A Clustering Particle Swarm Optimizer for Locating and Tracking Multiple Optima in Dynamic Environments

Shengxiang Yang, *Member, IEEE,* and Changhe Li

***Abstract*—In the real world, many optimization problems are dynamic. This requires an optimization algorithm to not only find the global optimal solution under a specific environment but also to track the trajectory of the changing optima over dynamic environments. To address this requirement, this paper investigates a clustering particle swarm optimizer (PSO) for dynamic optimization problems. This algorithm employs a hierarchical clustering method to locate and track multiple peaks. A fast local search method is also introduced to search optimal solutions in a promising subregion found by the clustering method. Experimental study is conducted based on the moving peaks benchmark to test the performance of the clustering PSO in comparison with several state-of-the-art algorithms from the literature. The experimental results show the efficiency of the clustering PSO for locating and tracking multiple optima in dynamic environments in comparison with other particle swarm optimization models based on the multiswarm method.**

***Index Terms*—Clustering, dynamic optimization problem (DOP), local search, multiswarm, particle swarm optimization.**

I. Introduction

GENERALLY speaking, most research on evolutionary algorithms (EAs) focuses on static optimization problems. However, many real-world problems are dynamic optimization problems (DOPs), where changes occur over time. This requires optimization algorithms to not only find the global optimal solution under a specific environment but also to continuously track the changing optima over different dynamic environments. Hence, optimization methods that are capable of continuously adapting to a changing environment are needed.

In recent years, investigating EAs for DOPs has attracted a growing interest because EAs are intrinsically inspired from natural or biological evolution, which is always subject to an ever-changing environment, and hence EAs, with proper enhancements, have a potential to be good optimizers for DOPs. Over the years, several approaches have been developed into traditional EAs to address DOPs [8], [18], [44], including diversity schemes [13], [15], [43], memory schemes [6], [42], [48], multipopulation schemes [7], [47], adaptive schemes [29], [45], [46], multiobjective optimization methods [11], and problem change detecting approaches [35].

Particle swarm optimizer (PSO) is a versatile population-based stochastic optimization technique. Similar to other EAs in many respects, PSO has been shown to perform well for many static problems [33]. However, it is difficult for the basic PSO to optimize DOPs. The difficulty lies in two aspects: 1) outdated memory due to the changing environment, and 2) diversity loss due to convergence. Of these two aspects, the diversity loss is by far more serious [5]. 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 [3].

In the basic PSO, the diversity loss is mainly due to the strong attraction of the global best particle, which results in that all the particles quickly converge on local or global optimum where the global best particle locates. This feature is beneficial for many stationary optimization problems. However, for DOPs, this feature is not good for PSO to track the changing optima. For DOPs, it is important to guide particles searching in different promising regions to obtain promising local optima as many as possible because these promising local optima may become the global best in the next new environment. Hence, local best particles are needed to guide the search in local regions in the search space. However, the question becomes how to determine which particles should be suitable as the neighborhood best and how to assign particles in different neighborhoods to move toward different subregions.

Several PSO algorithms have been recently proposed to address DOPs [5], [16], [17], [31], [41], of which using multiswarms seems a good technique. The multiswarm method can be used to enhance the diversity of the swarm, with the aim of maintaining multiple swarms on different peaks. The traditional method of using the multiswarm method to find optima for multimodal functions divides the whole search space into local subspaces, each of which might cover one or a small number of local optima, and then separately searches within these subspaces. Here, there are several key, usually difficult, issues to be addressed, e.g., how to guide particles to move toward different promising subregions, how to define

Manuscript received April 30, 2009; revised September 9, 2009, December 4, 2009, and February 3, 2010. Date of publication August 26, 2010; date of current version November 30, 2010. This work was supported by the Engineering and Physical Sciences Research Council of U.K., under Grant EP/E060722/1.

S. Yang is with the Department of Information Systems and Computing, Brunel University, Middlesex UB8 3PH, U.K. (e-mail: shengxiang.yang@brunel.ac.uk).

C. Li is with the Department of Computer Science, University of Leicester, University Road, Leicester LE1 7RH, U.K. (e-mail: cl160@mcs.le.ac.uk).

Digital Object Identifier 10.1109/TEVC.2010.2046667

the area of each subregion, how to determine the number of subswarms needed, and how to generate subswarms. These key issues will be further discussed later on in Section III.

In order to address the key issues relevant to the multiswarm method, a clustering PSO (CPSO) has recently been proposed for DOPs in [26]. CPSO [26] employs a nearest neighbor learning strategy to train particles and a hierarchical clustering method to locate and track multiple optima. This paper further investigates the performance of CPSO in dynamic environments. There are some simplifications compared with the original CPSO in this paper. First, the training process is removed in this paper. Second, in the original CPSO [26], the hierarchical clustering method involves two phases of clustering: 1) rough clustering, and 2) refining clustering. In this paper, the hierarchical clustering method is simplified into only one phase. The reason of these simplifications will be explained in detail later in this paper.

In [26], CPSO was compared with the basic PSO and a simple genetic algorithm on the generalized dynamic benchmark generator proposed in [25], [27]. This paper further extends the experimental study based on the moving peaks benchmark (MPB) problem [6] and compares the performance of CPSO with several state-of-the-art PSO algorithms that were developed for DOPs in the literature, including two PSO algorithms proposed by Blackwell and Branke [5], the collaborative model introduced by Lung and Dumitrescu [28], the speciation PSO proposed by Parrott and Li [31] as well as an improved version of SPSO with regression (rSPSO) [2]. Based on the experimental results, an algorithm performance analysis regarding the weakness and strength of investigated algorithms is carried out. This paper also carries out experiments on the sensitivity analysis with respect to several key parameters, such as the population size of the initial swarm and the maximum size of each subswarm, on the performance of CPSO for DOPs.

The rest of this paper is organized as follows. Section II introduces the basic PSO algorithm and some multiswarm PSO algorithms for locating multiple optima in both static and dynamic environments. Section III presents the general considerations of key issues relevant to the multiswarm method for PSO in dynamic environments. The investigated CPSO is described in detail in Section IV. Section V presents the experimental study and discussions based on the experimental results. Finally, conclusions and discussions on relevant future work are given in Section VI.

II. Related Work

A. Particle Swarm Optimization

Particle swarm optimization was first introduced by Kennedy and Eberhart [14], [19]. It is motivated from the social behavior of organisms, such as bird flocking and fish schooling. In PSO, a swarm of particles "fly" through the search space. Each particle follows the previous best position found by its neighbor particles and the previous best position found by itself. In the past decade, PSO has been actively studied and applied for many academic and real world problems with promising results due to its property of fast convergence [34].

Algorithm 1 Basic PSO

```
Generate the initial swarm by randomly generating the position and velocity for each particle;
Evaluate the fitness of each particle;
repeat
  for Each particle i do
    Update particle i according to (1) and (2);
    if f(x_i) < f(x_pbest_i) then
      x_pbest_i := x_i;
      if f(x_i) < f(x_gbest) then
        x_gbest := x_i;
      end if
    end if
  end for
until The stop criterion is satisfied
```

Ever since PSO was first introduced, several major versions of the PSO algorithm have been developed [34]. Each particle i is represented by a position vector $\vec{x}_i$ and a velocity vector $\vec{v}_i$, which are updated in the version of PSO with an inertia weight [37] as follows:

$$v'^{d}_{i} = \omega v^{d}_{i} + \eta_1 r_1 (x^{d}_{pbest_i} - x^{d}_{i}) + \eta_2 r_2 (x^{d}_{gbest} - x^{d}_{i}) \quad (1)$$

$$x'^{d}_{i} = x^{d}_{i} + v'^{d}_{i} \quad (2)$$

where x'^{d}_{i} and x^{d}_{i} represent the current and previous position in the dth dimension of particle i respectively, v'_i and v_i are the current and previous velocity of particle i respectively, $\vec{x}_{pbest_i}$ and $\vec{x}_{gbest}$ are the best position found by particle i so far and the best position found by the whole swarm so far respectively, $\omega \in (0, 1)$ is an inertia weight, which determines how much the previous velocity is preserved, η_1 and η_2 are the acceleration constants, and r_1 and r_2 are random numbers generated in the interval [0.0, 1.0] uniformly. The framework of the basic PSO algorithm is shown in Algorithm 1.

According to the theoretical analysis by Clerc and Kennedy [12], the trajectory of a particle $\vec{x}_i$ converges to a weighted mean of $\vec{x}_{pbest_i}$ and $\vec{x}_{gbest}$. Whenever the particle converges, it will "fly" to the individual best position and the global best position. According to the update equation, the individual best position of a particle will gradually move closer to the global best position. Therefore, all the particles will converge onto the global best particle's position.

There are two main models of the PSO algorithm, called *gbest* (global best) and *lbest* (local best), respectively. The two models differ in the way of defining the neighborhood for each particle. In the *gbest* model, the neighborhood of a particle consists of the particles in the whole swarm, which share information between each other. On the contrary, in the *lbest* model, the neighborhood of a particle is defined by several fixed particles. The two models give different optimization performances on different problems. Kennedy and Eberhart [22] and Poli *et al.* [34] pointed out that the *gbest* model has a faster convergence speed with a higher chance of getting stuck

in local optima than the *lbest* model. On the contrary, the *lbest* model is less vulnerable to the attraction of local optima but with a slower convergence speed than the *gbest* model.

B. Multiple Swarms

Many researchers have considered multipopulations as a means of enhancing the diversity of EAs to address DOPs. Kennedy [21] proposed a PSO algorithm that uses a k-means clustering algorithm to identify the centers of different clusters of particles in the population, and then uses these cluster centers to substitute the personal best or neighborhood best positions. In order to allow cluster centers to be stabilized, the k-mean algorithm iterates three times. The limitation of this clustering approach lies in that the number of clusters must be predefined.

Branke *et al.* proposed a self-organizing scouts (SOS) [7] algorithm that has been shown to give promising results on DOPs with many peaks. In SOS, the population is composed of a parent population that searches through the entire search space and child populations that track local optima. The parent population is regularly analyzed to check the condition for creating child populations, which are split off from the parent population. Although the total number of individuals is constant since no new individuals are introduced, the size of each child population is adjusted regularly.

