A General Framework of Multi-Population Methods with Clustering in Undetectable Dynamic Environments

Changhe Li and Shengxiang Yang, *Member, IEEE*

Abstract— To solve dynamic optimization problems, multiple population methods are used to enhance the population diversity for an algorithm with the aim of maintaining multiple populations in different sub-areas in the fitness landscape. Many experimental studies have shown that locating and tracking multiple relatively good optima rather than a single global optimum is an effective idea in dynamic environments. However, several challenges need to be addressed when multi-population methods are applied, e.g., how to create multiple populations, how to maintain them in different sub-areas, and how to deal with the situation where changes can not be detected or predicted. To address these issues, this paper investigates a hierarchical clustering method to locate and track multiple optima for dynamic optimization problems. To deal with undetectable dynamic environments, this paper applies the random immigrants method without change detection based on a mechanism that can automatically reduce redundant individuals in the search space throughout the run. These methods are implemented into several research areas, including particle swarm optimization, genetic algorithm, and differential evolution. An experimental study is conducted based on the moving peaks benchmark to test the performance with several other algorithms from the literature. The experimental results show the efficiency of the clustering method for locating and tracking multiple optima in comparison with other algorithms based on multi-population methods on the moving peaks benchmark.

Index Terms— Clustering, dynamic optimization problem, undetectable dynamism, multiple population methods, particle swarm optimization, genetic algorithm, differential evolution.

I. INTRODUCTION

GENERALLY speaking, to solve dynamic optimization problems (DOPs) where changes occur over time, it requires the optimization algorithm to not only find the global optimal solution under a specific environment but also continuously track the changing optima over different environments during the search process. Recently, investigating evolutionary algorithms (EAs) for DOPs has attracted many researchers 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 in traditional EAs to address DOPs, including diversity increasing and maintaining schemes [18], [25], [77], memory schemes [9], [76], [82], multi-population schemes [10], [81], adaptive schemes [45], [52], [79], [80], multi-objective optimization methods [14], hybrid approaches [46], [47], change prediction methods [63], and problem change detection approaches [57].

Many experimental studies have shown that locating and tracking a set of optima (the global optimal and near-global optimal solutions) rather than a single global optimum is an effective idea to solve DOPs [6], [44], [54], [78]. However, it is difficult for an algorithm to accurately locate the global optimum in a specific environment and it is even more difficult to track the changing global optimum in different environments. In order to effectively solve DOPs, one solution is to locate and track a set of good optima. This will greatly increase the chance of finding the global optimum by the assumption that one of the near-global optima in the current environment has a larger chance than those bad optima to be the new global optimum in the next environment.

From the literature for DOPs, many algorithms have been proposed to address DOPs using the multi-population method [5], [6], [33], [43], [53], [78], [81], which seems an ideal technique to serve the purpose of locating and tracking multiple optima in dynamic environments. The traditional approaches, which use the multi-population method to find optima for multi-modal functions, divide the whole search space into different sub-spaces, each of which may cover one or a small number of local optima, and then separately search within these sub-spaces. However, one challenging issue of using the multi-population method is that of how to create an appropriate number of sub-populations with an appropriate number of individuals to cover different sub-areas in the fitness landscape. In order to answer this question, a clustering particle swarm optimizer (CPSO) was proposed in [40], [78]. In CPSO, a hierarchical clustering method is employed to automatically create a proper number of sub-populations in different sub-areas.

So far, most algorithms proposed for DOPs are informed when a change occurs or use some techniques to detect changes. However, it is difficult or impossible to detect changes in some cases. For example, it will be very hard to detect changes if only some random local sub-areas change over time in the entire search space. In this case, we can not always successfully detect the changes or predict the changes because we do not know when or where the changes occur in the search space. Therefore, it is important to develop

C. Li is with the School of Computer Science, China University of Geosciences, Wuhan 430074, China (email: changhe.lw@gmail.com).
S. Yang is with the Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex UB8 3PH, U. K. (email: shengxiang.yang@brunel.ac.uk).

algorithms that do not need to detect changes or are able to be ready for changes in any time during the search progress.

In order to effectively use the multi-population method in undetectable environments, this paper proposes a basic framework that is not based on change detection. Several issues are discussed in this paper, e.g., how to create sub-populations, how to deal with the overcrowding problem, how to make sub-populations ready for changes, and how to apply this framework in general EAs for problems in different domains. The basic framework is instantiated into three different research areas, including particle swarm optimization (PSO) [20], [34], genetic algorithm (GA), and differential evolution (DE) [65]. The corresponding algorithms are called CPSOR, CGAR, and CDER respectively. To indicate the clustering algorithms without change detection, we put a suffix "R" for each clustering algorithm without change detection. For example, CPSOR and CPSO represent the clustering PSO algorithm without change detection and the clustering PSO algorithm with change detection, respectively.

This paper carries out a comprehensive experimental study based on the moving peaks benchmark (MPB) problem [9]. In order to test whether the clustering method benefits general research areas or not, we systemically compare the performance of CPSO, CGA, CDE with the corresponding algorithms that use traditional multi-population methods. We also compare the performance of CPSOR, CGAR, and CDER with CPSO, CGA, and CDE, respectively, to investigate whether the strategy that does not have change detection works or not. This paper also compares the performance of CPSOR, CGAR, and CDER with a set of algorithms that were developed for DOPs using multi-population methods in the literature.

The rest of this paper is organized as follows. Sect. II reviews some multi-population methods developed in both stationary and dynamic environments. The basic framework for multi-population methods with clustering is introduced in Sect. III. Sect. IV instantiates the basic framework into particle swarm optimization, genetic algorithm, and differential evolution. The experimental study regarding the configuration, working mechanism, and comparison with other algorithms is presented in Sect. V. Application domains of the proposed framework in the real world are discussed in Sect. VI. Finally, conclusions and discussions on the future work are given in Sect. VII.

II. Multi-population Methods

Many experimental studies have shown that multi-population methods are effective approaches to enhancing the diversity for EAs to solve multi-modal problems in both static and dynamic environments.

