2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

A Multi-swarm Particle Swarm Optimization with Orthogonal Learning for Locating and Tracking Multiple Optimization in Dynamic Environments

Ruochen Liu, *Member, IEEE,* Xu Niu, Licheng Jiao *IEEE Senior, Member*，*Jingjing Ma*, *Member, IEEE*

Abstract—**Due to the specificity and complexity of the dynamic optimization problems (DOPs), those excellent static optimization algorithms cannot be applied in these problems directly. So some special algorithms only for DOPs are needed. There is a multi-swarm algorithm with a better performance than others in DOPs, which utilizes a parent swarm to explore the search space and some child swarms to exploit promising areas found by the parent swarm. In addition, a static optimization algorithm OLPSO is so attractive，which utilize an orthogonal learning (OL) strategy to utilize previous search information (experience) more efficiently to predict the positions of particles and improve the convergence speed. In this paper, we bring the essence of OLPSO called OL strategy to the multi-swarm algorithm to improve its performance further. The experimental results conducted on different dynamic environments modeled by moving peaks benchmark show that the efficiency of this algorithm for locating and tracking multiple optima in dynamic environments is outstanding in comparison with other particle swarm optimization models, including MPSO, a similar particle swarm algorithm for dynamic environments.**

I. INTRODUCTION

Many real-word optimization problems are dynamic in which global optimum and local optima change over time. This requires an algorithm to not only find the global optimal solution under a specific environment but also track the trajectory of the changing optima over dynamic environments.But most research on evolutionary algorithms (EAs) focuses on static optimization problems. So optimization methods that are able to continuously adapt to a changing environment are needed.

Particle swarm optimizer (PSO) is a versatile population based stochastic optimization technique. The standard particle swarm optimization algorithms and its variants [5], [6] have been performed well for static environment.

But in the standard PSO for static environment, particles in the swarm will usually congregate to local or global optima on a few peaks in the landscape and lose its ability to find new peaks after several successive iterations, which is required after the environment changes. This phenomenon is called diversity loss. When a change occurs and the optima in the search space moves, the best location obtained in the past and its corresponding fitness may no longer be valid to give the particle a sufficient exploration ability to track new optima. This memory of the particles is the so called outdated memory.

For the reasons above, these algorithms cannot be directly applied into dynamic environments. Dynamic environment requires an algorithm not only to find the global optimal solution under a specific environment but also to track the trajectory of the changing optima over dynamic environments. In order to have these capabilities, two important problems: outdated memory and diversity loss [7] should be solved for designing a particle swarm optimization algorithm for DOPs.

Several PSO algorithms have been recently proposed to address DOPs [7], [23], [24], of which using multi-swarms seems a good technique. The multi-swarm method can be used to enhance the diversity of the swarm, with the aim of maintaining multiple swarms on different peaks. Outdated memory problem is usually solved in one of these two ways: re-evaluating the memory [22] or forgetting the memory [25].

A new multi-swarm algorithm (MPSO) for dynamic environments has been proposed by Masoud Kamosi [18], which introduce two types of swarm to address the diversity loss problema. A parent swarm is applied to explore the whole search space to find promising area containing local optima and several child swarms, each of which is nocn-overlapping and responsible for exploiting a promising area found by the parent swarm to get the optima. Inspired by this idea, we proposed OLMPSO which contain the same idea of multi-swarm in [18]. Parent swarm uses the standard PSO [10] with the local neighborhood (lbest) [11], in which a particle is only affected by the best experience of its own swarm rather than of all particles in the environment. The main difference between OLMPSO and the algorithm in [18] is that orthogonal learning particle swarm optimization (OLPSO)[12] has been used to get faster PSO convergence speed and higher solution accuracy. Owing to the orthogonal experimental design (OED) orthogonal test ability and prediction ability, the orthogonal learning OL strategy [12], which is utilized in OLPSO, could construct a guidance exemplar with an ability to predict promising search directions toward the global optimum. And this modified variant PSO will be described

This work was supported by the National Natural Science Foundation of C hina (Nos. 61373111, 61272279, 61103119 and 61203303); the Fundamen tal Research Funds for the Central Universities (Nos. K50511020014, K50 51302084, K50510020011, K5051302049, and K5051302023); the Fund f or Foreign Scholars in University Research and Teaching Programs (the 11 1 Project) (No. B07048);and the Program for New Century Excellent Tale nts in University (No. NCET-12-0920).

All authors are with the Key Laboratory of Intelligent Perception and Image Understanding of Ministry of Education of China, Institute of Intelligent Information Processing, Xidian University, Xi'an 710071, China. (email: rcliu@ieee.org).

in detail later. Moreover, for tracking the changing local optima, particles in every child swarm performs a random search around the previous best position of that child swarm after a change is detected in the environment. And experiments show that for all tested dynamic environments the proposed algorithm OLMPSO outperforms in the vast majority of all tested PSO algorithms, including MPSO [18], with which shares the idea of utilizing a parent swarms and child swarms.

The rest of this paper is outlined as follows. Section II describes the background of Multi-Swarm Strategy, orthogonal experimental design (OED), orthogonal learning (OL) strategy used in this paper and the modified PSO applied in child swarms in detail. The proposed algorithm is presented in section III. The experimental results of the proposed algorithm along with comparison with alternative approaches from the literature are given in Section IV. Finally, Section V concludes this paper with discussions on relevant future work.

