depicted in Fig. 8 and Fig.9, the optimal value for the number of particles in the child swarms for different environments is 10 particles. In addition, either increasing or decreasing the number of particles in the child swarms from its optimal value (π=10) results in an increases in offline. The reason is that existence of many particles in the child swarms not only does not help finding better solutions but also consumes the precious function evaluations. Conversely, when there are too few particles in the child swarms, the child swarms cannot exploit an area effectively thereby increasing offline error.

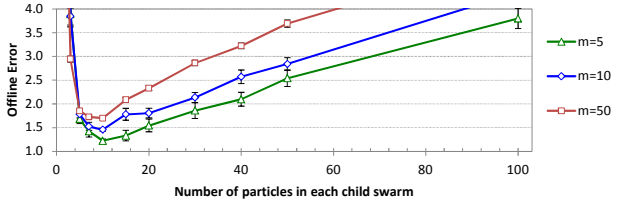

Figure 8. The effect of the number of particles in each child swarm on offline error for environments with different number of peaks, f=5000.

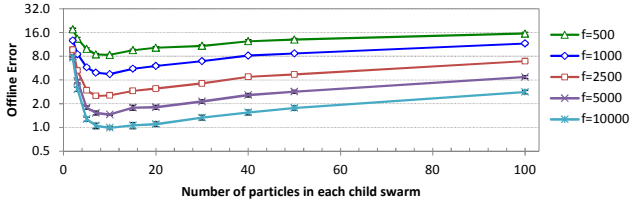

Figure 9. The effect of the number of particles in each child swarm on offline error for environments with different frequency of change, m=10.

V. CONCLUSION

In this paper, we proposed a hibernating multi-swarm optimization algorithm (HmSO) for dynamic environment environments. The proposed PSO consists of a parent swarm and some child swarms. The parent swarm is responsible for exploring the search space and finding promising regions containing a local optimum. Child swarms exploit promising regions found by the parent swarm while maintaining the diversity by removing worse child swarm when two child swarms collide. In order to keep the algorithm efficient, the child swarms whose efforts are not productive, hibernate until a change in the environment is detected. This hibernation, like the hibernation of animals that prevents losing the energy for searching foods that would not be enough to replace the spent energy, prevents the fruitless search of converged swarms and saves the precious fitness evaluations. Hence, it provides the opportunity to search other areas of the search space more effectively. The proposed algorithm introduces two parameters to detect a suitable swarm for hibernation. However, the results of extensive experiments show that for each parameter there is a value which performs the best in all environment. Moreover, the experiments on various dynamic environments modeled by the moving peaks benchmark show that the proposed algorithm outperforms all tested PSO algorithms including mPSO[12], cellular PSO[9, 10], mQSO10($5+5^q$) [4], adaptive mQSO[11], and FMSO [13] in most multi peaks environments.

REFERENCES

[1] J. Kennedy and R. C. Eberhart, "Particle Swarm Optimization," in *IEEE International conference on neural networks*, Piscataway, NJ, USA, 1995, pp. 1942-1948.

[2] Y. del Valle, G. K. Venayagamoorthy, S. Mohagheghi, J. C. Hernandez, and R. G. Harley, "Particle Swarm Optimization:

Basic Concepts, Variants and Applications in Power Systems," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 2, pp. 171-195, 2008.

[3] A. B. Hashemi and M. R. Meybodi, "A Note on the Learning Automata Based Algorithms for Adaptive Parameter Selection in PSO," *Applied Soft Computing, to be published*, 2010.

[4] T. Blackwell and J. Branke, "Multiswarms, Exclusion, and Anti-Convergence in Dynamic Environments," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 4, pp. 459-472, 2006.

[5] T. Blackwell, "Particle Swarm Optimization in Dynamic Environments," in *Evolutionary Computation in Dynamic and Uncertain Environments*, Studies in Computational Intelligence, vol. 51, 2007, pp. 29-49.

[6] R. C. Eberhart and Y. Shi, "Tracking and Optimizing Dynamic Systems with Particle Swarms," in *IEEE Congress on Evolutionary Computation*, Seoul, Korea, 2001, pp. 94-100.

[7] A. Carlisle and G. Dozier, "Adapting Particle Swarm Optimization to Dynamic Environments," in *International Conference on Artificial Intelligence*, Las Vegas, NV, USA, 2000, pp. 429-434.

[8] X. Hu and R. C. Eberhart, "Adaptive Particle Swarm Optimization: Detection and Response to Dynamic Systems," in *IEEE Congress on Evolutionary Computation*, Honolulu, HI, USA, 2002, pp. 1666-1670.