Brits *et al.* [9] proposed an *nbest PSO* algorithm which is in particular designed for locating multiple solutions to a system of equations. The *nbest* PSO algorithm defines the "neighborhood" of a particle as the closest particles in the population. The neighborhood best for each particle is defined as the average of the positions of these closest particles. In [10], a niching PSO (NichePSO) was proposed by incorporating a cognitive only PSO model and the guaranteed convergence PSO algorithm [39]. NichePSO maintains a main swarm that can create a subswarm once a niche is identified. The main swarm is trained by the cognition only model [20]. If a particle's fitness shows a little change over a small number generations, then a new subswarm is created with the particle and its closest neighbors. NichePSO uses some rules to decide the absorption of particles into a subswarm and the merging operation between two subswarms, which mainly depends on the radius of the involved subswarms.

Parrott and Li developed a speciation-based PSO (SPSO) [30], [23] which dynamically adjusts the number and size of swarms by constructing an ordered list of particles, ranked according to their fitness, with spatially close particles joining a particular species. At each generation, SPSO aims to identify multiple species seeds within a swarm. Once a species seed has been identified, all the particles within its radius are assigned to that same species. Parrott and Li also proposed an improved version with a mechanism to remove redundant duplicate particles in species in [31]. In [1], Bird and Li developed an adaptive niching PSO (ANPSO) algorithm which adaptively determines the radius of a species by using the population statistics. Recently, Bird and Li introduced another improved version of SPSO using a least squares regression (rSPSO) in [2].

The atomic swarm approach has been adapted to track multiple optima simultaneously with multiple swarms in dynamic environments by Blackwell and Branke [4], [5]. In their approach, a charged swarm is used for maintaining the diversity of the swarm, and an exclusion principle ensures that no more than one swarm surround a single peak. In [5], anti-convergence is introduced to detect new peaks by sharing information among all subswarms. This strategy was experimentally shown to be efficient for the MPB function [6].

To specify the number of clusters within the k-mean PSO algorithm, Passaro and Starita [32] used the optimization of a criterion function in a probabilistic mixture-model framework. In this framework, the particles are assumed to be generated by a mix of several probabilistic distributions. Each different cluster corresponds to a different distribution. Then, finding the optimum number k is equivalent to fitting the model with the observed data while optimizing some criterion. The performance of their algorithm was reported better than SPSO [23] and ANPSO [1] for static problems.

A collaborative evolutionary swarm optimization (CESO) was proposed in [28]. In CESO, two swarms, which use the crowding differential evolution (CDE) [38] and PSO model respectively, cooperate with each other by a collaborative mechanism. The swarm using CDE is responsible for preserving diversity while the PSO swarm is used for tracking the global optimum. The competitive results were reported in [28].

Inspired by the SOS algorithm [7], a fast multiswarm optimization (FMSO) algorithm was proposed in [24] to locate and track multiple optima in dynamic environments. In FMSO, a parent swarm is used as a basic swarm to detect the most promising area when the environment changes, and a group of child swarms are used to search the local optimum in their own subspaces. Each child swarm has a search radius, and there is no overlap among all child swarms by excluding them from each other. If the distance between two child swarms is less then their radius, then the whole swarm of the worse one is removed. This guarantees that no more than one child swarm covers a single peak.

A CPSO has recently been proposed for DOPs in [26]. In CPSO, each particle learns from its own historical best position and the historical best position of its nearest neighbor other than the global best position as in the basic PSO algorithm. The velocity update equation for training a particle i is as follows:

$$v_i'^d = \omega v_i^d + \eta_1 r_i^d (x^d_{pbest_i} - x_i^d) + \eta_2 \cdot r_i^d \cdot (x^d_{pbest_{i\_n}} - x_i^d) \quad (3)$$

where $\vec{x}_{pbest_{i\_n}}$ is the personal best position of the particle that is nearest to particle i. The position of particle i is updated the same way as shown in (2). This learning strategy enables particles in CPSO adaptively detect subregions by themselves and assign them to different neighborhoods. Using a hierarchical clustering method, the whole swarm in CPSO can be divided into subswarms that cover different local regions. In order to accelerate the local search, a learning strategy for the global best particle was also introduced in CPSO. CPSO has shown some promising results according to the preliminary experimental study in [26].

III. General Considerations for Multiswarms

In order to address the convergence problem of PSO for DOPs, the multiswarm method can be used to maintain multiple swarms on different peaks, which are referred to as the optima in this paper. For example, for the MPB problem [6], the highest peak in the fitness landscape is the global optimum and the other peaks with a lower height are local optima. Hence, for the multiswarm method to work, the whole search space can be divided into several subregions. Each subregion might contain one or more than one peak. Each subswarm covers one subregion and exploits it. As mentioned above, when applying the multiswarm method to achieve this purpose, there are several key issues to be considered.

The first issue concerns how to guide particles to move toward different promising subregions. This issue is important since if particles cannot move to different subregions, PSO cannot locate and track multiple optima. This requires that an algorithm should have a good global search capability to explore promising subregions. In [10], the cognitive only PSO model, which was tested by Kennedy [20], was used to train particles. Since there is no information sharing among particles in the cognitive only model, each particle just blindly searches around its personal best position. This may cause the stagnation problem if there are deceptive subregions in the search space. Hence, in order to guide particles toward different promising subregions, particles should cooperate with the other nearby particles.

The second issue concerns how to define the area for each subregion in the search space. The area of a subregion determines how many peaks it may contain. If the area of a subregion is too small, there is a potential problem that small isolated subswarms may converge on local optima. In this case, the diversity will be lost and then the algorithm can hardly make any progress. However, if a subregion is too large, there may be more than one peaks within the subregion covered by a subswarm. The best situation is that each subregion just contains one peak. However, to achieve this goal is very hard due to the complexity of the search space, especially for real world problems. Traditionally, the search area of each subregion is predefined by users according to preliminary experiments [30] or a formulated estimation [5], and the search area for all subregions is the same. Obviously, it is not true that all the peaks have exactly the same shape or width in the whole search space. It is very hard to know the shape of a subregion. Hence, how to define the search area of a subregion is a very hard problem. Ideally, particles within the neighborhood of a peak should be able to calculate the subarea by themselves.

How many subswarms are needed is the third issue to consider. From the experimental results in [5], the optimal number of subswarms is equal to the total number of peaks in the whole search space. The more peaks in the search space, the more subswarms we probably need. If too many subswarms distribute in the fitness landscape, the limited computation resources may be wasted. On the contrary, if there are too small number of subswarms, the PSO algorithm cannot efficiently track different local optima. Again, the problem is that the number of peaks in the search space is usually unknown in advance, especially for real world problems. Although the number of peaks is given for some benchmark problems, we should assume that it is unknown to us.

Finally, how to generate subswarms is also an open issue. Generally, subswarms are simply obtained by separating the main swarm according to some mechanism. In [21], a k-mean clustering method was used to generate clusters. The limitation of the k-mean method is that the number of clusters must be predefined. In the speciation-based PSO [30], a new species is produced around a species seed. That is, all the particles within a radius r_s distance to a species seed are classified into a species corresponding to that species seed. Hence, a new species is created by a given radius r_s around its seed. The number of subswarms is simply predetermined in mCPSO [5], although exclusion and anti-convergence strategies were used. Exclusion prevents subswarms from covering a single peak by an exclusion radius r_{excl}, and anti-convergence allows new peaks to be detected, which was implemented by defining a convergence radius r_{conv}. However, the serious disadvantages of SPSO and mCPSO are those radius parameters must be given. In order to generate subswarms as accurate as possible, that is, only all particles on a same peak form a subswarm, the analysis of population distribution should be done before creating subswarms by some statistic methods.

If particles close to a peak can detect the peak by themselves, then they can classify themselves into a same cluster, and the search area can also be automatically defined when the new cluster is formed. This thinking motivated the proposal of CPSO in [26]. In the following section, CPSO in its simplified version proposed in this paper is described in detail to show how it overcomes the above problems when using the multiswarm method.

IV. Clustering Particle Swarm Optimization

A. Framework of the CPSO for DOPs

To address the above considerations for multiswarm methods, a clustering method is introduced in CPSO. The clustering method can enable CPSO to assign particles to different promising subregions, adaptively adjust the number of subswarms needed, and automatically calculate the search region for each subswarm.

CPSO starts from an initial swarm, named the cradle swarm. Then, subswarms are created by a hierarchical clustering method. When subswarms are created, local search is launched on them in order to exploit potential peaks covered by these subswarms respectively. Finally, overlapping, convergence, and overcrowding checks are performed on the subswarms before the next iteration starts. If an environmental change is detected, a new cradle swarm will be randomly re-generated with the reservation of the positions located by all survived subswarms in the previous environment.

The framework of CPSO for DOPs is given in Algorithm 2. In the following sections, the major components of CPSO, including the clustering method, local search operator, subswarm checks, and detecting environmental changes, are described in detail, respectively.

B. Single Linkage Hierarchical Clustering

Some researchers have used the k-mean clustering method to generate subswarms, the problem of the k-mean method is

Algorithm 2 CPSO Algorithm

```
Create an empty convergence list clst to record the best particles of converged subswarms;
Create an empty list slst to record subswarms;
Set the fitness evaluation counter evals := 0;
Generate an initial cradle swarm C randomly;
Clustering(C, slst);
while stop criteria is not satisfied do
    for each subswarm slst[i] do
        LocalSearch(slst[i], evals);
    end for
    CheckSubswarms(C, slst, clst);
    if |C| > 0 then
        LocalSearch(C, evals);
    end if
    if DetectChange(C, slst, clst, evals) then
        Clustering(C, slst);
    end if
end while
```

Algorithm 3 *Clustering*$(C, slst)$

```
Create a temporary cluster list G of size |C|;
for each particle i in C do
    G[i]:=C[i]; {i.e., each particle forms one cluster in G}
end for
Calculate the distance between all clusters (i.e., particles) in G and construct a distance matrix M of size |G| × |G|;
while TRUE do
    if FindNearestPair(G, r, s) = FALSE then
        Break;
    end if
    r := r + s; {i.e., merge clusters r and s into r}
    Delete the cluster s from G;
    Re-calculate all distances in M which have been affected by the merge of r and s;
    if each cluster in G has more than one particle then
        Break;
    end if
end while
slst := G;
Empty C;
```

that we do not know the optimum value of k for the current population. In addition, the optimum value of k is problem dependant. Setting k to a too large or a too small value will cause the problem of an improper number of subswarms, as discussed above. Traditionally, subswarms are created by directly using a number of swarms or simply splitting off from a main swarm. There is little or no analysis of distribution of individuals in the search space. Different from traditional clustering methods for multiswarm-based PSO algorithms, CPSO uses a single linkage hierarchical clustering method [36], as shown in Algorithm 3, to create subswarms.