II. Related Background

A. Particle Swarm Optimization (PSO)

The particle swarm optimization (PSO) was first introduced by Kennedy and Eberhart [1], which was inspired by the social behavior of bird flocking. In PSO, a potential solution for a problem is considered as a bird, which is named particle here, flies through a D-dimensional search space and adjusts its position according to the previous best position found by its neighborhood and the previous best position found by itself. Corresponding to two different kinds of the neighborhood, there are two main models of the PSO algorithm, called gbest (global best) and lbest (local best), respectively. In gbest model, the neighborhood of a particle consists of the particles in the whole swarm. But in the lbest model, the neighborhood of a particle is defined by several fixed particles.

Several major versions of the PSO algorithm have been developed due to its property of fast convergence for many academic and real world problems with promising results since PSO was first introduced [2]. Each particle i is represented by a position vector X_i and a velocity vector V_i, which are updated in the version of PSO with an inertia weight [3] as follows:

$$v_i'^d = \omega v_i^d + c_1 r_1 (p_i^d - x_i^d) + c_2 r_2 (p_g^d - x_i^d) \quad (1)$$

$$x_i'^d = x_i^d + v_i'^d \quad (2)$$

where ω 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. $v_i'^d$ and v_i^d represent the current and previous position in the dth dimension of particle i. p_i^d and p_g^d is the best position that particle i has been visited and found by all particles in the swarm in the dth dimension. r_1 and r_2 are uniform random variables in range [0,1].

In the standard PSO, both gbest model and lbest model, the information of a particle's best experience and its neighborhood's best experience is utilized in a simple way, where the flying is adjusted by a simple learning summation of the two experiences which will be given in (1) in Section I. But one exemplar may have good values on some dimensions of the solution vector while the other exemplar may have good values on some other dimensions. So the particle may suffer from the "two steps forward, one step back" phenomenon [13]. In all, this simple way is not efficient to make the best use of the search information in these two experiences.

Earlier studies have shown that orthogonal experimental design (OED) offers an ability to discover the best combination for different factors with a small number of experimental samples [14], [15]. In this paper, the OED is used to construct a promising learning exemplar, i.e. orthogonal learning (OL) strategy, which is used to discover the best combination of a particle's best historical position and its neighborhood's best historical position.

B. Multi-Swarm Strategy

The multi-swarm approach is a considerable technique for maintaining the swarm diversity to address the convergence problem of PSO for dynamic optimal problems. The multi-swarm method can be used to maintain multiple swarms on different peaks, which are referred to as the optima in this paper, with the purpose of tracking different local optima in dynamic optimal problems. For the multi-swarm method to work, the whole search space can be divided into several sub-regions. Each sub-region might contain one or more than one peak and each sub-swarm covers one sub-region and exploits it. So many researchers have considered multi-populations as a means of enhancing the diversity of EAs to address dynamic optimal problems [8, 9].

III. OLPSO

Using the OED method, the original PSO can be modified as an OLPSO with an OL strategy. In this section, the OED method and implementation of the OL strategy are presented. The complete OLPSO algorithms are derived at the end of the section.

A. Orthogonal Experimental Design (OED)

Scientific experiment usually involves three or more factors, which requires multi-factor analysis. Multi-factor experiments include full factorial design and fractional factorial design. Full factorial design tests all possible combinations of factors. For a full factorial experiment with 10 factors and 3 levels, the number of trials is 3^{10}= 59,049, which is so large that it is difficult to be implemented.

An efficient way to study the effect of several factors simultaneously is to use the OED with both the OA and the factor analysis，which selects representative points from full factorial experiment in a way that the points are distributed uniformly within the test range and thus can represent the overall situation. So the OED is highly efficient for the arrangement of multi-factor experiment with optimal combination levels.

In order to illustrate how to use the OED, a simple example is shown in Table I, which is a maximization problem. The objective function is given as follows:

$$\text{Maximize } y(x_1, x_2, x_3)=100x_1 - 10x_2 - x_3) \quad (3)$$

$$x_1 \in \{1,2\}, x_2 \in \{3,4\}, x_3 \in \{5,6\}$$

This example of maximization problem shows that three factors, which will affect maximizing results, are the x_1, x_2 and x_3, denoted as factors A, B, and C, respectively. Moreover, there are two levels (different choices) involved in each factor. Thus, there are in total 2^3= 8 combinations of experimental designs. However, with the help of OED, one can obtain or predict the best combination by testing only few representative experimental cases.

For the example (3), the $L_4(2^3)$ OA given by (4) is suitable [15].

1) Orthogonal Array:

The OED method works on a predefined table called an orthogonal array (OA), which is a fractional factorial array. An OA with N factors and Q levels per factor is always denoted by $L_M(Q^N)$, where L denotes the orthogonal array and M is the number of combinations of test cases.

$$L_4(2^3)= \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{bmatrix} \quad (4)$$

The OA in (4) has 3 columns, meaning that it is suitable for the problems with at most 3 factors. For example, the three columns in the first row is [1, 1, 1], meaning that in this experiment, the first factor (x_1), the second factor (x_2), and the third factor (x_3) are all designed to the first level, that is, 1, 3, and 5 as given in Table I. Similarly, combination of [1, 2, 2] is used in the second experiment, and so on. The total of four experiments specified by the $L_4(2^3)$ are presented in Table I.