A. Multi-population in Static Environments

Kennedy [33] 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-means algorithm iterates three times. The limitation of this clustering approach lies in that the number of clusters, which is problem dependent, must be predefined.

Brits et al. [12] proposed a *nbest PSO* algorithm which is 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 [11], a niching PSO (NichePSO) was proposed by incorporating a cognitive only PSO model and the guaranteed convergence PSO (GCPSO) algorithm [72]. NichePSO maintains a main swarm that can create a sub-swarm once a niche is identified. The main swarm is trained by the cognition only model [32]. If a particle's fitness shows a little change over a small number of generations, then a new sub-swarm is created with the particle and its closest neighbors. NichePSO uses some rules to decide the absorption of particles into a sub-swarm and the merging operation between two sub-swarms, which mainly depends on the radius of the involved sub-swarms.

Parrott and Li developed a speciation based PSO (SPSO) [43], [53], 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 [54]. In [3], Bird and Li developed an adaptive niching PSO (ANPSO) algorithm which adaptively determines the radius of a species by using the population statistics. Based on their previous work, Bird and Li introduced another improved version of SPSO using a least squares regression (rSPSO) in [4]. Recently, in order to determine niche boundaries, a vector-based PSO [62] was proposed to locate and maintain niches by using additional vector operations.

To specify the number of clusters within the k-means PSO algorithm, Passaro and Starita [55] 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 [43] and ANPSO [3] for static problems.

Recently, a new GA, called CardiffGA (CGA), was proposed in [17]. In CGA, each individual is given a life-span and an age, and the population size is also allowed to change. The interesting work in CGA is that two populations, which are named two-human populations, are designed to simulate the competition in species in real life. The two-human populations undergo the competition of a shared resource, called "water". An experimental study showed that the two-human CGA found the solution in a shorter time compared with the single population CGA, but with lower success rate than the single population CGA.

B. Multi-population in Dynamic Environments

Branke et al. proposed a self-organizing scouts (SOS) [10] algorithm that has been shown to give promising results on DOPs with many peaks. In SOS, the whole 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.

Another term of "multi-population", called "multi-nation", was introduced in [71], where multi-national GAs were described for multi-modal problems in dynamic environments. The basic idea of multi-national GAs in dynamic environments is to maintain multiple populations in different search areas in the search space so that the algorithm can search for both local and global optima. A valley detection method was introduced in order to identify different "nations" in the fitness landscape.

The atomic swarm approach has been adapted to track multiple optima simultaneously with multiple swarms in dynamic environments by Blackwell and Branke [5], [6]. 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 surrounds a single peak. In the algorithm, called mQSO in [6], anti-convergence is introduced to detect new peaks by sharing information among all sub-swarms. This strategy was experimentally shown to be efficient for the MPB function [9]. Borrowing the idea of exclusion from [5], Mendes and Mohais developed a multi-population DE algorithm [49] to solve the MPB problem. In their approach, a dynamic strategy for the mutation factor F and probability factor CR in DE was introduced. Recently, an enhanced version of mQSO was proposed by applying two heuristic rules to further enhance the diversity of mQSO in [19]. One of the two rules is to increase the number of quantum particles and to decrease the number of trajectory particles when a change occurs. The other rule is to re-initialize or pause the swarms that have bad performance.

A collaborative evolutionary swarm optimization (CESO) was proposed in [46]. In CESO, two swarms, which use the crowding differential evolution (CDE) [68] and the 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 [46]. Thereafter, a similar algorithm, called evolutionary swarm cooperative algorithm (ESCA), was proposed in [47] based on the collaboration between a PSO algorithm and an EA. In ESCA, three populations using different EAs are used. Two of them follow the rules of CDE [68] to maintain the diversity. The third population uses the rules of PSO. Three types of collaborative mechanism are also developed to transmit information among the three populations.

Inspired by the SOS algorithm [10], a fast multi-swarm optimization (FMSO) algorithm was proposed in [38] 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 sub-spaces. 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 than their radius, then the whole swarm of the worse one is removed. This guarantees that no more than one child swarm will cover a single peak. Another similar idea of hibernation multi-swarm optimization algorithm (HmSO) was introduced in [29], where a child swarm will hibernate if it is not productive anymore and will be woken up if an environmental change has been detected.

A clustering PSO (CPSO) has recently been proposed for DOPs in [40], [78]. CPSO applies a hierarchical clustering method to divide an initial swarm into sub-swarms that cover different local regions. CPSO was proposed to attempt to solve some challenging issues when applying multi-population methods, e.g., how to guide particles to move toward different promising sub-regions, how to define the area of each sub-merging, and how to determine the number of sub-swarms needed. CPSO has shown some promising results compared with several state-of-the-art algorithms in [78].

III. General Framework of Multi-population Methods with Clustering for DOPs

So far, most EAs developed for DOPs either use some change detection methods [6], [43], [40], [46], [47], [57], [78] or predict changes by assuming that changes have a pattern [63]. Once a change has been detected or predicted, different kinds of strategies are applied to increase the diversity, e.g., random immigrants strategies, or to re-use stored useful information by assuming that the new environment is closely related to the current environment, e.g., memory-based strategies. However, to use those strategies, a condition must be applied. That is, the environmental changes must be successfully detected. So, here comes a common question: What can these algorithms do if they fail to detect the changes?

Maintaining diversity without change detection throughout the run is an interesting topic. In [25], random individuals are created every iteration. Three different mutation strategies were designed to control the diversity in [18]. Sharing or crowding mechanisms in [16] were introduced to ensure diversity. The thermodynamical genetic algorithm (TDGA) [50] was proposed to control the diversity explicitly via a measure, called "free energy". However, these methods are not effective because the continuous focus on diversity slows down the optimization process as pointed out in [27] and hence little research on maintaining diversity without change detection has been carried out in the literature. Therefore, far more effective algorithms are still needed.