In the clustering method, the distance $d(i, j)$ between two particles i and j in the D-dimensional space is defined as the euclidean distance between them as follows:

$$d(i, j) = \sqrt{\sum_{d=1}^{D} (x_i^d - x_j^d)^2}. \tag{4}$$

The distance of two clusters r and s in G, which is an element in M in Algorithm 3 and is denoted $M(r, s)$, is defined as the distance of the two closest particles i and j that belong to clusters r and s respectively. $M(r, s)$ can be formulated as

$$M(r, s) = \min_{i \in r, j \in s} d(i, j). \tag{5}$$

Given a cradle swarm C, the clustering method works as follows. It first creates a list G of clusters with each cluster only containing one particle in C. Then, in each iteration, it uses Algorithm 4 to find a pair of clusters r and s such that they are the closest among those pairs of clusters, of which the total number of particles in the two clusters is not greater than $max\_subsize$ ($max\_subsize$ is a prefixed maximum subswarm size), and, if successful, combines r and s into one cluster. This iteration continues until all clusters in G contain more than one particle. The value of $max\_subsize$ directly determines how many clusters can be obtained by the hierarchical clustering method. Definitely, it also determines the number of subswarms.

From the above description, it can be seen that using the above clustering method, subswarms will be automatically created depending on the distribution of initial particles in the fitness landscape. The number of subswarms and the size of each subswarm are also automatically determined by the fitness landscape and the unique parameter $max\_subsize$.

In this paper, we have removed the training process used in the original CPSO [26]. In [26], the aim of training the initial swarm is to guide particles to move toward different promising subregions. After the training process, the clustering operation will be conducted to generate subswarms. From the experimental results, we found that training for the initial

Algorithm 4 *FindNearestPair*(G, r, s)

```
found := FALSE;
min_dist := sqrt(sum_{i=1}^{D} (U_i - L_i)^2), where U_i and L_i are the upper and lower bounds of the ith dimension of the search space;
for i := 0 to |G| do
    for j := i + 1 to |G| do
        if (|G[i]| + |G[j]| > max_subsize) then
            continue;
        end if
        if (min_dist > M(G[i], G[j])) then
            min_dist := M(G[i], G[j]);
            r := G[i];
            s := G[j];
            found := TRUE;
        end if
    end for
end for
Return found;
```

Algorithm 5 *LocalSearch*(S, *evals*)

```
for each particle i ∈ S do
    Update particle i according to (1) and (2);
    evals:=++evals%U;
    if particle i is better than pbest_i then
        Update pbest_i;
        LearnGBest(particle i, gbest, evals);
        if particle i is better than gbest then
            Update gbest;
        end if
    end if
end for
```

swarm is not necessary. There are no overlapping search areas among the subswarms that are produced from the initial swarm using the clustering method. Exploitation will then be carried out immediately in the own local search areas of the subswarms and the subswarms will gradually move toward the local optima that are close to them respectively. Finally, all subswarms will distribute in different subregions where local optima are located in the fitness landscape. So, the same objective as the training process used in the original CPSO [26] can be achieved without training in this paper. The test results regarding CPSO with and without training will be shown later in the experimental study section in this paper. The advantage of removing the training phase in the original CPSO lies in that more computational resources can be distributed to subswarms to perform local search. The refining clustering operation in the original CPSO is also removed in this paper because the number of subswarms can be controlled by the value of $max\_subsize$. Setting a proper value for $max\_subsize$ can help CPSO allow one subswarm to cover a single peak. So, the refining clustering phase is also redundant. For example, if we set $max\_subsize$ to an extreme value (e.g., $max\_subsize = 1$), then each subswarm contains only one particle and can just cover a single peak.

It should be noted that it is very difficult for algorithms to track all peaks in the fitness landscape, especially when we use a limited population resources to solve a problem with a large number of peaks in the search space. However, we can just track the peaks that have relatively higher heights compared with the other peaks in the fitness landscape since these peaks have a higher probability of becoming the highest peak in the next environment. In CPSO, if one subswarm covers more than one peak in a local subregion, particles would focus on the search on the relatively higher peaks in that local subregion.

C. Local Search Strategy

When a subswarm is created using the above clustering method, it will undergo the local search process in order to exploit the subregion covered by the subswarm. The framework of the local search process is described in Algorithm 5. In the local search process, in order for a subswarm to locate a local peak quickly, the PSO with *gbest* model is used. That is, each particle in a subswarm also learns from the global best position *gbest* found by the subswarm.

Algorithm 6 *LearnGBest*(*particle i*, *gbest*, *evals*)

```
for each dimension d of gbest do
    x⃗_{t_gbest} := x⃗_gbest {x⃗_{t_gbest} is a temporary particle};
    x_{t_gbest}[d] := x_i[d];
    if x⃗_{t_gbest} is better than x⃗_gbest then
        x_gbest[d] := x_{t_gbest}[d];
    end if
    evals:=++evals%U;
end for
```

In order to speed up the convergence process of subswarms, a linear decreasing scheme is also used in CPSO to adjust the inertia weight ω in (1) as follows:

$$\omega = \omega_{\max} - \frac{(\omega_{\max} - \omega_{\min}) \times c\_itr}{r\_itr} \tag{6}$$

where $\omega_{\max}$ and $\omega_{\min}$ are respectively the maximum and minimum value of ω, c_itr is the iteration counter for a subswarm, which starts from 0 when a subswarm is newly created, and r_itr is the remaining iterations before the next change when a subswarm is created, i.e., $r\_itr = (U - evals)/pop\_size$, where pop_size is the total number of particles in all subswarms and the cradle swarm.

In the basic PSO algorithm, the global best position *gbest* is updated when any particle finds a better position than the current *gbest*. Once *gbest* is updated, the information of all dimensions of *gbest* is replaced with that of the better position found. This updating mechanism has a disadvantage: the promising information of some dimensions in one particle cannot be kept due to the bad information in other dimensions that causes the low fitness of the particle. This problem is called "two step forward, one step back" in [40]. If a particle gets better, the information of some dimensions probably becomes more promising. Other particles should learn from such useful information relevant to some dimensions of that particle although its fitness may be low.

Based on the above discussion, we introduce a new learning method into CPSO during the local search process of each subswarm. This learning method tries to extract useful information relevant to those potentially improved dimensions of an improved particle to update the *gbest* of the subswarm, as shown in Algorithm 6. When a particle i in a subswarm finds a better position, we iteratively check each dimension of *gbest*: replace the dimension with the corresponding dimensional value of particle i if *gbest* is updated by doing so. In this way, *gbest* learns the useful information from those dimensions of a particle that has been improved. This learning method is time consuming. Hence, it is not used on all particles of a subswarm. Instead, we choose the *gbest* as the learner for each subswarm. Our experimental study shows that this strategy makes the convergence speed very fast, which is favorable for CPSO in dynamic environments.

D. Check the Status of Subswarms

After the local search operation, subswarms are checked regarding overlapping, convergence, and overcrowding. The checking of subswarms is as shown in Algorithm 7.

Algorithm 7 *CheckSubswarms*(C, $slst$, $clst$)

for each pair of subswarms (r, s) in $slst$ **do**
  if $r_{\text{overlap}}(r, s) > R_{\text{overlap}}$ **then**
    Merge r and s into r;
    Remove s from $slst$;
  end if
end for
for each subswarm $r \in slst$ **do**
  if $|r| > max\_subsize$ **then**
    Remove worst $(|r| - max\_subsize)$ particles from r;
  end if
end for
for each subswarm $s \in slst$ **do**
  if $radius(s) < R_{\text{conv}}$ **then**
    Add $gbest$ into $clst$;
    Remove s from $slst$;
  end if
end for
if $|C| = 0$ && $|slst| = 0$ **then**
  Add $max\_subsize$ random particles into C;
end if

Traditionally, the overlapping check between two subswarms is carried out using their search radius. The search radius of a subswarm s can be calculated as follows:

$$radius(s) = \frac{1}{|s|} \sum_{i \in s} d(i, s_{\text{center}}) \tag{7}$$

where s_{center} is the center position of the subswarm s and $|s|$ is the size of s. If any particle in a subswarm is within the search radius of another subswarm, then the overlapping search occurs. If the distance of the best particles of two subswarms is less than their search radius, then they are combined or one of them is removed. The above checking mechanism assumes that each subswarm just covers one peak. However, it is not true for real PSO algorithms. If a subswarm in a subregion covers more than one peak, other subswarms that are within its search area should not be removed or combined together with this subswarm.

In CPSO, we adopt the following overlapping check scheme. If two subswarms r and s are within each other's search area, an overlapping ratio between them, denoted $r_{\text{overlap}}(r, s)$, is calculated as follows. We first calculate the percentage of particles in r which are within the search area of s and the percentage of particles in s which are within the search area of r, and then set $r_{\text{overlap}}(r, s)$ to the smaller one of the two percentages. The two subswarms r and s are combined only when $r_{\text{overlap}}(r, s)$ is greater than a threshold value R_{overlap}, which is set to 0.7 in this paper.

In order to avoid too many particles searching on a single peak and hence save computing resources, an overcrowding check is performed on each subswarm in CPSO after the above overlapping check. If the number of particles in a subswarm is greater than $max\_subsize$, then the particles with the worst personal best positions are removed one by one until the size of the subswarm is equal to $max\_subsize$.

Algorithm 8 *DetectChange*(C, $slst$, $clst$, $evals$)

Re-evaluate the global best particle over all subswarms;
$evals$:=++$evals\%U$;
if The fitness of the re-evaluated position changes **then**
  Save the $gbest$ of each subswarm in $slst$ into $clst$;
  Remove all subswarms in $slst$;
  Generate a new cradle swarm C;
  Add the particles in $clst$ into C;
  Empty $clst$;
  Return $TRUE$;
else
  Return $FALSE$;
end if

For DOPs, the best solutions found in the current environment may be useful for tracking the movements of peaks in the next environment. Hence, in CPSO, after the crowding check, the convergence check is carried out to see whether a subswarm has converged. A subswarm convergence list $clst$ is used to record the best positions found by those converged subswarms in the current environment. If the radius of a subswarm is less than a small threshold value R_{conv}, which is set to 0.0001 in this paper, the subswarm is regarded to be converged on a peak. If a subswarm is converged, its $gbest$ is added into $clst$ in order to be used in the next environment. Correspondingly, the converged subswarm is removed form the subswarm list: $slst$.