TABLE I
Deciding the Best Combination Levels of the Maximization Problem Factors Using an OED Method

Combinations	A: x_1	B: x_2	C: x_3	Results
1	(1) 1	(1) 3	(1) 5	Y_1=65
2	(1) 1	(2) 4	(2) 6	Y_2=54
3	(2) 2	(1) 3	(2) 6	Y_3=164
4	(2) 2	(2) 4	(1) 5	Y_4=155
Levels	**Factor Analysis**			
L1	(Y1+Y2)/2=59.5	(Y1+Y3)/2=114.5	(Y1+Y4)/2=110	
L2	(Y3+Y4)/2=159.5	(Y2+Y4)/2=104.5	(Y2+Y3)/2=109	
OED Results	**A2**	**B1**	**C1**	

2) Factor analysis:

After evaluation of the combinations, the summarized data are analyzed using factor analysis to rank the most effective factors, and discover the best combination of levels for each factor such that the function is optimized. The FA results are shown in Table I and the process is described as follows.

Let y_m denote the experimental result of the mth ($1 \le m \le M$) combination and define the main effect of factor n with qth ($1 \le q \le Q$) level as S_{nq}. The calculation of S_{nq} is to add up all the y_m in which the level is q in the nth factor, and then divide the total count of z_{mnq}, as shown in (5) where z_{mnq} is 1 if the mth experimental test is with the qth level of the nth factor, otherwise, z_{mnq} is 0

$$S_{nq} = \frac{\sum_{m=1}^{M} y_m \times z_{mnq}}{\sum_{m=1}^{M} z_{mnq}} \quad (5)$$

For the example shown in Table II, when we calculate the effect of level 1 on factor A, denoted by element A1, the experimental results of *C*1, and *C2* are summed up firstly for (5) because only these two combinations are involved in level 1 of factor A. Then, the sum divides the combination number (2 in this case) to yield *S*nq. With all the *S*nq calculated, the level of each factor that provides the highest-quality *S*nq will be selected to be the best combination of the levels. For maximization problem, the larger the *S*nq is, the better the *q*th level on factor n will be. Otherwise, vice versa. As in the maximization example shown in Table II, the best result is the combination of *A*2, *B*1, and *C*1, which does not exist in the four combinations tested, but discovered by the FA process.

B. Orthogonal Learning (OL) Strategy

Using the OED method, the original PSO can be modified as an OLPSO with an OL strategy that combines information of P_i and P_n to form a better guidance vector P_0. The particle's flying velocity is not the same as in (1) and changed as

$$V_{id} = \omega V_{id} + \mathrm{cr_d}(\mathrm{p_{0d}} - \mathrm{x_{id}}) \quad (6)$$

where ω is the same as in (1) and c is fixed to be 2.0, the same as c_1 and c_2, and r_d is a random value uniformly generated within the interval [0, 1].

The guidance vector P_0 is constructed for each particle i, from P_i and P_n as

$$P_0 = P_0 \oplus P_n \quad (7)$$

where the symbol $\oplus$ stands for the OED operation. Therefore, the value P_0 is the combination of P_i and P_n as the construct result of OED. In order to avoid the guidance changing the direction frequently, before the vector P_0 cannot lead the particle to a better position any more, it will not be used as the exemplar for a certain number of generations. For example, if the personal best position P_i has not been improved for G generations, then particle *i* will reconstruct a new P_0 by using P_i and P_n [12].

The construction process of P_0 is described as the following six steps.

Step 1) An OA is generated as $L_M(2^D)$ where $\mathbf{M}=2^{\lfloor log_2(D+1) \rfloor}$, using the procedure as given in Appendix.

Step 2) Make up M tested solutions $X_j (1 \le \mathbf{j} \le \mathbf{M})$ by selecting the corresponding value from P_i ore P_n according to the OA. Here, if the level value in the OA is 1, then the corresponding factor (dimension) selects P_i; otherwise, selects P_n.

Step 3) Evaluate each tested solution X_j ($1 \le \mathbf{j} \le \mathbf{M}$), and record the best (with best fitness) solution X_b.

Step 4) Calculate the effect of each level on each factor and determine the best level for each factor using (5).

Step 5) Derive a predictive solution X_p with the levels determined in Step 4 and evaluate X_p.

Step 6) Compare $f(X_b)$ and $f(X_p)$ and the level combination of the better solution is used to construct the vector P_0.

In the above process, the factors are the dimensions of the problem and the levels of each dimension (factor) are the two choices of a particle's best position value and its neighborhood's best position value on this corresponding dimension.

C. Orthogonal Learning Particle Swarm Optimization (OLPSO)

The OL strategy has a strong ability to construct a guidance exemplar to predict promising search directions toward the global optimum because of the OEDs orthogonal prediction ability. The OL strategy is expected to bring better learning efficiency to PSO, as well as faster PSO convergence speed, higher solution accuracy and better global optimization performance. This learning strategy can be applied to both GPSO and LPSO. In this paper, the orthogonal learning particle swarm optimization (OLPSO) [12] is used only in child swarms to process precise exploiting in its own promising area. The OLPSO algorithm is as easy to understand as the traditional PSO and retains the simplicity of PSO.