A. Undetectable Dynamic Environments

It is important and meaningful to build effective mechanisms into EAs for DOPs that do not need to detect changes. This is because sometimes it is hard or impossible for algorithms to detect changes. For example, if there are only some random sub-areas in the whole search space that change, it will
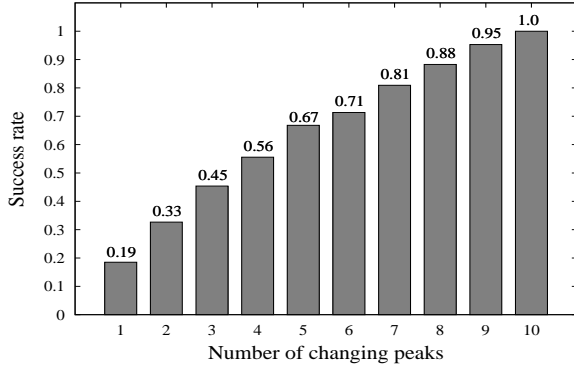
Fig. 1. Successful detection rate with different numbers of changing peaks in the MPB problem with a total number of ten peaks.

become very difficult for algorithms to detect the changes. If an algorithm fails to detect the changes, all the traditional corresponding strategies will lose their functions.

In order to illustrate that such kind of dynamic environments exist, we add a new feature into the MPB problem (it will be described in detail in Sect. V-A.1): Only some random peaks, rather than all peaks in the original MPB problem [9], are allowed to change during one environmental change. This feature will make the changes difficult to detect. To show the difficulty of detecting changes in the modified MPB problem, the following experiments were conducted, where 10000 uniformly distributed random points were sampled in the fitness landscape with a total number of ten peaks in five dimensions. We record the total number of points (suc) that successfully detect the changes over 1000 changes. The average rate of successfully detecting changes can be estimated by $suc/10^7$. A detection by a point is successful if the fitness of that point is different from its previous fitness. Fig. 1 shows the rate of successful detection with different numbers of changing peaks in the MPB problem with a total number of ten peaks.

Fig. 1 clearly shows that the successful detection rate linearly decreases when the number of changing peaks decreases. The smaller the number of changing peaks, the harder it will be to detect the change. It can be seen that the successful detection rate is less than 0.2 when only one out of ten peaks changes in the search space. And the figure increases to 1 when all peaks change, i.e., any one point in the fitness landscape can detect the changes. It means any single individual is enough for an EA to successfully detect the changes in the MPB problem when all peaks change, which is a default setting of the MPB problem used by all EAs so far.

The dynamic knapsack problem [31], [39] is another example for dynamic environments where changes are difficult to detect. In the dynamic knapsack problem, detecting a change is not always as straightforward as it may seem. Normally, monitoring a possible change in fitness value is used to detect the change but this does not always work. For example, you will detect a change by monitoring the fitness of an individual over two successive iterations, if the maximum allowed capacity for a knapsack problem is reduced. But, on the other hand, if the allowed capacity is increased, it will be likely to go undetected. This is because the capacity change does not affect the current population, which is still in the feasible areas in the search space.

The recurrent change with noise in [41] is one example of dynamic environments where the changes are completely undetectable. Because of noise, every evaluation for a similar solution is different. Therefore, in this kind of dynamic environment, the changes are never detected by checking whether there is a change in the fitness of the same individual in the fitness landscape.

In the real world, there are also many DOPs where the changes are hard-to-detect or undetectable. For example, in the design of the path planning system, it is difficult for mobile robots to detect the environmental changes due to moving obstacles with unknown trajectories. In the dynamic scheduling problem, it will be hard to detect a constraint change if the change does not affect current solutions. For the multi-rover coordination problem in noisy environments, the signals that rovers receive from the environment are not reliable due to the noise generated by the rovers' sensors.

The above cases show that the design of approaches without change detection is very necessary to solve optimization problems in dynamic environments where the changes are difficult to detect. This is because traditional EAs based on change detection will fail to work in such kind of dynamic environments due to the difficulties of detecting the changes. For example, change detection methods will not work in dynamic environments with noise due to the disturbance caused by noise. As a result, the performance of these algorithms will significantly deteriorate.

B. Clustering Methods to Create Multiple Populations

The methods of creating multiple populations can be roughly classified into three different categories in terms of the *way to generate multiple populations*. The first class of multi-population methods simply uses a certain number of randomly generated populations. Usually, all populations use the same search strategy with the same number of individuals. However, in some of these kinds of algorithms, the populations are assigned into different groups where each group uses different search methods (e.g., ESCA [47], CESO [46], and mQSO [6]) to serve different purposes (e.g., exploring new promising sub-areas or exploiting local optima). The second class of multi-population methods starts from a main population and maintains it to generate sub-populations by splitting off from the main population (e.g., SOS [10], NichePSO [11], FMSO [38], and HmSO [29]) if some predefined criteria are satisfied (e.g., the best individual in the main population does not improve for a certain number of iterations). The third class of multi-population methods divides a large randomly generated population into several small sub-populations to make them cover different sub-areas in the search space (e.g., the k-means PSO [33], SPSO [43], $nbest$ PSO [12], and CPSO [40], [78]).