The removal of converged subswarm and combining two overlapping subswarms may result in the consequence of no particle surviving. If this happens, the algorithm will run forever. Therefore, if all subswarms are converged, $max\_subsize$ random particles will be generated into the current cradle swarm C to deal with the special situation.

E. Detecting Environmental Changes

Usually, for an algorithm to address DOPs efficiently, it is important to detect the environmental changes [35]. To detect the environmental changes, we may use the deterioration of the population performance or the time-averaged best performance as indicator [8]. The fitness landscape change will affect all particles based on our experimental test on the MPB problem. Based on this fact, we can figure out several simple efficient methods to detect the environmental changes. Before updating $pbest$ of each particle, we may re-evaluate its $pbest$ position (e.g., the method used in [31]). If the fitness changes, it means that a change of the fitness landscape occurs. Another simple approach is to set several monitoring particles in the search space. The monitoring particles will be re-evaluated every iteration. If the environment changes, it will be detected by these monitoring particles using the above detecting method.

In CPSO, we use the global best particle over all subswarms as the monitoring particle to detect the environmental changes. Before updating the global best particle, we re-evaluate its fitness at each iteration. If its fitness changes, it indicates that an environmental change occurs. Once an environmental change is detected, CPSO takes the actions shown in Algorithm 8; otherwise, if it fails to detect the

change, the previous population will be used in the new environment. In order to adapt to the new environment quickly, we take the following actions. First, the best position *gbest* of each subswarm in *slst* before the change is saved into *clst*. Then, all particles are re-initialized to create a new cradle swarm and the particles stored in *clst* are added to the new cradle swarm by replacing the worst particles in it. Finally, *slst* and *clst* are re-set to be empty. Again, the clustering method will be performed to generate the new subswarm list.

F. Complexity Analysis

The major components of CPSO are the clustering operation, local search, and status checking of each subswarm. The clustering operation is performed only once at the very beginning when an environmental change is detected. From Algorithm 3, it can be seen that the time complexity of the clustering operation is $O(M^3)$, where M is the population size of the cradle swarm. We first compute all distances among each pair of particles in $O(M^2)$. For each iteration of merging two clusters r and s from the cluster list G, we find the nearest pair of clusters of G in $O(|G|^2)$ (if we use dynamic programming method, it would be reduced to $O(|G|\log(|G|))$), then update the distance matrix M in $O(|G|)$. The number of clusters in G will decease by 1 every iteration until the stop criteria is met. Finally, we perform the clustering operation in $O(M^3)$.

In the local search operation (Algorithm 5) for each subswarm, except performing the *gbest* learning on improved particles, there is no big difference from the basic PSO algorithm. In addition, the time complexity will reduce as performing overlapping and overcrowding check because of decreasing number of total particles.

The time complexity of the subswarm status check depends on how many subswarms produced by the clustering operation. However, it also will decrease as performing overlapping and convergence check for subswarms. In total, according to the above component complexity analysis, the extra computing time needed for CPSO is not so high in comparison with the basic PSO algorithm.

G. CPSO and PSO with the lbest Model

In the PSO with the *lbest* model (PSO_{lbest}), if we define the neighborhood of a particle as its nearest $max\_subsize$ particles, and assign the best one of its neighborhood as its social component in (1), it seems that we will get a similar search behavior as CPSO. This is because the clustering method in CPSO will assign the particles that are close to each other into a subswarm, which is the same as the neighborhood defined in PSO_{lbest}. However, CPSO has several major advantages in comparison with PSO_{lbest}.

First, CPSO can track multiple optima in dynamic environments. In CPSO, if more than one subswarms cover a same peak, they will finally be combined with each other into one subswarm by the overlapping check function. Hence, the positions found by the converged subswarms in *clst* are distributed on different peaks. Once an environmental change occurs, the elements in *clst* will be added into the new cradle swarm. When a peak moves not too far away from the previous location, the previous peak position locates on the slope of the new current peak. As we know, if this case happens, the previous peak location does help the search of the current peak in the new environment. However, PSO_{lbest} cannot recognize such kind of peak locations even they are found by PSO_{lbest}. It is impossible to directly check which particles in the whole swarm are from different peaks since different peaks have quite different heights. Therefore, PSO_{lbest} cannot track multiple optima in dynamic environments.

Second, CPSO can control overcrowding in a single peak. There are two aspects regarding the overcrowding over a single peak in CPSO. One is that more than one subswarms cover a single peak, and the other is that too many particles exist within one subswarm on a peak. The first problem can be solved by the overlapping check as analyzed above. Hence, those subswarms that are inferior to the best subswarm on a peak will be automatically removed. The second problem is solved by the overcrowding check in Algorithm 7. However, PSO_{lbest} cannot solve the overcrowding problem since it cannot check which particles locate on which peaks.

Third, CPSO has a higher probability of covering more local optima than PSO_{lbest} can do. In CPSO, since there is no communication among subswarms, each subswarm just searches its local area, and finally will converges on a local optima if it survives till the next change takes place. Hence, every peak will be found if it is covered by a subswarm. However, in PSO_{lbest}, the *gbest* with a relatively better fitness of one particle's neighborhood may belong to the neighborhood of different particles. That is, particles from different peaks may share the same *gbest*. So, particles from the peaks with lower heights will be attracted by the *gbest* from the peak with a higher height. Finally, they will converge on that peak with a higher height and lose the track of the peaks with relatively lower heights.

Finally, CPSO can partially adaptively adjust the number of particles and subswarms needed to achieve the best performance based on its work mechanism. However, the number of particles in PSO_{lbest} is fixed during the whole run.

The experimental results of comparing CPSO and PSO_{lbest} will be presented later in the experimental section.

V. Experimental Study

In this section, three groups of experiments were carried out based on the MPB problem [6]. The objective of the first group of experiments is to investigate the work mechanism of CPSO, analyze the sensitivity of key parameters, and study the effect of the training process used in the original CPSO [26]. In the second group of experiments, the performance of CPSO is compared with a number of PSO algorithms taken from the literature. The involved algorithms include mCPSO [5], mQSO [5], SPSO [31], rSPSO [2], and CESO [28]. All the results of the peer algorithms shown in this paper are provided in the papers where they were proposed. Finally, we give the comparison results between CPSO and PSO with the *lbest* model in the third group of experiments.

For the convenience of description, the configuration of CPSO is represented by $C(M, N)$ in this paper, where M is the initial swarm size of the cradle swarm and N is the value of $max\_subsize$. In CPSO, the acceleration constants η_1 and η_2 were both set to 1.7. The inertia weight ω is linearly decreased from $\omega_{\max} = 0.6$ to $\omega_{\min} = 0.3$ using (5) for subswarms to perform local search. The value of R_{overlap} was set to 0.7 for the MPB problem.

A. Experimental Setup

1) *Moving Peaks Benchmark (MPB) Problem:* The MPB problem proposed by Branke [6] has been widely used as dynamic benchmark problems in the literature. Within the MPB problem, the optima can be varied by three features, i.e., the location, height, and width of peaks. For the D-dimensional landscape, the problem is defined as follows:

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

where $H_i(t)$ and $W_i(t)$ are the height and width of peak i at time t, respectively, and $X_{ij}(t)$ is the jth element of the location of peak i at time t. The p independently specified peaks are blended together by the "max" function. The position of each peak is shifted in a random direction by a vector $\vec{v}_i$ of a distance s (s is also called the shift length, which determines the severity of the problem dynamics), and the move of a single peak can be described as follows:

$$\vec{v}_i(t) = \frac{s}{|\vec{r} + \vec{v}_i(t-1)|}((1-\lambda)\vec{r} + \lambda\vec{v}_i(t-1)) \tag{9}$$

where the shift vector $\vec{v}_i(t)$ is a linear combination of a random vector $\vec{r}$ and the previous shift vector $\vec{v}_i(t-1)$ 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:

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

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

$$\vec{X}_i(t) = \vec{X}_i(t)(t-1) + \vec{v}_i(t) \tag{12}$$

where σ is a normal distributed random number with mean zero and variation of 1.

2) *Experimental Settings:* The default settings and definition of the benchmark used in the experiments of this paper can be found in Table I, which are the same as in all the involved algorithms. In Table I, the term "change frequency (U)" means that environment changes every U fitness evaluations, S denotes the range of allele values, and I denotes the initial height for all peaks. The dynamism of changes is described as follows. The height of peaks is shifted randomly in the range $H = [30, 70]$ and the width of peaks is shifted randomly in the range $W = [1, 12]$.

TABLE I
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

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 kth environmental change, h_k is the optimum value of the kth 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 $K = 100$ environments, which result in $K \times U = 5 \times 10^5$ fitness evaluations. All the results reported are based on the average over 50 independent runs with different random seeds.

B. Experimental Investigation of CPSO

1) *Testing the Work Mechanism of CPSO:* In order to understand the work mechanism of CPSO, in this section experiments are carried out to investigate the dynamic behavior of internal features of CPSO, including the number of subswarms, the number of total particles, and the number of converged subswarms, during the solving process. In the experiments, the configuration of $C(100, 4)$ was applied for CPSO. Fig. 1 shows the average dynamic behavior of these features and the offline error against the solving process for five environmental changes based on the default settings of the MPB problem over 50 runs.