The framework of OLPSO is given in Algorithm 1.

IV. THE MULTI-SWARM PARTICLE WARM OPTIMIZATION WITH ORTHOGONAL LEARNING

A. Framework of the OLMPSO for DOPs

To address the essential requirements for dynamic environment, multi-swarm methods and orthogonal learning (OL) strategy are introduced in the proposed multi-swarm algorithm, i.e. OLMPSO, which consists of a parent swarm and some child swarms. The parent swarm is responsible for finding promising area in the search space while a child swarm updated with orthogonal learning particle swarm optimization (OLPSO) is created to exploit the new found promising area. The framework of OLMPSO is given in Algorithm 2.

Algorithm 1. OLPSO

1: Generate the initial swarm by randomly generating the position $X_i$and velocity V_i for particle i. Then evaluate the fitness of each particle;
2:Calculate P_i and P_n; Set gen=0, ω=0.9,c=2.0;
3: For each particle, construct the learning exemplar P_0 through P_i and P_n
4:**while** gen < GENERATION, i.e. stop criteria is not satisfied **do**
5: update ω= 0.9-0.5×gen/GENERATION
6: **for** each particle **i** in the swarm **do**
7: update particle **i** according to eq. (6) and eq. (2).
8: evaluate the fitness of particle i
9: **if** $f(X_i) < f(P_i)$
10: $P_i = X_i$; $stagnate_i$=0
11: **if** $f(P_i) < f(P_n)$
12: $P_n = P_i$
13: **else continue**
14: **end-if**
15: **else**
16: $stagnate_i$=$stagnate_i$+1
17: **if** $stagnate_i$> G
18: construct the learning exemplar P_0 through P_i and P_n; $stagnate_i$=0
19: **else continue**
20: **end-if**
21: **end-if**
22: **end-for**
23:**end-while**

Algorithm 2. OLMPSO

1: Initialization the parent swarm
2: **while** stop criteria is not satisfied **do**
3: **for** each particle **i** in the parent swarm **do**
4: update particle **i** according to eq. (1) and eq. (2).
5: update $pbest_i$
6: **for** each swarm c **do**
7: **if** distance(p_i, $cbest_c$) < r **then**
8: **if** f (p_i) > f ($cbest_c$) **then**
9: $cbest_c = p_i$
10: **end-if**
11: reinitialize particle i
12: **end-if**
13: **end-for**
14: **end-for**
15: update $cbest_{parent}$ (the gbest in the parent swarm)
16: **if** $cbest_{parent}$is updated **then**
17: $cbest_v = cbest_{parent}$
18: **for** each particle **i** in parent swarm **do**
19: **if** distance(p_i, $cbest_c$) < **r**
20: **if** |**v** | < π **then**
21: move particle **i** to the child swarm **v**
22: **end-if**
23: initialize particle **i**
24: **end-if**
25: **end-for**
26: **while** |**v**| < **π**
27: Create a new particle in the child swarm **v** within a radius **r/3** centered at $\mathbf{cbest}_c$
28: **end-while**
29: **end-if**
30: **for** child swarm **c** **do**
31: **for** each particle **i** in the child swarm **c do**
32: update particle **i** according to eq. (3) and eq. (2)
33: update $pbest_i$
34: **end-for**
35: Set $\mathbf{cbest}_c$ to the best position found by the particles in child swarm **c**
36: **end-for**
37: **for** each pair of child swarms **(k , l)**, k≠l **do**
38: **if** distance($cbest_k$, $cbest_l$) < r_{excl} **then**
39: destroy the child swarm whose **cbest** has a less fitness value.

```
40:     end-if
41:    end-for
42:   if a change is detected in the environment then
43:    for each particle i in the parent swarm do
44:     pbest_i = p_i
45:    end-for
46:    update cbest_parent
47:    for each child swarm c do
48:     for each particle i in the child swarm c do
49:      p_i = a random position in a hyper-sphere with a radius r_s
         centered at cbest_C
50:      pbest_i = p_i
51:     end-for
52:     Set cbest_C to the best position found by the particles in
         child swarm c
53:    end-for
54:   end-if
```

B. Updating the Particle of Parent Swarm

There is only one parent swarm in OLMPSO. After initializing this parent swarm, the particles in the parent swarm begin searching in the search space. At each iteration, the velocity (v_i) and position (p_i) of a particle *i* in the parent swarm is updated according to (1) and (2) with its best personal position ($pbest_i$) and the best position found by the parent swarm ($cbest_{parent}$), respectively. 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.

C. Creating New Child Swarm

When all particles in the parent swarm are updated, the distance between particle i and the best position found by a child swarm c ($cbest_c$), i.e. the attractor of each child swarm c, is calculated. If the distance between particle i and $cbest_c$ is less than *r*, $cbest_c$ will be updated to the position of particle i, and particle i will be reinitialized. Afterwards, the best position found in the parent swarm ($cbest_{parent}$) will be updated. If $cbest_{parent}$ is improved, a new child swarm will be created with $cbest_{parent}$ as its attractor. At the same time, the particles whose distances to the attractor of the newly created child swarm are less than r will be moved to the newly created swarm and a corresponding quantity of new particles will be created and initialized in the parent swarm. If the number of particles moved to the newly created child swarm (m) is less than the number of particles required for a child swarm (π), π-m particles will be created and initialized in a hyper-sphere within a radius r/3 centered at the $cbest_c$ in the child swarm. After this, the velocities and positions of all particles in every child swarm will be updated with the OLPSO optimization according to (3) and (2), respectively. Then, the personal best position (*pbest*) for all child particles and the child swarms' best position (*cbest*) will be updated.