The major common problem of these multi-population methods is that, although they benefit the algorithms, they also bring new issues which are difficult to solve. For example,

Algorithm 1 $Clustering(pop)$

1: Create a temporary cluster list G of size $|pop|$;
2: **for** each individual i in pop **do**
3:    $G[i] := pop[i]$; {i.e., each individual forms one cluster in G}
4: **end for**
5: Calculate the distance between all clusters (i.e., individuals) in G and construct a distance matrix **M** of size $|G| \times |G|$;
6: **while** $TRUE$ **do**
7:    **if** $!FindNearestPair(G, t, s)$ **then**
8:       Break;
9:    **end if**
10:    $t := t + s$; {i.e., merge clusters t and s into t}
11:    Delete the cluster s from G;
12:    Re-calculate all distances in **M** which have been affected by the merge of t and s;
13:    **if** each cluster in G has more than one individual **then**
14:       Break;
15:    **end if**
16: **end while**
17: Remove pop;
18: $plst := plst + G$;

Algorithm 2 $FindNearestPair(G, t, s)$

1: $found := FALSE$;
2: $min\_dist := \sqrt{\sum_{i=1}^{D}(U_i - L_i)^2}$, where U_i and L_i are the upper and lower bounds of the i-th dimension of the search space;
3: **for** $i := 0$ to $|G|$ **do**
4:    **for** $j := i + 1$ to $|G|$ **do**
5:       **if** $(|G[i]| + |G[j]| > subSize)$ **then**
6:          continue;
7:       **end if**
8:       **if** $(min\_dist > M(G[i], G[j]))$ **then**
9:          $min\_dist := M(G[i], G[j])$;
10:          $t := G[i]$;
11:          $s := G[j]$;
12:          $found := TRUE$;
13:       **end if**
14:    **end for**
15: **end for**
16: Return $found$;

---

the optimal value of k in the k-means PSO is unknown and problem dependent; the optimal number of sub-populations is unknown for the mQSO algorithm; there is no overlapping control among populations in ESCA and CESO; it is difficult to identify the proper representative individuals in SPSO, NichePSO, SOS, FMSO, HmSO, and $nbest$ PSO; and it is very difficult to define an appropriate radius for each population in SPSO, mQSO, FMSO, and HmSO.

Among these methods, CPSO seems a competitive one as it alleviates the common problems that other methods suffer. Different from other methods, CPSO uses a single linkage hierarchical clustering method [28], as shown in Algorithm 1, to create sub-populations. The clustering method can enable CPSO to assign individuals to different promising sub-regions, adaptively adjust the number of sub-populations needed, and automatically calculate the search region for each sub-population.

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

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

The distance of two clusters t and s in the list of G, which is an element in M in Algorithm 1 and is denoted $M(t, s)$, is defined as the distance of the two closest individuals i and j that belong to clusters t and s, respectively. $M(t, s)$ can be formulated as:

$$M(t, s) = \min_{i \in t, j \in s} d(i, j) \qquad (2)$$

Given an initial population pop, the clustering method works as follows: It first creates a list G of clusters with each cluster

only containing one individual in pop. Then, in each iteration, it uses Algorithm 2 to find a pair of clusters t and s such that they are the closest among those pairs of clusters, of which the total number of individuals in the two clusters is not greater than $subSize$ ($subSize$ is a prefixed maximum sub-population size), and, if successful, combines t and s into one cluster. This iteration continues until all clusters in G contain more than one individual. Finally, the cluster list G is appended to a global sub-population list $plst$.

From the above description, it can be seen that using the above clustering method, sub-populations will be automatically created with close individuals depending on the distribution of initial individuals in the fitness landscape. The number of sub-populations and the size of the search area of each sub-population are also automatically determined by the fitness landscape and the unique parameter $subSize$.

C. Redundancy Control

Redundancy is a very important factor that will affect the performance of an algorithm. Here, redundancy control is to remove redundant individuals, including the converged or near converged individuals, the overcrowded individuals in a local sub-area, and the individuals that are located in the overlapping areas of two conjunction populations.

It is important to perform the operation of redundancy control. First of all, redundant individuals normally do not contribute much to the search progress. Taking the converged or near converged individuals as an example, they are inactive and almost dying so they hardly contribute to the search. Secondly, we can save computing resources and give the saved resources to the *useful* individuals. Thirdly, removal of redundant individuals is a preparation phase in order to increase the diversity in this paper (the phase of maintaining diversity will be discussed later in Sect. III-D).

Traditionally, the overlapping check between two populations is carried out using their search radius. The search radius

of a sub-population s can be calculated as follows:

$$radius(s) = \frac{1}{|s|} \sum_{i \in s} d(i, s_{center}), \qquad (3)$$

where s_{center} is the central position of the sub-population s and $|s|$ is the number of individuals in s. If any individual in a sub-population is within the search radius of another sub-population, then the overlapping search occurs. If the distance of the best individuals of two sub-populations is less than their search radius, then they are combined or one of them is removed. The above checking mechanism assumes that each sub-population just covers one peak. However, it is not always true for the real situation. If a sub-population in a sub-region covers more than one peak, other sub-populations that are within its search area should not be removed or combined together with this sub-population; otherwise, we will lose the peaks which are currently being searched due to the combination. Therefore, we should take this issue into account before combining two overlapping populations.

To address the above issues, when applying the merge operation, we adopt the following overlapping checking scheme. If two sub-populations t and s are within each other's search area, an overlapping ratio between them, denoted $r_{overlap}(t, s)$, is calculated as follows: We first calculate the percentage of individuals in t which are within the search area of s and the percentage of individuals in s which are within the search area of t, and then set $r_{overlap}(t, s)$ to the smaller one of these two percentages. The two sub-populations t and s are combined only when $r_{overlap}(t, s)$ is greater than a threshold value β.

It should be noted that the radius of s and t used in the overlapping check operation is their initial radius when s and t are first created by the clustering method rather than their current radius. Usually, the radius of a sub-population will decrease with the evolutionary process as the sub-population will gradually converge. Therefore, the initial radius obtained when a sub-population is formed is usually larger than its current radius. Therefore, using the initial radius rather than the current radius of sub-populations, we can identify an overcrowded area as early as possible to save computational resources.

In order to avoid too many individuals searching on a single peak and hence save computing resources, an overcrowding check is performed on each sub-population after the above overlapping check. If the number of individuals in a sub-population is greater than $subSize$, then the worst individuals are removed one by one until the size of the sub-population is equal to $subSize$.

Individuals that have already converged, are also redundant and should be removed. If the radius of a sub-population is less than a small threshold value ϵ, which is set to 0.01 in this paper, the sub-population is regarded to be converged on a peak. If a sub-population is converged, it will be removed from the sub-population list $plst$. The convergence check is carried out after the overcrowding check. The procedure of redundancy control is as shown in Algorithm 3.