Fig. 1(a) and (b) clearly presents the changes of the total number of particles and the total number of subswarms during the evolution process. When CPSO is configured to $C(100, 4)$, the largest population size of each subswarm is 4. So, we can estimate that the total number of initial subswarms produced by the clustering method should be a little larger than 25, which can be observed from Fig. 1(a) where an environmental change has just occurred. Since there are ten peaks in the whole fitness landscape, we need to remove some redundant subswarms to achieve the best performance. This is automatically conducted by the overlapping check mechanism. It can be seen from Fig. 1(a) that the number of subswarms decreases with the evolutionary process in each environmental change. When approaching the next environment, some subswarms converge on some different peaks and they are recorded in $clst$, which can be seen in Fig. 1(c). Due to the overlapping

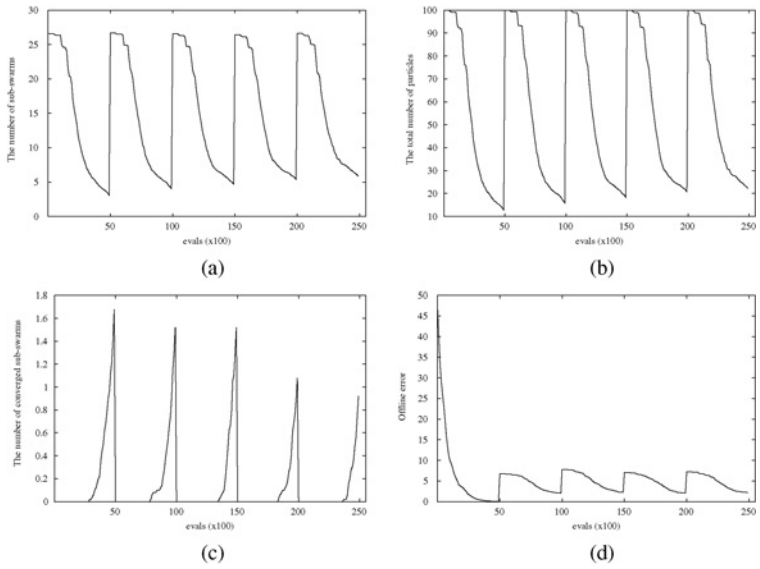

Fig. 1. Dynamic behavior of CPSO regarding (a) number of subswarms, (b) number of total particles, (c) number of converged subswarms, and (d) offline error for five environmental changes.

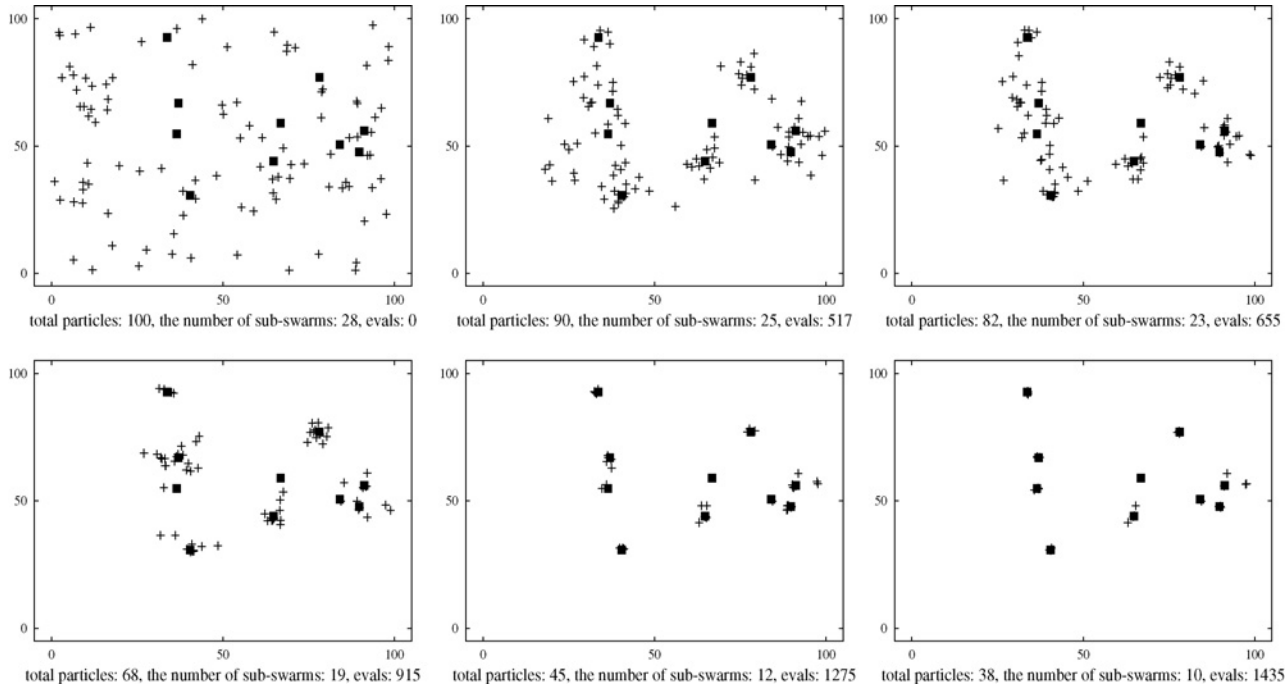

Fig. 2. *pbest* locations at different *evals* within a single environmental change of a typical run of CPSO on a 2-D fitness landscape.

check mechanism and the convergence check mechanism, the number of the total particles decreases during each environmental change. This process can be seen from Fig. 1(b). From Fig. 1(d), we can see that the offline error reaches almost zero for the initial environment, that is because the ten peaks have the same initial height in the fitness landscape, which enable the algorithm to easily find one or some of the peaks.

In order to visualize the search trajectories of all particles in different evolutionary stages, we give the *pbest* locations of particles at different *evals* within a single environmental change of a typical run of CPSO on a 2-D fitness landscape in Fig. 2. In Fig. 2, the cross and black square points are the *pbest* positions and the locations of ten peaks, respectively. From left to right and from top to bottom, the six images in Fig. 2 show the movements of *pbest* positions as the evolution progresses. Overlapping and overcrowding happen when more than one subswarms move toward a same peak. At this moment, overlapping and overcrowding check will take effect to remove the redundant particles. The mechanism can be seen by the changing distribution with the deceasing number of particles and subswarms. When *evals* reaches 1275 in Fig. 2, the number of total particles reduces to 45 and the number of subswarms decreases from 28 to 12. Finally, just before the environmental change occurs, the positions in

TABLE II
OFFLINE ERROR OF DIFFERENT PARAMETER CONFIGURATIONS

	$M=10$	$M=30$	$M=50$	$M=70$	$M=100$	$M=120$	$M=150$	$M=200$
$N=2$	**3.79**	**1.77**	1.39	1.48	2.31	3.12	3.95	5.05
$N=3$	4.47	1.98	**1.41**	**1.06**	1.3	1.85	3.1	4.28
$N=4$	4.82	2.47	1.77	1.47	**1.11**	**1.2**	1.67	3.36
$N=5$	6.05	2.76	1.9	1.61	1.27	1.33	**1.47**	2.67
$N=6$	6.61	3.64	2.21	1.85	1.26	1.21	1.58	2.59
$N=7$	5.91	3.41	2.57	2.05	1.74	1.44	1.49	**1.94**
$N=10$	8.27	4.63	3.44	2.7	2.04	1.83	1.88	2.03
$N=12$	7.82	4.82	3.45	2.88	2.22	2.15	1.9	2.13
$N=15$	8.65	5.73	3.87	3.29	2.94	2.44	2.32	2.28

TABLE III
NUMBER OF SUBSWARMS CREATED BY THE CLUSTERING METHOD

	$M=10$	$M=30$	$M=50$	$M=70$	$M=100$	$M=120$	$M=150$	$M=200$
$N=2$	5	15	25	35	50	60	75	100
$N=3$	4	10.4	17.4	24.4	34.7	41.5	51.8	69
$N=4$	3.07	8.27	13.5	18.7	26.6	31.8	39.7	52.7
$N=5$	2.39	6.73	11	15.3	21.7	25.9	32.3	42.8
$N=6$	2.29	5.75	9.52	13	18.3	21.9	27.2	36.1
$N=7$	2.14	5.29	8.39	11.3	16	19	23.6	31.2
$N=10$	1.48	3.72	5.95	8.15	11.4	13.6	16.8	22.2
$N=12$	1.44	3.5	5.33	6.92	9.76	11.4	14.2	18.6
$N=15$	1.46	2.61	4.49	5.76	7.91	9.29	11.5	15.1

TABLE IV
NUMBER OF SURVIVED SUBSWARMS

	$M=10$	$M=30$	$M=50$	$M=70$	$M=100$	$M=120$	$M=150$	$M=200$
$N=2$	3.68	7.48	10.7	14.4	21.8	28.3	39.8	63
$N=3$	3.45	5.44	6.97	8.45	10.8	13.2	21.9	41.5
$N=4$	2.56	4.68	5.71	6.63	7.72	8.5	10	19.5
$N=5$	2.08	4	5.12	5.86	7	7.41	8.47	12.4
$N=6$	1.98	3.59	4.82	5.48	6.62	7.01	7.49	9.98
$N=7$	1.88	3.39	4.53	5.08	5.89	6.61	7.06	8.4
$N=10$	1.33	2.68	3.55	4.21	4.99	5.53	6	6.57
$N=12$	1.31	2.55	3.31	3.78	4.64	5.01	5.57	6.25
$N=15$	1.34	2.07	2.97	3.38	4.07	4.54	4.85	5.76

TABLE V
NUMBER OF PEAKS FOUND BY CPSO

	$M=10$	$M=30$	$M=50$	$M=70$	$M=100$	$M=120$	$M=150$	$M=200$
$N=2$	3.67	5.92	6.82	7.27	7.52	7.7	7.9	8
$N=3$	3.13	5.18	6.39	7.15	7.71	7.88	8.33	8.72
$N=4$	2.79	4.53	5.69	6.36	6.99	7.33	7.76	8.27
$N=5$	2.42	4.11	5.03	5.85	6.5	6.98	7.36	8.02
$N=6$	2.33	3.9	4.89	5.46	6.17	6.77	7.05	7.73
$N=7$	2.24	3.58	4.55	5.03	5.74	6.06	6.49	7.22
$N=10$	1.65	3.01	3.88	4.39	4.87	5.29	5.71	6.26
$N=12$	1.66	2.93	3.66	4.03	4.79	4.98	5.56	6.14
$N=15$	1.76	2.48	3.33	3.65	4.23	4.59	5.03	5.48

clst and the best particles in subswarms will be recorded for tracking the movement of peaks in the next environment.

From the first image of the initial stage to the last image of the final stage in Fig. 2, it can also be seen that particles gradually move toward subregions where the ten peaks are located. Finally, they converge on these peaks. This observation validates our previous analysis that the training process in the original CPSO is not necessary.

2) *Effect of Varying the Configurations:* The aim of this set of experiments is to examine the effect of the different configurations on the performance of CPSO. The default parameter settings of the MPB problem were used in these experiments. CPSO was run 50 times with each of the combined values of *max_subsize* (N) in $\{2, 3, 4, 5, 7, 10, 15\}$ and the initial size of the cradle swarm (M) in $\{10, 30, 50, 70, 100, 120, 150, 200\}$. The offline error is shown in Table II, where the best result over all values of *max_subsize* for each fixed number of initial population size of the cradle swarm is shown in bold font. Table III shows the number of subswarms created from the cradle swarm using the clustering method. Table IV gives the results of the number of survived subswarms before the environment changes. The survived subswarms include those converged subswarms and nonconverged subswarms at the last generation before a change occurs. The number of peaks found by CPSO is presented in Table V. If a peak is within the radius of a survived subswarm, it is considered to be found by CPSO. Strictly speaking, we cannot assume that a peak has been found just because it is within a subswarm's radius. We use this simple approximate measure to consider whether a peak is found or not since it works for the purpose of our experiments here, i.e., to compare the relative performance of different configurations of the initial swarm size M and *max_subsize* (N) in terms of locating peaks. This performance measue is derived from the measure used in [31].