[9] A. B. Hashemi and M. R. Meybodi, "Cellular PSO: A PSO for Dynamic Environments," in *Advances in Computation and Intelligence*, Lecture Notes in Computer Science, vol. 5821/2009, 2009, pp. 422-433.

[10] A. B. Hashemi and M. R. Meybodi, "A Multi-Role Cellular PSO for Dynamic Environments," in *14th International CSI Computer Conference*, Tehran, Iran, 2009, pp. 412-417.

[11] T. Blackwell, J. Branke, and X. Li, "Particle Swarms for Dynamic Optimization Problems," in *Swarm Intelligence*, Natural Computing Series, vol. Part II, 2008, pp. 193-217.

[12] M. Kamosi, A. B. Hashemi, and M. R. Meybodi, "A New Particle Swarm Optimization Algorithm for Dynamic Environments," in *International Conference on Swarm, Evolutionary and Memetic Computing*, Chennai, India, 2010.

[13] C. Li and S. Yang, "Fast Multi-Swarm Optimization for Dynamic Optimization Problems," in *Fourth International Conference on Natural Computation*, Jinan, Shandong, China, 2008, pp. 624-628.

[14] X. Li and K. H. Dam, "Comparing Particle Swarms for Tracking Extrema in Dynamic Environments," in *IEEE Congress on Evolutionary Computation*, Canberra, Australia, 2003, pp. 1772-1779.

[15] J. Kennedy and R. Mendes, "Population Structure and Particle Swarm Performance," in *Evolutionary Computation Congress*, Honolulu, Hawaii, USA, 2002, pp. 1671-1676.

[16] S. Janson and M. Middendorf, "A Hierarchical Particle Swarm Optimizer for Dynamic Optimization Problems," in *Applications of Evolutionary Computing*, Lecture Notes in Computer Science, vol. 3005, 2004, pp. 513-524.

[17] S. Janson and M. Middendorf, "A Hierarchical Particle Swarm Optimizer for Noisy and Dynamic Environments," *Genetic Programming and Evolvable Machines*, vol. 7, no. 4, pp. 329-354, 2006.

[18] R. I. Lung and D. Dumitrescu, "A Collaborative Model for Tracking Optima in Dynamic Environments," in *IEEE Congress on Evolutionary Computation*, Singapore, 2007, pp. 564-567.

[19] R. Thomsen, "Multimodal Optimization Using Crowding-Based Differential Evolution," in *IEEE Congress on Evolutionary Computation*, Portland, Oregon, USA, 2004, pp. 1382-1389.

[20] T. Blackwell, "Swarms in Dynamic Environments," in *Genetic and Evolutionary Computation — Gecco 2003*, Lecture Notes in Computer Science, vol. 2723, 2003, pp. 200-200.

[21] T. M. Blackwell and P. J. Bentley, "Dynamic Search with Charged Swarms," in *Genetic and evolutionary computation conference*, New York, NY, USA, 2002, pp. 19-26.

[22] T. Blackwell and J. Branke, "Multi-Swarm Optimization in Dynamic Environments," in *Applications of Evolutionary Computing*, Lecture Notes in Computer Science, vol. 3005, Springer Berlin / Heidelberg, 2004, pp. 489-500.

[23] W. Du and B. Li, "Multi-Strategy Ensemble Particle Swarm Optimization for Dynamic Optimization," *Information Sciences*, vol. 178, no. 15, pp. 3096-3109, 2008.

[24] C. Li and S. Yang, "A Clustering Particle Swarm Optimizer for Dynamic Optimization," in *IEEE Congress on Evolutionary Computation*, 2009, pp. 439-446.

[25] L. Liu, D. Wang, and S. Yang, "Compound Particle Swarm Optimization in Dynamic Environments," in *Applications of Evolutionary Computing*, Lecture Notes in Computer Science, vol. 4974, 2008, pp. 616-625.

[26] L. Liu, S. Yang, and D. Wang, "Particle Swarm Optimization with Composite Particles in Dynamic Environments," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. PP, no. 99, pp. 1-15, 2010.

[27] J. Branke, "Memory Enhanced Evolutionary Algorithms for Changing Optimization Problems," in *1999 Congress on Evolutionary Computation*, Washington D.C., USA, 1999, pp. 1875-1882.

[28] I. Moser, "All Currently Known Publications on Approaches Which Solve the Moving Peaks Problem," Swinburne University of Technology, Melbourne, Australia, 2007.

[29] J. Branke and H. Schmeck, "Designing Evolutionary Algorithms for Dynamic Optimization Problems," in *Advances in Evolutionary Computing: Theory and Applications*, Natural Computing Series, Springer-Verlag New York, Inc., 2003, pp. 239-262.

[30] F. van den Bergh, "An Analysis of Particle Swarm Optimizers," Ph.D. dissertation, Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2002.