Algorithm 3 $Remove(plst)$

1: **for** each pair of sub-populations (t, s) in $plst$ **do**
2:     **if** $r_{overlap}(t, s) > \beta$ **then**
3:         Merge t and s into t;
4:         Remove s from $plst$;
5:     **end if**
6: **end for**
7: **for** each sub-population $t \in plst$ **do**
8:     **if** $|t| > subSize$ **then**
9:         Remove worst $(|t| - subSize)$ individuals from t;
10:     **end if**
11: **end for**
12: **for** each sub-population $s \in plst$ **do**
13:     **if** $radius(s) < \epsilon$ **then**
14:         Remove s from $plst$;
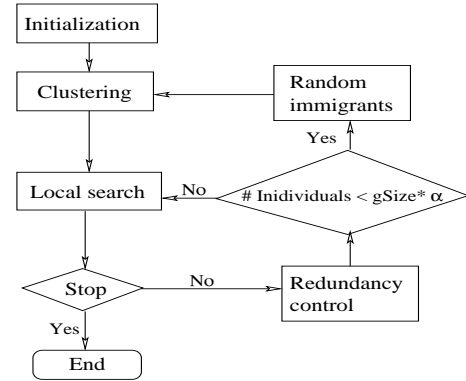15:   **end if**
16: **end for**

Fig. 2. Framework for multi-population methods with clustering in undetectable dynamic environments.

D. Maintaining Diversity

Based on the above description, it is easy to figure out how to maintain the diversity. We do not increase the diversity every iteration as traditional methods do. Instead, we only increase the diversity if the population diversity decreases to a certain level. In this paper, we use a simple metric to measure the population diversity: $\#s\_indis(t)/gSize$, where $\#s\_indis(t)$ is the number of survived individuals at iteration t and $gSize$ is the size of the initial population. If the population diversity decreases to a constant threshold (α), then we apply a random immigrants method where a temporal population of the size $gSize - \#s\_indis(t)$ is randomly generated. Thereafter, we cluster the temporal population by the above clustering method and append the new sub-populations to $plst$.

Thanks to the clustering method and the redundancy control mechanism, we can easily implement an algorithm for DOPs using any population-based EA. Fig. 2 describes the whole general framework for multi-population methods using the clustering method in dynamic environments. The step of local search in Fig. 2 is the optimization process which should be replaced by a specific EA (e.g., PSO, GA, or DE).

Using this framework to solve DOPs has several advantages.

Firstly, it can be used in dynamic environments with any properties, e.g., mild change, severe change, cyclic change, chaotic change, or even undetectable changes. From Fig. 2, it can be seen that the whole process has nothing to do with the environmental changes. When to increase the diversity only depends on the information gathered from the current populations. Secondly, it can be implemented by any EA. Thirdly, it is simple to implement. Sect. IV will present some instances implemented from this framework for DOPs in several different research areas.

E. Complexity Analysis

In the framework, compared with a traditional EA, we need to perform some extra operations, including the clustering process and the overcrowding check among sub-populations. From Algorithm 1, it can be seen that the time complexity of the clustering operation is $O(gSize^3)$. We first compute all distances among each pair of clusters in $O(gSize^2)$. For each iteration of merging two clusters t and s from the cluster list G, we find the nearest pair of clusters of G in $O(|G|^2)$ (if we use a 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 decrease by one every iteration until the stop criteria is met. Finally, we perform the clustering operation in $O(gSize^3)$. It should be noted that the clustering operation is not performed every iteration during the search process, instead it is triggered only when the total number of individuals is less than $\alpha \cdot gSize$ (see Fig.2 for the framework). In addition, after the first clustering operation, only $(1 - \alpha) \cdot gSize$ individuals will be involved in the following clustering operations in the whole run.

The time complexity of the overcrowding check, which is mainly on the calculation of the overlapping ratio between two sub-populations (see Algorithm 3), depends on how many sub-populations have survived. For each pair of sub-populations, the calculation of the overlapping ratio is done in $O(subSize)$, which is a constant time ($subSize = 7$ is suggested in this paper, see Sect. V-B.1 for details). Furthermore, the number of sub-populations will decrease before the next clustering operation is performed during the search process.

In total, according to the above component complexity analysis, it can be seen that the extra computing time needed for the framework is not so high.

IV. INSTANTIATION OF THE FRAMEWORK

In this section, we instantiate the framework of multi-population with clustering into three different research areas: PSO, GA, and DE. For the convenience of description, the corresponding algorithms are denoted CPSOR, CGAR, and CDER, respectively, in this paper. We also discuss how to apply this framework for DOPs in combinatorial space in the end of this section.

A. Clustering with PSO

PSO was first introduced by Kennedy and Eberhart in [20], [34]. Ever since PSO was first introduced, several major

---

Algorithm 4 $gbestLearn(particle\ \vec{x}_i)$

1: **for** each dimension d of $gbest$ **do**
2:     $\vec{x}_{t\_gbest} := \vec{x}_{gbest}$ \{$\vec{x}_{t\_gbest}$ is a temporary particle\};
3:     $x_{t\_gbest}[d] := x_i[d]$;
4:     **if** $\vec{x}_{t\_gbest}$ is better than $\vec{x}_{gbest}$ **then**
5:         $x_{gbest}[d] := x_{t\_gbest}[d]$;
6:     **end if**
7: **end for**

---

versions of the PSO algorithm have been developed [56]. In PSO, each particle i, which is a candidate solution, 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 [75] as follows:

$$v'^d_i = \omega v^d_i + \eta_1 r_1(x^d_{pbest_i} - x^d_i) + \eta_2 r_2(x^d_{gbest} - x^d_i) \quad (4)$$

$$x'^d_i = x^d_i + v'^d_i, \quad (5)$$

where x'^d_i and x^d_i represent the current and previous position in the d-th 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, $\omega \in (0, 1)$, η_1, and η_2 are constant parameters, and r_1 and r_2 are random numbers generated in the interval $[0.0, 1.0]$ uniformly.