From Table II, it can be seen that the different configurations of CPSO significantly affect the performance of CPSO. When the maximum subswarm size, i.e., N, is fixed a specific value, setting the initial size of the cradle swarm, i.e., M, to a too large or too small value will affect the performance of CPSO, vice versa. The optimal configuration of CPSO for the default settings of the MPB problem with ten peaks is $C(70, 3)$, which enables CPSO to achieve the smallest offline error of 1.06.

As discussed above, the value of *max_subsize* directly determines the number of subswarms that are generated by the clustering method. For example, if we take the extreme value of *max_subsize* which is equal to the size of the cradle swarm, only one subswarm may be obtained. It can be seen from Table III, where the larger the value of *max_subsize* under a fixed size of the initial swarm, the smaller the number of subswarms that are created. Too large or too small *max_subsize* will cause too few or too many subswarms created, which may be far away from the real number of peaks in the search space. This is why the performance of CPSO greatly depends on the value of *max_subsize*. On the other hand, from Table III it can be observed that when the value of *max_subsize* is fixed, the larger the initial population size of the cradle swarm, the larger the number of subswarms that are created.

By observing Tables II and IV, it can be seen that when the number of survived subswarms remains at the level of 7–10, which is slightly smaller than the total number of peaks, i.e., 10, CPSO achieves relatively smaller offline errors that are below 1.5. Although the number of peaks found by CPSO is less than the total number of peaks in the search space, it is reasonable. There might be two possible reasons: first, some peaks of low heights might be covered by the peaks with higher heights and are invisible in the fitness landscape. Second, actually, it is very hard for algorithm to track all peaks

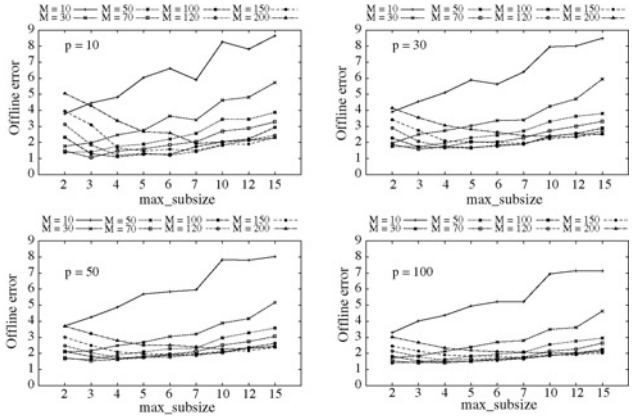

Fig. 3. Offline error of CPSO with different configurations on the MPB problems with different number of peaks.

even they are all visible, especially when there are many peaks in the fitness landscape since the swarm size is limited to track all these peaks.

Comparing the results in Tables II and V, it can be seen that the larger the initial size for the cradle swarm, the more peaks are found by CPSO, which is more close to the total number of peaks in the fitness landscape. Intuitively, the performance of CPSO should get better with a larger initial size for the cradle swarm since the clustering technique in CPSO can find obtain more clusters (and peaks) in the fitness landscape. But, this result is not seen in Table II as expected. This occurs because the change frequency was set to 5000, which may not be big enough to achieve the best performance of CPSO. If the change frequency is increased, we will get better results as expected. The experimental results will be shown later in the following experiments.

Fig. 3 presents the performance of CPSO with different configurations on the MPB problems with the number of peaks set in $\{10, 30, 50, 100\}$. From Fig. 3, similar observations can be obtained on the four problems with different number of peaks. Just as shown in Table II, to achieve the best performance, CPSO needs an optimal configuration. For example, on the ten peaks problem, when the initial population size M is set to a specific number (e.g., $M = 150$), the offline error first decreases as the value of $max\_subsize$ (N) increases from 2 to a turning point 7. Then, after the turning point, the offline error increases. In addition, it can be observed from Fig. 3 that the turning point is different for different configurations. For example, for the ten peaks problem, the turning point is $N = 7$ for $M = 150$, $N = 4$ for $M = 100$, etc. It is understandable. In order to achieve the best performance, CPSO needs to adjust the value of $max\_subsize$ to adapt to the environments when a specific initial population size M is given. From Fig. 3, interestingly, the optimal configuration of CPSO is $C(70, 3)$ on the MPB problems with different number of peaks.

3) *Effect of the Training Process:* In this set of experiments, we test the effect of the training process used in the original CPSO on the performance of CPSO on the MPB problems. Here, the same training method in [26] was applied in CPSO, where the neighborhood of a particle is defined as the nearest particle to that particle. This unique particle in the neighborhood of a particle is used for the particle's velocity update in (1). As analyzed above, the training process in [26] does not help the search for CPSO. In order to give an explanation from the experimental view, experiments were conducted based on the configuration of $C(70, 3)$ for CPSO with different number of iterations for the training process. The comparison results are shown in Table VI.

TABLE VI
RESULTS OF CPSO WITH DIFFERENT NUMBER OF ITERATIONS FOR TRAINING

Training Iterations	0	1	3	5	7	9
Offline error	1.06	1.56	1.65	1.77	1.697	1.81
Subswarms produced	24.4	26.01	26.33	26.55	26.64	26.62
Survived subswarms	8.45	12.60	14.56	15.38	16.49	17.24
Real peaks found	7.5	7.3	7.18	6.96	6.98	6.79

TABLE VII
OFFLINE ERROR OF ALGORITHMS ON THE MPB PROBLEMS WITH DIFFERENT SHIFT SEVERITIES

s	$C(70, 3)$	mCPSO	mQSO	CESO	rSPSO	SPSO
0.0	0.80	1.18	1.18	0.85	0.74	0.95
	±0.21	±0.07	±0.07	±0.02	±0.08	±0.08
1.0	1.056	2.05	1.75	1.38	1.50	2.51
	±0.24	±0.07	±0.06	±0.02	±0.08	±0.09
2.0	1.17	2.80	2.40	1.78	1.87	3.78
	±0.22	±0.07	±0.06	±0.02	±0.05	±0.09
3.0	1.36	3.57	3.00	2.03	2.4	4.96
	±0.28	±0.08	±0.06	±0.03	±0.08	±0.12
4.0	1.38	4.18	3.59	2.23	2.90	2.56
	±0.29	±0.09	±0.10	±0.05	±0.08	±0.13
5.0	1.58	4.89	4.24	2.52	3.25	6.76
	±0.32	±0.11	±0.10	±0.06	±0.09	±0.15
6.0	1.53	5.53	4.79	2.74	3.86	7.68
	±0.29	±0.13	±0.10	±0.10	±0.11	±0.16

From Table VI, it can be seen that the results of CPSO with training are much worse than the results obtained by CPSO without training, where the smallest offline error achieved is 1.06. The training process may cause too many pairs of close particles moving together since the neighborhood of a particle is composed of only the nearest particle. It can be seen from Table VI that too many subswarms are generated by the clustering method due to the training consequence. By observing the number of real peaks found by CPSO, we can find another disadvantage of the training process: the larger number of iterations for training the smaller number of peaks which can be tracked by CPSO. Just as pointed out above, the training process is not necessary since subswarms produced from the cradle swarm without training can also achieve the same objective of training the cradle swarm in [26]. In addition, we can take the advantage of assigning the computing resources for training the cradle swarm to subswarms to perform local search. Therefore, the training process in [26] has been removed in the updated version of CPSO in this paper.

C. Comparison of CPSO With Peer Algorithms

In this group of experiments, we compare the performance of CPSO with mCPSO, mQSO, SPSO, rSPSO, and CESO on the MPB problems with different settings.

TABLE VIII
OFFLINE ERROR OF ALGORITHMS ON THE MPB PROBLEMS WITH DIFFERENT NUMBER OF PEAKS

Peaks	CPSO	mCPSO	mQSO	mCPSO*	mQSO*	CESO	rSPSO	SPSO
1	0.14	4.93	5.07	4.93	5.07	1.04	1.42	2.64
	±0.11	±0.17	±0.17	±0.17	±0.17	±0.00	±0.06	±0.10
2	0.20	3.36	3.47	3.36	3.47	–	1.10	2.31
	±0.19	±0.26	±0.23	±0.26	±0.23	–	±0.03	±0.11
5	0.72	2.07	1.81	2.07	1.81	–	1.04	2.15
	±0.30	±0.08	±0.07	±0.11	±0.07	–	±0.03	±0.07
7	0.93	2.11	1.77	2.11	1.77	–	1.21	1.98
	±0.30	±0.11	±0.07	±0.11	±0.07	–	±0.05	±0.04
10	1.056	2.08	1.80	2.05	1.75	1.38	1.50	2.51
	±0.24	±0.07	±0.06	±0.07	±0.06	±0.02	±0.08	±0.09
20	1.59	2.64	2.42	2.95	2.74	1.72	2.20	3.21
	±0.22	±0.07	±0.07	±0.08	±0.07	±0.02	±0.07	±0.07
30	1.58	2.63	2.48	3.38	3.27	1.24	2.62	3.64
	±0.17	±0.08	±0.07	±0.11	±0.11	±0.01	±0.07	±0.07
40	1.51	2.67	2.55	3.69	3.60	1.30	2.76	3.85
	±0.12	±0.07	±0.07	±0.11	±0.08	±0.02	±0.08	±0.08
50	1.54	2.65	2.50	3.68	3.65	1.45	2.72	3.86
	±0.12	±0.06	±0.06	±0.11	±0.11	±0.01	±0.08	±0.08
100	1.41	2.49	2.36	4.07	3.93	1.28	2.93	4.01
	±0.08	±0.04	±0.04	±0.09	±0.08	±0.02	±0.06	±0.07
200	1.24	2.44	2.26	3.97	3.86	–	2.79	3.82
	±0.06	±0.04	±0.03	±0.08	±0.07	–	±0.05	±0.05

1) *Effect of Varying the Shift Severity:* This set of experiments compare the performance of CPSO with peer algorithms on the MPB problems with different settings of the shift length s. The experimental results regarding the offline error and standard deviation are shown in Table VII. The experimental results of the peer algorithms are taken from the corresponding papers with the configuration that enables them to achieve their best performance. We choose the optimal configuration of $C(70, 3)$ for CPSO. Other parameter settings are the same as above.