D. Overlapping Check Scheme

There is an overlapping check scheme in the next procedure. At the end of each iteration, the distance between every two child swarm is computed to check whether they are searching in the same area or not, for the reason that searching a local area with more than one child swarm is not very useful and the limited computation resources may be wasted. If the Euclidian distance between their attractors is less than a specified threshold r_{excl}, two child swarms are searching in the same area. Once this happen, the worse child swarm whose attractor have a worse fitness than the other, will be destroyed.

E. Coping Mechanism for Environment Change

At the end of the proposed algorithm, there is a coping mechanism to be triggered when an environment change is detected. Once environment changes, particles in the parent swarm will re-evaluate their fitness with their positions current and reset their best personal positions to these positions. Different from the parent swarm, the particles in the child swarms will set their new positions to a random position in a hyper-sphere with a radius r_s centered at their swarm's attractor. Then they will reset their best personal positions to their new positions and update their child swarms attractors.

V. Experiments and Results

A. Experimental Setup

1) Moving Peaks Benchmark (MPB) Problem:

The MPB problem proposed by Branke [16] has been widely used as dynamic benchmark problems in the literature to evaluate the performance of optimization algorithms in dynamic environments. Within the MPB problem, the optima can be varied with time by three features, the location (X (t)), height (H (t)), and width of peaks (W(t)). For the D dimensional landscape, the problem is defined as follows:

$$F(\vec{x}, t) = \max_{i=1,\dots p} \frac{H_i(t)}{1+W_i(t)\sum_{j=1}^{D}(x_i(t)-X_{ij}(t))^2} \quad (8)$$

where $H_i(t)$, $W_i(t)$ and $X_{ij}(t)$ are the height, the width and the *j*th dimension of the location of peak i at time t, respectively. There are p independently specified peaks in total which are blended together by the "max" function. The location of each peak is varied in a direction by a random vector $\vec{v_i}(t)$ of a distance s (the shift length), and the random vector $\vec{v_i}(t)$ is defined as follows:

$$\vec{v_i}(t) = \frac{s}{|\vec{r}+\vec{v_i}(t-1)|}((1-\lambda)\vec{r} + \lambda\vec{v_i}(t-1)) \quad (9)$$

where the shift vector $\vec{v_i}(t)$ is a linear combination of a random vector $\vec{r}$ and the previous shift vector $\vec{v_i}(t)$ and is normalized to the shift length s. The correlated parameter λ is set to 0, which implies that the peak movements are uncorrelated.

More formally, a change of a single peak can be described as follows:

$$H_i(t) = H_i(t-1) + \text{height\_severity} * \sigma \quad (10)$$

$$W_i(t) = W_i(t-1) + \text{width\_severity} * \sigma \quad (11)$$

$$\vec{X_i}(t) = \vec{X_i}(t-1) + \vec{v_i}(t) \quad (12)$$

2) Experimental Settings:

In this paper, unless stated otherwise, the default settings and definition of the benchmark used can be found in Table II, which are the same as in all the involved algorithms. In Table II, the term "change frequency (*U*)" means that environment changes every *U* fitness evaluations, S denotes the variable moves in the range of [0,100], and *I* denotes the initial height of every peak. The height of peaks is shifted randomly in the range *H*= [30, 70] and the width of peaks is shifted randomly in the range *W*= [1, 12].

The performance measure used is the offline error, which is defined as follows:

$$\mu = \frac{1}{K}\sum_{k=1}^{K}(h_k - f_k) \tag{13}$$

where f_k is the best solution obtained by an algorithm just before the *k*th environmental change, h_k is the optimum value of the *k*th environment, μ is the average of all differences between h_k and f_k over the environmental changes, and *K* is the total number of environments. For each run, there were *K*= 50 environments, which result in 5×10^5 fitness evaluations. All the results reported are based on the average over 100 independent runs.

TABLE II
Default Settings for the MPB Problem

Parameter	Value
Number of peaks, *p*	10
Change frequency, *U*	5000
Height severity	7.0
Width severity	1.0
Peak shape	Cone
Basic function	No
Shift length, *s*	1.0
Number of dimensions, *D*	5
Correlation coefficient, *λ*	0
S	[0, 100]
H	[30.0, 70.0]
W	[1, 12]
I	50.0

B. Different Algorithm Settings

For OLMPSO, the acceleration coefficients c1 and c2 are set to 1.496180 and the inertial weight ω is set to 0.729844 [17]. The number of particles in the parent swarm and the child swarms (π) are set to 2 and 7 particles, respectively. The radius of the child swarms (r), the minimum allowed distance between two child swarm (r_{excl}) and the radius of quantum particles (r_s) are set to 30.0, 30.0, and 0.5, respectively. The proposed algorithm is compared with MPSO [18], mQSO [7], FMSO [11], and cellular PSO [19, 20]. For MPSO, the configurations are almost the same as OLMPSO, except for the number of particles in the parent swarm and the child swarms, which are set to 5 and 10 particles [18]. For mQSO, we adapt a configuration $10(5+5^q)$ which creates 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 [7, 21]. 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 [11]. 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) is set to 0.5 for all experiments. The cell capacity θ is set to 10 particles for every cell [19, 20].