In order to speed up the local search within the PSO algorithm, we introduce a learning method for the $gbest$ particle used in CPSO [78]. This learning method tries to extract useful information relevant to those potentially improved dimensions of an improved particle to update $gbest$, as shown in Algorithm 4. When a particle i in a sub-population finds a better position, we iteratively check each dimension of the $gbest$ particle: replace the dimension with the corresponding dimensional value of particle i if the $gbest$ particle is improved by doing so. In this way, the $gbest$ particle is able to learn the useful information from those dimensions of a particle that has been improved.

The PSO algorithm with the $gbest$ model is used in the CPSOR algorithm where each particle's neighborhood is defined as the whole swarm. The basic PSO algorithm in CPSOR is shown in Algorithm 5, and the whole framework of the CPSOR algorithm is described in Algorithm 6.

B. Clustering with GA

In the CGAR algorithm, we use a simple real-coded GA as described in Algorithm 7. The crossover and mutation operators used in CGAR are the arithmetic crossover and the normal mutation operator, respectively. They are described as follows.

For two individuals $\vec{x}_i$ and $\vec{x}_j$, the arithmetic crossover operator is performed on the d-th dimension as follows:

$$x^d_i = x^d_i * r + x^d_j * (1 - r) \quad (6)$$

$$x^d_j = x^d_j * r + x^d_i * (1 - r) \quad (7)$$

Algorithm 5 PSO

1: **for** each particle $\vec{x}_i$ **do**
2:     $\vec{x}_t := \vec{x}_i$; {$\vec{x}_t$ is a temporal particle}
3:     Update particle i according to Eqs. (4) and (5);
4:     **if** $f(\vec{x}_i) < f(\vec{x}_{pbest_i})$ **then**
5:        $\vec{x}_{pbest_i} := \vec{x}_i$;
6:        **if** $f(\vec{x}_i) < f(\vec{x}_{gbest})$ **then**
7:           $\vec{x}_{gbest} := \vec{x}_i$;
8:        **end if**
9:        **if** $f(\vec{x}_i) < f(\vec{x}_t)$ **then**
10:        $gbestLearn(\vec{x}_i)$;
11:        **end if**
12:     **end if**
13: **end for**

Algorithm 6 CPSOR

1: Create an initial population pop with $gSize$ particles by randomly generating the position and velocity for each particle;
2: Create an empty list $plst$ to store sub-populations;
3: $Clustering(pop)$;
4: **while** stop criteria is not satisfied **do**
5:     **for** each sub-population $plst[i]$ **do**
6:        $plst[i].PSO()$;
7:     **end for**
8:     $Remove(plst)$;
9:     Count the number of survived individuals s_indis;
10:    **if** $s\_indis < gSize \cdot \alpha$ **then**
11:       Create a temporal population t_pop with $(gSize - s\_indis)$ random individuals;
12:       $Clustering(t\_pop)$;
13:    **end if**
14: **end while**

Algorithm 7 GA

1: Select individuals into mating pool by the roulette wheel selection mechanism;
2: **for** each pair of individuals $\vec{x}_i$ and $\vec{x}_{i+1}$ **do**
3:     **if** $r < p_c$ **then**
4:        Perform arithmetic crossover; {p_c is the crossover probability}
5:     **end if**
6: **end for**
7: **for** each individuals $\vec{x}_i$ **do**
8:     Perform the mutation operation;
9: **end for**

where r is a random number uniformly distributed in the interval $(0.0, 1.0)$.

The mutation operation for the d-th dimension of individual $\vec{x}_i$ is performed with a probability of $1/D$ as follows:

$$x_j^d = x_j^d + N(0,1) \tag{8}$$

where $N(0,1)$ is a normally distributed random number with mean 0 and variance 1.

Algorithm 8 DE

1: **for** each individual $\vec{x}_i$ **do**
2:     Generate a donor vector $\vec{v}$ by: $\vec{v} := \vec{x}_{r1} + F \cdot (\vec{x}_{r2} - \vec{x}_{r3})$; {$F$ is mutation factor in [0,2] and $\vec{x}_{r1}$, $\vec{x}_{r2}$, and $\vec{x}_{r3}$ are randomly selected individuals (indices of i, $r1$, $r2$, and $r3$ are distinct)}
3:     Generate a trial vector $\vec{u}$ as follows:
$$u^d := \begin{cases} v^d, & \text{if } r <= CR \text{ or } d = I_{rand} \\ x^d, & \text{if } r > CR \text{ and } d! = I_{rand} \end{cases}$$
    where CR is a probability constant and I_{rand} is a random integer within [1,D].
4:     **if** $f(\vec{u}) < f(\vec{x}_i)$ **then**
5:        $\vec{x}_i := \vec{u}$;
6:     **end if**
7: **end for**

C. Clustering with DE

DE [65] is also a population-based optimization approach, whose main strategy is to generate a new position for an individual by calculating vector differences between other randomly selected members of the population. A simple DE algorithm used in CDER is described in Algorithm 8.

It should be noted that the CPSOR algorithm is different from the CGAR and CDER algorithms where the *pbest* position of each particle does not involve the evaluation process. Therefore, there is an issue of outdated memory of the *pbest* position that will mislead the future search for each particle. In order to solve this problem for the CPSOR algorithm, we re-evaluate a particle's *pbest* position before comparing with its current position every iteration. Fortunately, the issue of outdated memory in CGAR and CDER is not as serious as in CPSOR as each individual will be evaluated every iteration.

So far, we have recognized that we do not need to develop any complex techniques to deal with the dynamism or even to detect the environmental changes. Furthermore, this framework is easy to apply in any EA. The only differences between two algorithms using this framework are the steps of initialization and local search. Due to the similarities between CGAR and CPSOR, the framework of CGAR is not provided in this paper, and neither for the CDER algorithm.