From Table VII, it can be seen that the results achieved by CPSO with different configurations are much better than the results of the other five algorithms on the MPB problems with different shift severities. As we know, the peaks are more and more difficult to track with the increasing of the shift length. Naturally, the performance of all algorithms degrades when the shift length increases. However, the offline error of CPSO is only slightly affected in comparison with the other five algorithms. This result shows that CPSO is very robust to locate and track multiple optima even in severely changing environments.

2) *Effect of Varying the Number of Peaks:* This set of experiments investigate how CPSO scales with the number of peaks in the MPB problem. The number of peaks was set to different values in the range form 1 to 200. The configuration of $C(70, 3)$ was chosen for CPSO on the MPB problems with different number of peaks in the experiments. Table VIII presents the experimental results in terms of the offline error and standard deviation of eight algorithms, where the results of the other seven algorithms are provided by the corresponding papers with their optima configuration that enables them to achieve their best performance. In Table VIII, mCPSO* and mQSO* denote mCPSO without anti-convergence and mQSO without anti-convergence, respectively.

From Table VIII, it can be seen that the performance of CPSO is not influenced too much when the number of peaks is increased. Generally speaking, increasing the number of peaks makes it harder for algorithms to track the optima. However, interestingly, the offline error decreases when the number of peaks is larger than 20 for CPSO. Similar trend can also be observed from the results of the other seven algorithms. It seems contrary to our prediction. The reason behind this is that when the number of peaks increases, there will be more local optima that have a similar height as the global optima and hence, there will be a higher probability for algorithms to find relatively better local optima.

Comparing the results of CPSO with the other seven algorithms, the offline error achieved by CPSO is much less than that achieved by all the other algorithms when the number of peaks is less than 20. Although the results of CPSO are slightly worse than the results of CESO when the number of peaks exceeds 30, they are much better than the results of the other six algorithms. In addition, if we increase the value of the change frequency, CPSO can achieve much better results, which can be seen in the following section.

From Table VIII, it can also be seen that CESO outperforms all algorithms including CPSO when the number of peaks is larger than 30. As we know, a large number of peaks needs more subswarms to locate and track. It means that an algorithm with a good diversity maintaining mechanism may perform well to find more relatively better local optima. CESO just benefits from such sort of algorithms: the CDE algorithm [38] which is used as a component algorithm in CESO to maintain the population diversity. However, to locate and track more local optima for CPSO, we need a larger initial swarm and big enough change frequency to globally locate optima in the whole fitness landscape. It can be seen from Table IX (to be described below) that CPSO achieves much better results when the initial swarm size is increased to 120 for many peaks problems (i.e., problems with more than 10 peaks).

3) *Effect of Varying the Environmental Change Frequency:* This set of experiments investigates the effect of different

TABLE IX
OFFLINE ERROR OF CPSO ON THE MPB PROBLEMS WITH DIFFERENT NUMBER OF PEAKS AND THE CHANGE FREQUENCY OF 10 000

Peaks	$M=10$	$M=30$	$M=50$	$M=70$	$M=100$	$M=120$	$M=150$	$M=200$
1	0.008	0.010	0.017	0.022	0.069	0.11	0.494	0.494
2	0.354	0.175	0.0931	0.0929	0.107	0.152	0.447	0.447
5	0.942	0.969	0.375	0.22	0.375	0.321	0.522	0.522
7	1.51	1.02	0.708	0.563	0.401	0.468	0.732	0.732
10	2.39	1.16	0.907	0.625	0.638	0.594	0.873	0.873
20	2.19	1.53	1.22	1.06	0.922	0.809	1.04	1.04
30	2.24	1.57	1.36	1.02	1	0.96	1.12	1.12
40	2.21	1.54	1.31	1.05	0.915	0.964	1.16	1.16
50	2.11	1.47	1.31	1.05	0.982	0.961	1.18	1.18
100	1.79	1.3	1.13	0.945	0.925	0.932	1.14	1.14
200	1.53	1.09	0.941	0.802	0.773	0.843	1.03	1.03

TABLE X
OFFLINE ERROR OF CPSO AND PSO_{lbest}

	$N=2$	$N=3$	$N=4$	$N=5$	$N=6$	$N=7$	$N=10$	$N=12$	$N=15$
CPSO	2.31	1.3	1.11	1.27	1.26	1.74	2.04	2.22	2.94
PSO_{lbest}	9.85	9.40	8.51	8.74	8.25	8.27	8.14	9.22	8.70

environmental changing speeds on the performance of CPSO. The *max_subsize* was set to 3 in this set of experiments. Table IX presents the results of CPSO with different initial size of the cradle swarm on the MPB problems with different number of peaks when the value of the change frequency is increased from the default setting of 5000 to 10 000.

From Table IX, it can be seen that as the performance of CPSO gets much better as the size of the cradle swarm increases from $M=30$ until $M=120$ when we increase the value of the change frequency, where the smallest offline error of CPSO on the MPB problems with different number of peaks is less than 1.0 in Table IX. For example, the offline error of CPSO with M = 100 on the MPB problem with ten peaks is now 0.638, which is much lower than the value of 1.3 for CPSO with the same configuration but on the MPB problem with the change frequency set to 5000, as seen from Table II. This result is reasonable since increasing the value of the change frequency gives CPSO more time to search before the next environmental change occurs.

D. Comparison of CPSO and PSO_{lbest}

In this set of experiments, we test the aforementioned advantages of CPSO over PSO with the *lbest* model, i.e., PSO_{lbest}. Both CPSO and PSO_{lbest} were run under a fair algorithm setting. The same *gbest* learning strategy used in CPSO was used in PSO_{lbest}. For a particle in PSO_{lbest}, we define the *max_subsize* nearest particles as its neighborhood. The best particle of its neighborhood is assigned to *gbest*, which is used in (1). The same linearly decreasing ω strategy was used for PSO_{lbest}. For both algorithms, the initial swarm size was set to 100 and the default problem settings were used. Table X presents the experimental results of the two algorithms with different settings of *max_subsize*.

From Table X, it can be seen that all the results of CPSO are much better than the results of PSO_{lbest}. The comparison results obviously show the advantages of CPSO over PSO_{lbest}. Together with the above experimental results, CPSO shows its outstanding capability of tracking multiple optima, overcrowding control, and adaptively adjusting the number of particles needed when solving problems in dynamic environments.

VI. CONCLUSION AND FUTURE WORK

Particle swarm optimization algorithms have been applied to address DOPs in recent years with some promising results. For DOPs, it is usually important that an optimization algorithm should be able to locate and track multiple changing optima over time. To address this problem, researchers have applied the multiswarm method to enhance the performance of PSO algorithms for DOPs. However, for the multiswarm method to work efficiently, there are several important yet difficult issues to address, including how to guide particles to different promising subregions, how to determine the proper number of subswarms, how to calculate the search area of each subswarm, and how to create subswarms.

This paper investigated a CPSO algorithm for locating and tracking multiple optima in dynamic environments. In order to track and locate multiple optima, a single linkage hierarchical clustering method was applied in CPSO to create clusters of particles to form subswarms. With this clustering method, the proper number of subswarms is automatically determined and the search area of each subswarm is also automatically calculated. When a subswarm is created, it will undergo the local search process to exploit the promising subregion for optimal solutions. In order to speed up the searching of a subswarm, a new learning mechanism was introduced for its global best particle to learn from those particles that are improved during the local search process. In order to address environmental changes directly, a restart scheme with reservation of best positions found in the previous environment was applied in CPSO.

In order to justify the proposed CPSO, experiments were carried out to compare the performance of CPSO with several advanced PSO algorithms that are the state-of-the-art algorithms for DOPs in real space (i.e., the SPSO and rSPSO algorithms [5], [2], the mCPSO and mQSO algorithms with and without anti-convergence [31], CESO [28]) on the widely used MPB generator [6].

From the experimental results, the following conclusions can be drawn on the dynamic test environments. CPSO greatly improves the performance of PSO in terms of tracking and locating multiple optima in a dynamic fitness landscape with multiple changing peaks by introducing the clustering method. The performance of CPSO scales well regarding the change severity in the fitness landscape in comparison with other peer PSO algorithms. CPSO performs much better than mCPSO, mQSO, SPSO, rSPSO, and CESO in locating and tracking multiple changing peaks in dynamic environments, especially in severely changing environments. The performance of CPSO also scales well regarding the number of peaks in dynamic environments. When the number of peaks in the fitness landscape is relatively small (e.g., less than 20), CPSO outperforms all the other peer algorithms.

Generally speaking, CPSO can effectively locate and track multiple optima in dynamic environments. The experimental results indicate that the proposed CPSO can be a good optimizer in dynamic environments, especially for a dynamic fitness landscape with multiple changing peaks.

There are several relevant works to pursue in the future. First, although the clustering method applied in CPSO is effective to generate subswarms, it is still difficult to create accurate subswarms, especially for the situation when a single peak is covered by only one particle. More work should be done to solve this problem.

Second, CPSO cannot explore an untouched area in the search space when no particles cover that area since all subwarms only concern exploitation in their own local search area in CPSO. Introducing new local search algorithms may solve this problem to improve the exploring capability of CPSO.

Third, in the CPSO studied in this paper, when a subswarm becomes converged or overcrowded, we just remove the subswarm or the overcrowded particles from the subswarm. Hence, the whole population size may become smaller and smaller during the solving process. However, according to our preliminary experiments, simply adding corresponding number of random particles into the cradle swarm does not work well since they may be attracted to those existing subswarms. It is worth investigating how to effectively add particles into the cradle swarm to maintain the whole population size at an optimal value.

How to deal with the environmental changes is another important issue. We need to introduce more effective methods rather than a simple restart with elitism scheme to address the dynamism in DOPs.

Finally, it would be also interesting to combine other techniques into CPSO to further improve its performance in dynamic environments. For example, the $max\_subsize$ parameter has a great impact on the performance of CPSO. More research could be conducted to look at adaptation or self-adaptation of $max\_subsize$ and the population size to adjust them according to the environmental dynamics and the properties of the base function. To the opposite direction, extending the ideas applied in CPSO, e.g., the clustering idea and the learning technique for updating the global best particle of a subswarm, to other algorithms may also be valuable to improve their performance in dynamic environments.