C. Comparison of OLMPSO With Peer Algorithms

In this experiment, we compare the performance of OLMPSO with MPSO, mQSO10, FMSO, and Cellular PSO on the MPB problems with different number of peaks and different value of change frequency. The average offline error and 95% confidence interval are taken for 100 runs. Offline error of the algorithm above for different dynamic environment is presented in table III and table VI. For each environment, result of the best performing algorithm(s) with 95% confidence is printed in bold.

TABLE III
OFFLINE ERROR ± STANDARD ERROR FOR f=5000

m	OLMPSO	MPSO	mQSO10	FMSO	Cellular PSO
1	**1.9e-6**±0.8e-6	0.56±0.04	3.82±0.35	3.44±0.11	2.55±0.12
5	**0.07**±0.11	1.06±0.06	1.90±0.08	2.94±0.07	1.68±0.11
10	**0.99**±0.43	1.51±0.04	1.91±0.08	3.11±0.06	1.78±0.05
20	**1.67**±0.40	1.89±0.04	2.56±0.10	3.36±0.06	2.61±0.07
30	**1.83**±0.51	2.03±0.06	2.68±0.10	3.28±0.05	2.93±0.08
40	**1.94**±0.55	2.04±0.06	2.65±0.08	3.26±0.04	3.14±0.08
50	2.26±0.54	**2.08**±0.02	2.63±0.08	3.22±0.05	3.26±0.08

TABLE IV
OFFLINE ERROR ± STANDARD ERROR FOR f=10000

m	OLMPSO	MPSO	mQSO10	FMSO	Cellular PSO
1	**9.3e-12**±5e-11	0.27±0.02	1.90±0.18	1.90±0.06	1.53±0.12
5	**0.02**±0.001	0.70±0.10	1.03±0.06	1.75±0.06	0.92±0.10
10	**0.48**±0.20	0.97±0.04	1.10±0.07	1.91±0.04	1.19±0.07
20	**0.43**±0.22	1.34±0.08	1.84±0.08	2.16±0.04	2.20±0.10
30	**0.64**±0.38	1.43±0.05	2.00±0.09	2.18±0.04	2.60±0.13
40	**0.65**±0.30	1.47±0.06	1.99±0.07	2.21±0.03	2.73±0.11
50	**0.64**±0.28	1.47±0.04	1.99±0.07	2.60±0.08	2.84±0.12

As depicted in table III, i.e. the change frequency f = 5000, the result of OLMPSO is slightly worse than the result of MPSO when the number of peaks exceeds 40, but they are also much better than the other three algorithms results. In addition, if the value of the change frequency increase, OLMPSO can achieve much better results, which can be seen in table IV, the proposed algorithm (OLMPSO) outperforms all the other tested PSO algorithms. Such an outstanding performance this algorithm to have is due to the application of OLPSO in the child swarms. And this is also the main difference between OLMPSO and the next best algorithm, i.e. MPSO, because MPSO only applies the standard PSO. With OLPSO, a particle can fly more

promisingly toward the global optimum because the OL strategy could construct a guidance exemplar with an ability to predict promising search directions toward the global optimum. So OLPSO can quickly find better solutions than other algorithms after a change occurs in the environment. Furthermore, in OLMPSO the number of child swarms converges to the number of peaks in the environment, which is the same as MPSO. This will help OLMPSO to track the changes more effectively.

D. Effect of Varying the Size of the Parent Swarm

This set of experiments investigates the effect of the size of the parent swarm, i.e. the number of particles in the parent swarm, on the performance of OLMPSO. The number of particles in the child swarms (π) is set to 7 particles, and the number of particles in the parent swarm is varied as the X coordinate axis in Fig. 1 and Fig. 2. As the results depicted in Fig. 1 and Fig. 1, when there are few