D. The Clustering Framework for Combinatorial DOPs

From the description of the clustering method in Sect. III-B, it is easy to apply the framework for DOPs in the combinatorial space. The only work that we need to do is to re-define a proper distance metric for the particular problem to be solved, i.e., re-calculation of the distance of two individuals in Eq. (1). Then, we can directly use Algorithm 1 and Algorithm 2 to generate multiple populations, Algorithm 3 to reduce the redundancy, and the framework in Fig. 2 to adapt the clustering method into dynamic environments.

The choice of the distance metric depends on the problem to be solved. For example, a proper distance metric for two individuals in binary encoding would be the Hamming distance, that is the number of genes at which the corresponding values are different. For the travelling salesman problem (TSP), the

distance between two individuals would be the number of different edges. A proper distance metric would determine the performance of this framework.

1) Discussion: As we know, poor offspring will be generated by recombination if two parents are in two different peaks. This is because the offspring are highly likely to be in the valley between the two peaks. Therefore, recombination is deleterious to the performance of EAs if the involved parents are in different peaks [64]. We expect that this issue should be relieved with the clustering method. The aim of the clustering method is to divide the whole search space into multiple sub-areas. The result of the clustering method is that similar individuals, which are close to each other and likely to be in one peak in the fitness landscape, are assigned to one cluster. The ideal result after using the clustering method is that each sub-area only contains a single peak. In this ideal case, all parents will be in the same peak, and, hence, recombination will produce promising offspring other than bad ones.

However, regarding the framework for DOPs in the combinatorial space, our major concern is the effectiveness of the clustering method to locate multiple sub-areas in the search space. The original motivation of the clustering method is for multi-modal problems in the continuous space. There is clear understanding of the fitness landscape for most problems in the continuous space, i.e., a typical fitness landscape is composed of multiple peaks that are distributed in different areas in the fitness landscape. Therefore, we can divide the whole fitness landscape into multiple sub-areas, then search and track them simultaneously. However, there is no such clear indication for many combinatorial problems regarding what the fitness landscape is, e.g., the TSP, and how to define the distance between two individuals. Although we can easily extend this framework for DOPs into combinatorial problems, it lacks clear explanatory support from the motivation point of view. Therefore, we will not implement this framework for combinatorial problems in this paper.

2) Possible Combinatorial DOP Generator: Although we do not test the performance of this framework for combinatorial problems in this paper, it is an interesting topic. If users wish to extend this framework into the combinatorial space, we suggest a possible combinatorial DOP generator that is able to create multi-modal problems (it surely can be applied in any combinatorial problem). To generate a fitness landscape with P random peaks [64], we can randomly generate P binary strings with length L, which represent the locations of the P peaks in the fitness landscape. To evaluate the fitness of an arbitrary individual x, we first locate the nearest peak by the Hamming distance, then the fitness of individual x is calculated as the number of bits that are in common with that nearest peak, divided by L, as follows:

$$f(x) = \frac{1}{L} \max_{p=1,\cdots,P} (L - d(x, Peak_p)) \qquad (9)$$

where $d(x, Peak_p)$ is the Hamming distance between individual x and the p-th peak $Peak_p$.

Interestingly, we can also easily extend this problem to dynamic environments with characteristics as in the MPB problem [9], where the fitness landscape has a certain number of peaks, whose locations and heights are able to change overtime. To achieve this, we first need to make the height of each peak changeable. For the above expression, it implies that all the P peaks have an equal height of 1. However, we can assign a changing weight ($H_p(t)$) to each peak at time t, which enables the peaks to have different heights over time. Therefore, a DOP can be obtained as follows:

$$f(x,t) = \frac{1}{L} \max_{p=1,\cdots,P} H_p(t)(L - d(x, Peak_p(t))) \qquad (10)$$

It should be noted that the above problem generator is just an example of the kind of problem the clustering framework is capable of solving in the combinatorial space. If we are able to fully understand the fitness landscape for a given problem with multiple optima, i.e., the distance metric is properly defined, then, we believe, the framework will be applicable to that problem.

V. EXPERIMENTAL STUDY

In this section, four groups of experiments are carried out based on the MPB problem [9]. The objective of the first group of experiments is to investigate the working mechanism of the framework through the CPSOR algorithm, analyze the sensitivity of key parameters, and suggest some methods to set up the parameters, including the overlapping ratio β, the diversity degree α, the global population size $gSize$, and the sub-population size $subSize$. The objective of the second group of experiments is to investigate whether EAs benefit or not from using the framework of multi-population with clustering to solve DOPs. In the third group of experiments, the performance of CPSOR, CGAR, and CDER is compared with a set of EAs taken from the literature for DOPs. The involved algorithms include mCPSO [6], mQSO [6], SPSO [54], rSPSO [4], CESO [46], ESCA [47], CPSO [78], PSO-CP [45], HmSO [29], and RVDEA [73].

The RVDEA [73] is a dynamic EA that uses variable relocation to adapt already converged or currently evolving individuals to the new environmental condition. In this paper, we used the version of RVDEA with clusters [73], whose performance is better than that of the version of RVDEA with memory [73] on the MPB problem. In the version of RVDEA with clusters, multiple clusters (populations) are used to guide the selection and replacement procedures with the aim of exploring different sub-areas in the search space.

All the results of the peer algorithms shown in this paper are provided in the papers where they were proposed except the CPSO algorithm, whose results are updated in this paper. Finally, we present the comparison between CPSOR and CPSO in dynamic environments where changes are difficult to detect.

A. Experimental Setup

1) The Moving Peaks Benchmark (MPB) Problem: The MPB problem, proposed by Branke [9], 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:

$$F(\vec{x}, t) = \max_{i=1,\ldots,p} \frac{H_i(t)}{1 + W_i(t) \sum_{j=1}^{D} (x_j(t) - X_{ij}(t))^2}, \quad (11)$$

where $W_i(t)$ and $H_i(t)$ are the height and width of peak i at time t, respectively, and $X_{ij}(t)$ is the j-th 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:

$$\vec{v}_i(t) = \frac{s}{|\vec{r} + \vec{v}_i(t-1)|}((1-\lambda)\vec{r} + \lambda\vec{v}_i(t-1)), \quad (12)$$

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:

$$H_i(t) = H_i(t-1) + height\_severity * \sigma \quad (13)$$

$$W_i(t) = W_i(t-1) + width\_severity * \sigma \quad (14)$$

$$\vec{X}_i(t) = \vec{X}_i(t)(t-1) + \vec{v}_i(t) \quad (15)$$

where σ is a normal distributed random number with mean 0 and variation 1.