REFERENCES

[1] S. Bird and X. Li, "Adaptively choosing niching parameters in a PSO," in *Proc. Genetic Evol. Comput. Conf.*, 2006, pp. 3–10.

[2] S. Bird and X. Li, "Using regression to improve local convergence," in *Proc. IEEE Congr. Evol. Comput.*, 2007, pp. 592–599.

[3] T. M. Blackwell, "Particle swarms and population diversity II: Experiments," in *Proc. Genetic Evol. Comput. Workshops*, 2003, pp. 108–112.

[4] T. M. Blackwell and J. Branke, "Multiswarm optimization in dynamic environments," in *Proc. EvoWorkshops: Appl. Evol. Comput.*, LNCS 3005. 2004, pp. 489–500.

[5] T. M. Blackwell and J. Branke, "Multiswarms, exclusion, and anti-convergence in dynamic environments," *IEEE Trans. Evol. Comput.*, vol. 10, no. 4, pp. 459–472, Aug. 2006.

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

[7] J. Branke, T. Kaußler, C. Schmidt, and H. Schmeck, "A multi-population approach to dynamic optimization problems," in *Proc. 4th Int. Conf. Adaptive Comput. Des. Manuf.*, 2000, pp. 299–308.

[8] J. Branke, *Evolutionary Optimization in Dynamic Environments*. Norwell, MA: Kluwer, 2002.

[9] R. Brits, A. P. Engelbrecht, and F. van den Bergh, "Solving systems of unconstrained equations using particle swarm optimization," in *Proc. IEEE Conf. Syst., Man, Cybern.*, 2002, pp. 102–107.

[10] R. Brits, A. Engelbrecht, and F. van den Bergh, "A niching particle swarm optimizer," in *Proc. 4th Asia-Pacific Conf. Simulated Evol. Learn.*, vol. 2. 2002, pp. 692–696.

[11] L. T. Bui, H. A. Abbass, and J. Branke, "Multiobjective optimization for dynamic environments," in *Proc. Congr. Evol. Comput.*, vol. 3. 2005, pp. 2349–2356.

[12] M. Clerc and J. Kennedy, "The particle swarm: Explosion, stability and convergence in a multidimensional complex space," *IEEE Trans. Evol. Comput.*, vol. 6, no. 1, pp. 58–73, Feb. 2002.

[13] H. G. Cobb and J. J. Grefenstette, "Genetic algorithms for tracking changing environments," in *Proc. 5th Int. Conf. Genetic Algorithms*, 1993, pp. 523–530.

[14] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proc. 6th Int. Symp. Micro Mach. Human Sci.*, 1995, pp. 39–43.

[15] J. J. Grefenstette, "Genetic algorithms for changing environments," in *Proc. 2nd Int. Conf. Parallel Problem Solving Nature*, 1992, pp. 137–144.

[16] S. Janson and M. Middendorf, "A hierachical particle swarm optimizer for dynamic optimization problems," in *Proc. EvoWorkshops: Appl. Evol. Comput.*, LNCS 3005. 2004, pp. 513–524.

[17] S. Janson and M. Middendorf, "A hierarchical particle swarm optimizer and its adaptive variant," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 35, no. 6, pp. 1272–1282, Dec. 2005.

[18] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments: A survey," *IEEE Trans. Evol. Comput.*, vol. 9, no. 3, pp. 303–317, Jun. 2005.

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

[20] J. Kennedy, "The particle swarm: Social adaptation of knowledge," in *Proc. Congr. Evol. Comput.*, 1997, pp. 303–308.

[21] J. Kennedy, "Stereotyping: Improving particle swarm performance with cluster analysis," in *Proc. Congr. Evol. Comput.*, 2000, pp. 1507–1512.

[22] J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. San Mateo, CA: Morgan Kaufmann, 2001.

[23] X. Li, "Adaptively choosing neighborhood bests using species in a particle swarm optimizer for multimodal function optimization," in *Proc. Genetic Evol. Comput. Conf.*, 2004, pp. 105–116.

[24] C. Li and S. Yang, "Fast multiswarm optimization for dynamic optimization problems," in *Proc. 4th Int. Conf. Natural Comput.*, vol. 7. 2008, pp. 624–628.

[25] C. Li and S. Yang, "A generalized approach to construct benchmark problems for dynamic optimization," in *Proc. 7th Int. Conf. Simulated Evol. Learn.*, 2008, pp. 391–400.

[26] C. Li and S. Yang, "A clustering particle swarm optimizer for dynamic optimization," in *Proc. Congr. Evol. Comput.*, 2009, pp. 439–446.

[27] C. Li, S. Yang, T. T. Nguyen, E. L. Yu, X. Yao, Y. Jin, H.-G. Beyer, and P. N. Suganthan, "Benchmark generator for CEC'2009 competition on dynamic optimization," Dept. Comput. Sci., Univ. Leicester, Leicester, U.K., Tech. Rep., 2008.

[28] R. I. Lung and D. Dumitrescu, "A collaborative model for tracking optima in dynamic environments," in *Proc. Congr. Evol. Comput.*, 2007, pp. 564–567.

[29] R. W. Morrison and K. A. De Jong, "Triggered hypermutation revisited," in *Proc. Congr. Evol. Comput.*, 2000, pp. 1025–1032.

[30] D. Parrott and X. Li, "A particle swarm model for tracking multiple peaks in a dynamic environment using speciation," in *Proc. Congr. Evol. Comput.*, 2004, pp. 98–103.

[31] D. Parrott and X. Li, "Locating and tracking multiple dynamic optima by a particle swarm model using speciation," *IEEE Trans. Evol. Comput.*, vol. 10, no. 4, pp. 440–458, Aug. 2006.

[32] A. Passaro and A. Starita, "Particle swarm optimization for multimodal functions: A clustering approach," *J. Artif. Evol. Appl.*, vol. 2008, pp. 1–15, 2008.

[33] K. E. Parsopoulos and M. N. Vrahatis, "Recent approaches to global optimization problems through particle swarm optimization," *Nat. Comput.*, vol. 1, nos. 2–3, pp. 235–306, 2002.
[34] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization: An overview," *Swarm Intell.*, vol. 1, no. 1, pp. 33–58, 2007.
[35] H. Richter, "Detecting change in dynamic fitness landscapes," in *Proc. Congr. Evol. Comput.*, 2009, pp. 1613–1620.
[36] *Single-Linkage Clustering*, Wikipedia [Online]. Available: http://en.wikipedia.org/wiki/Single-linkage_clustering
[37] Y. Shi, R. Eberhart, "A modified particle swarm optimizer," in *Proc. IEEE Conf. Evol. Comput.*, 1998, pp. 69–73.
[38] R. Thomsen, "Multimodal optimization using crowding-based differential evolution," in *Proc. Congr. Evol. Comput.*, vol. 2. 2004, pp. 1382–1389.
[39] F. van den Bergh, "An analysis of particle swarm optimizers," Ph.D. dissertation, Dept. Comput. Sci., Univ. Pretoria, Pretoria, South Africa, 2002.
[40] F. van den Bergh and A. P. Engelbrecht. "A cooperative approach to particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 225–239, Apr. 2004.
[41] H. Wang, D. Wang, and S. Yang, "Triggered memory-based swarm optimization in dynamic environments," in *Proc. EvoWorkshops: Appl. Evol. Comput.*, LNCS 4448. 2007, pp. 637–646.
[42] S. Yang, "Associative memory scheme for genetic algorithms in dynamic environments," in *Proc. EvoWorkshops: Appl. Evol.Comput.*, LNCS 3907. 2006, pp. 788–799.
[43] S. Yang, "Genetic algorithms with memory and elitism-based immigrants in dynamic environments," *Evol. Comput.*, vol. 16, no. 3, pp. 385–416, 2008.
[44] S. Yang, Y. S. Ong, and Y. Jin, Eds., *Evolutionary Computation in Dynamic and Uncertain Environments*. Berlin, Germany: Springer, 2007.
[45] S. Yang and H. Richter, "Hyper-learning for population-based incremental learning in dynamic environments," in *Proc. Congr. Evol. Comput.*, 2009, pp. 682–689.
[46] S. Yang and R. Tinos, "Hyper-selection in dynamic environments," in *Proc. Congr. Evol. Comput.*, 2008, pp. 3185–3192.
[47] S. Yang and X. Yao, "Experimental study on population-based incremental learning algorithms for dynamic optimization problems," *Soft Comput.*, vol. 9, no. 11, pp. 815–834, Nov. 2005.
[48] S. Yang and X. Yao, "Population-based incremental learning with associative memory for dynamic environments," *IEEE Trans. Evol. Comput.*, vol. 12, no. 5, pp. 542–561, Oct. 2008.

Shengxiang Yang (M'00) received the B.S. and M.S. degrees in automatic control, and the Ph.D. degree in systems engineering from Northeastern University, Shenyang, China, in 1993, 1996, and 1999, respectively.

From 1999 to 2000, he was a Post-Doctoral Research Associate with the Algorithm Design Group, Department of Computer Science, King's College London, London, U.K. From 2000 to 2010, he was a Lecturer with the Department of Computer Science, University of Leicester, Leicester, U.K. He is currently a Senior Lecturer with the Department of Information Systems and Computing, Brunel University, U.K. He has over 100 publications. His current research interests include evolutionary algorithms, swarm intelligence, meta-heuristics and hyper-heuristics, artificial neural networks, computational intelligence in dynamic and uncertain environments, scheduling, network flow problems and algorithms, and real-world applications.

Dr. Yang is a member of the Association of Computing Machinery Special Interest Group on Genetic and Evolutionary Computation. He is a member of the Task Force on Evolutionary Computation in Dynamic and Uncertain Environments, Evolutionary Computation Technical Committee, IEEE Computational Intelligence Society. He has given invited keynote speeches in several international conferences and co-organized several workshops and special sessions in conferences. He serves as the Area Editor, an Associate Editor, or an Editorial Board Member for four international journals. He has co-edited several books and conference proceedings and co-guest-edited several journal special issues.

Changhe Li received the B.S. and M.S. degrees in computer science from China University of Geosciences, Wuhan, China, in 2005 and 2008, respectively. He is currently working toward the Ph.D. degree from the Department of Computer Science, University of Leicester, Leicester, U.K.

His current research interests are evolutionary algorithms, particle swarm optimization, and dynamic optimization problems.