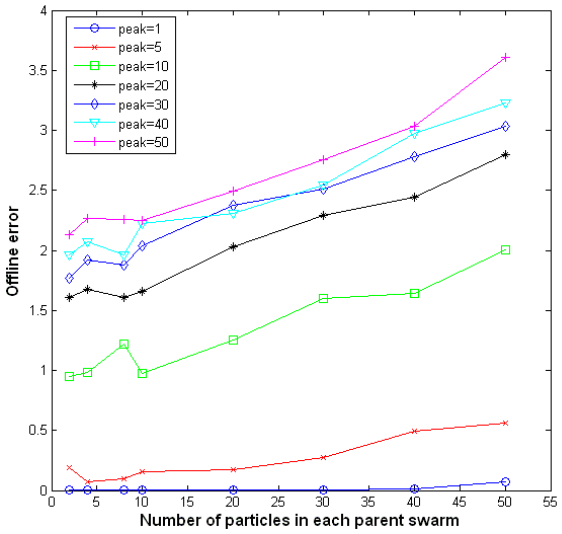

Fig.1. The effect of the size of the parent swarm on the offline error

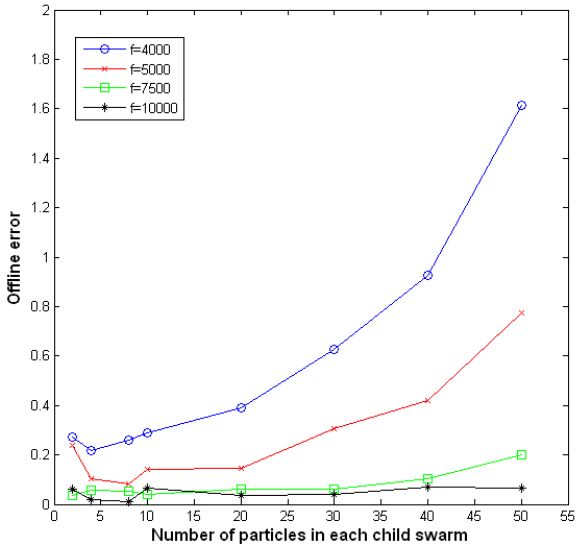

Fig.2. The effect of the size of the parent swarm on the offline error

peaks in the environment (peak=1 or peak=5) or when the environment changes slowly (f=7500 or f=10000) the size of the parent swarm does not affect offline error significantly. But for other environments, i.e. the more peaks exist in the environment or the more frequently the changes occur, offline error escalates and be affected significantly by increasing the size of the parent swarm. It also can be seen in Fig.1 that the offline error slightly increases when the size of the parent swarm increases from 2 to 4 and then decreases from 4 to 7. So we set the size of the parent swarm to 2 in the former experiment to compare OLMPSO with peer algorithms on the condition that the number of peaks varying.

E. Effect of Varying the Size of the Child Swarms

This set of experiments investigates the effect of the size of the child swarms (π), i.e. the number of particles in the child swarm, on the performance of CPSO. The number of particles in the parent swarms is set to 2 particles, and the number of particles in the child swarm is varied as the X coordinate axis in Fig. 3 and Fig. 4. As depicted in Fig. 3 and Fig. 4, when there are few peaks in the environment (peak=1 or peak=5) or when the environment changes slowly (f=7500 or f=10000) the size of the child swarm does not affect offline error significantly, which have the same tendency as the former experiment shown in Fig. 1 and Fig.2. Furthermore, when the number of particles in the child swarms is 7, the offline error is the least in all different environments, i.e. the optimal value for the size of the child swarms is 7. The reason is that too many particles in the child swarms does not help the particles to find better solutions, but consumes precious function evaluations at the same time. Conversely, when there are too few particles in the child swarms, there is no enough resource to quickly find the peaks, so the offline error increases.

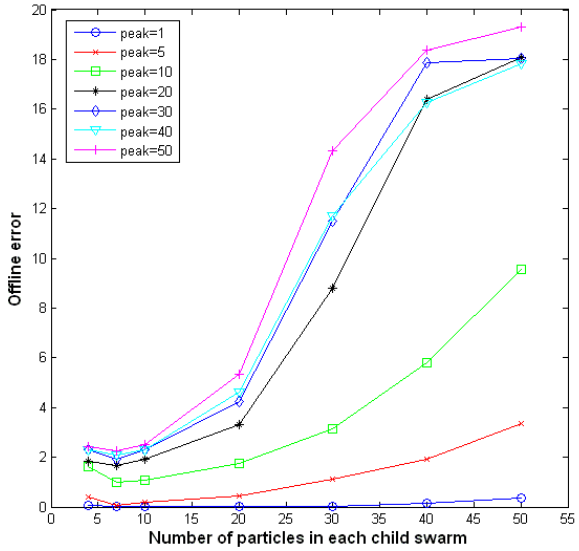

Fig. 3. The effect of the number of particles in each child swarm on the offline error

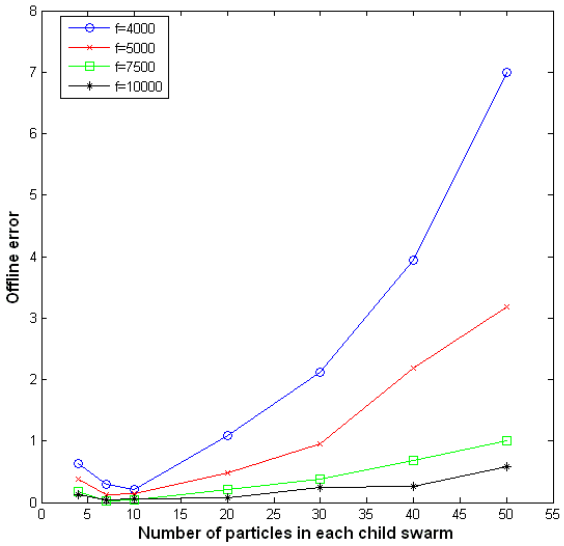

Fig. 4. The effect of the number of particles in each child swarm on the offline error

VI. CONCLUSIONS

In this paper, we proposed a new multi-swarm PSO algorithm for dynamic environments. This algorithm (OLMPSO) comprises three main parts, which are a parent swarm to explore the search space, some child swarms to exploit promising areas found by the parent swarm and an orthogonal learning (OL) strategy to utilize previous search information (experience) more efficiently in these child swarms to improve the convergence speed. The worse child swarms will be removed to improve the search performance, when the search areas of two child swarms overlap. Moreover, in order to quickly track the changes in the environment, all particles in a child swarm perform a random local search around the best position found by the child swarm after a change in the environment is detected. The experimental results show that the efficiency of OLMPSO for locating and tracking multiple optima in dynamic environments is outstanding in comparison with other particle swarm optimization models, including MPSO, a previously presented multi-swarm algorithm with the similar approach.

As the results show in table 2 and 3, the offline error of OLMPSO are the least in comparison with other particle swarm optimization models. But we cannot get good standard error as offline error, the big values of standard error show that the performance of OLMPSO is not as stable as other wick algorithm. So this is a shortcoming of this algorithm, and we will concentrate on solving it in the future work.

REFERENCES

[1] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," *in Proc. IEEE Int. Conf. Neural Netw*., vol. 4. 1995, pp. 1942–1948.

[2] P. Riccardo, J. Kennedy, and T Blackwell, "Particle swarm optimization." *Swarm intelligence 1.1* (2007): 33-57.

[3] Y. H. Shi, R. C. Eberhart, "A modified particle swarm optimizer." *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on.* IEEE, 1998.

[4] P.C. Cosman., R. M. Gray, and M. Vetterli, "Vector quantization of image subbands: A survey." *Image Processing, IEEE Transactions on* 5.2 (1996): 202-225.

[5] 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 system,"IEEE Trans. Evol. Comput., vol. 12,no. 2, pp. 171–195, Apr. 2008

[6] A. B. Hashemi, R. M. Mohammad, "A note on the learning automata based algorithms for adaptive parameter selection in PSO." *Applied Soft Computing* 11.1 (2011): 689-705.

[7] T. Blackwell, and J. Branke, "Multiswarms, exclusion, and anti-convergence in dynamic environments." *Evolutionary Computation, IEEE Transactions on* 10.4 (2006): 459-472.

[8] J. Kennedy, "Stereotyping: improving particle swarm performance with cluster analysis Stereotyping: improving particle swarm performance with cluster analysis." *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*. Vol. 2. 2000.

[9] C. H. Li, S. X. Yang, "A clustering particle swarm optimizer for dynamic optimization." *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*. IEEE, 2009.

[10] R. C. Eberhart, R. Dobbins, P. Simpson: Evolutionary Computation Implementations. Computational Intelligence PC Tools, pp. 212–226. Morgan Kaufmann, San Francisco (1996)

[11] C. H. Li, S. X. Yang, "Fast multi-swarm optimization for dynamic optimization problems." *Natural Computation, 2008. ICNC'08. Fourth International Conference on*. Vol. 7. IEEE, 2008.

[12] Z. H. Zhan, J. Zhang, Y Li, Y. H. Shi, "Orthogonal learning particle swarm optimization." *Evolutionary Computation, IEEE Transactions on* 15.6 (2011): 832-847.

[13] Y. K. Kim and J. B. Ra, "Adaptive learning method in self-organizing map for edge preserving vector quantization," *IEEE Trans. Neural Networks*, 1995, 6:278-280.

[14] D. C. Montgomery, Design and Analysis of Experiments, 5th ed. New York: Wiley, 2000.

[15] Math. Stat. Res. Group, Chinese Acad. Sci., Orthogonal Design (in Chinese). Beijing, China: People Education Pub., 1975.

[16] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," *in Proc. Congr. Evol. Comput*., vol. 3. 1999, pp. 1875–1882.

[17] F. Van Den Bergh, "An analysis of particle swarm optimizers", *University of Pretoria*, 2006.

[18] M. Kamosi, A. B. Hashemi, and M. R. Meybodi, "A new particle swarm optimization algorithm for dynamic environments." *Swarm, evolutionary, and memetic computing*. Springer Berlin Heidelberg, 2010. 129-138.

[19] A. B. Hashemi, M. R. Meybodi, "Cellular PSO: A PSO for dynamic environments." *Advances in computation and intelligence*. Springer Berlin Heidelberg, 2009. 422-433.

[20] A. B. Hashemi, M. R. Meybodi, "A multi-role cellular PSO for dynamic environments." *Computer Conference, 2009. CSICC 2009. 14th International CSI*. IEEE, 2009.

[21] T. Blackwell, J. Branke, and X. D. Li, "Particle swarms for dynamic optimization problems." *Swarm Intelligence*. Springer Berlin Heidelberg, 2008. 193-217.

[22] A. Carlisle, G. Dozier, "Adapting particle swarm optimization to dynamic environments." *Proceedings of the International Conference on Artificial Intelligence*. Vol. 1. Athens, GA, USA: CSPIEA Press, 2000.

[23] S. Janson, and M. Middendorf, "A hierarchical particle swarm optimizer for dynamic optimization problems." *Applications of evolutionary computing*. Springer Berlin Heidelberg, 2004. 513-524.

[24] S. Janson, and M. Middendorf, "A hierarchical particle swarm optimizer and its adaptive variant." *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* 35.6 (2005): 1272-1282.

[25] R. C. Eberhart, Y. H. Shi. "Tracking and optimizing dynamic systems with particle swarms." *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*. Vol. 1. IEEE, 2001.