It should be noted that the same setup for the MPB problem [9] – the second scenario – was used for all the involved peers algorithms. i.e., the comparison of all the peer algorithms is under the same configuration of the MPB problem in this paper.

It should also be noted that the choice of the MPB problem does not favor our method. There are several other benchmarks in the literature for DOPs, e.g., the DF1 [51] and the GDBG [41]. The structures of the DF1 [51] problem and the GDBG [41] problem are similar to the MPB problem, which are able to create a user-defined number of peaks with changing heights, widths, and locations. However, they are not as widely used as the MPB problem in the literature. Although the GDBG system is more complex than the MPB and DF1 problems, it is not familiar to many researchers since it was just recently proposed (the experimental results in thesis [36] shown that the clustering idea is also effective to solve the GDBG system in comparison with several other algorithms). The motivation of the design of the MPB problem is to bridge the gap between very complex, hard to understand real-world problems and all too simple toy problems [9]. Although it is not the real-world problem, it has been put forward as representative of real-world dynamic problems [5], [6], [9].

TABLE I

DEFAULT SETTINGS FOR THE MPB PROBLEM, WHERE 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 HEIGHT OF PEAKS IS SHIFTED RANDOMLY IN THE RANGE $H = [30, 70]$ AND THE WIDTH OF PEAKS IS SHIFTED RANDOMLY IN THE RANGE $W = [1, 12]$

Parameter	Value
$peaks$ (number of peaks)	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
percentages of changing peaks $cPeaks$	1.0
S	[0, 100]
H	[30.0, 70.0]
W	[1, 12]
I	50.0

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. It should be noted that different from the traditional MPB problem, the percentages of changing peaks ($cPeaks$), which is a new feature, is added in the MPB problem in this paper. This feature will make the MPB problem harder to solve because many techniques based on change detection may lose their functions. The traditional MPB problem is a special case of the MPB problem used in this paper where $cPeaks = 1.0$.

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

$$\mu = \frac{1}{K} \sum_{k=1}^{K} (h_k - f_k), \quad (16)$$

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, which is set to $K = 100$ in this paper. All the results reported are based on the average over 30 independent runs with different random seeds.

In CPSOR, the acceleration constants η_1 and η_2 were both set to 1.7. The inertia weight ω was set to 0.6. For the CGAR algorithm, the crossover probability (p_c) was set to 0.6. $F=0.5$ and $CR=0.1$ were used in the CDER algorithm. It should be noted that the values of the parameters used in the three algorithms may not be the best choice. However, it is not the main task in this paper to investigate the optimal values of these parameters.

B. Experimental Investigation of CPSOR

To effectively apply the framework to solve DOPs, we first investigate the effect of different configurations and suggest

general settings for the four parameters: the overlapping ratio β, the diversity degree α, the global population size $gSize$, and the sub-population size $subSize$. Then, the working mechanism of the CPSOR algorithm is illustrated based on the MPB problem in two dimensions with different aspects.

Actually, how to setup the four parameters to get the best performance of the framework is a difficult optimization problem because the four parameters may depend on each other, and there are many potential and unknown factors that may affect the choice of the four parameters. Therefore, in order to investigate whether there are general rules to setup the four parameters and how to find out the guidance, we first have an overview of the four parameters.

The overlapping ratio β determines the moment to merge two sub-populations when they overlap each other. The larger the value of β, the longer the time it will take to start the merging process, and vice versa. This combination process will remove redundant individuals to save computational resources and hence speed up the search. Performing this process a bit late or early should not affect too much of the performance. Intuitively, this parameter should not be crucial to the performance of the framework. In addition, it will not depend on the other parameters. Therefore, it can be separately analyzed.

For the global population size $gSize$ and the sub-population size $subSize$, they are apparently two crucial parameters and they directly determine how many sub-populations will be generated in the fitness landscape, which is the most important factor that will affect the performance of the framework. A too small number of sub-populations generated will bring about more than one peak contained in a single sub-population, so the algorithm will not effectively track different optima. On the contrary, if too many sub-populations are distributed in the fitness landscape, too many isolated sub-populations will cover a single peak, which is also not a good option. To effectively locate and track the changing optimum in the fitness landscape with a fixed number of peaks, the number of sub-populations has to be optimized. Therefore, these two parameters will be analyzed together in this section.

In order to solve DOPs without change detection, we need to maintain the population diversity throughout the run by the parameter α. Intuitively, the choice of α would depend on how much diversity is needed. Usually, the larger the number of peaks in the fitness landscape, the larger diversity is needed and hence the larger value of α.

Based on the above analysis regarding the configuration of the four parameters, we carried out the following experiments based on the assumption that β, α, and the group of $gSize$ and $subSize$ are independent of each other. It should be noted that the assumption might not be exactly the fact of their real relationship and there are many other factors related to the problem that would affect the setup of the four parameters, e.g., the number of peaks, the change severity, and the change frequency, etc. The following section will find out how to setup the framework to effectively solve DOPs.

1) Parameter Sensitivity Study: In this section, the four parameters in the basic framework are investigated separately except for $gSize$ and $subSize$, which are bound together to analyze the sensitivity. In order to test one particular

TABLE II
OFFLINE ERRORS AND $\pm$ STANDARD ERROR FOR DIFFERENT VALUES OF β WITH DIFFERENT NUMBERS OF PEAKS, WHERE $\alpha = 0.3$, $gSize = 200$, AND $subSize = 7$, AND THE DEFAULT SETTINGS FOR THE MPB PROBLEM IN TABLE I WERE USED

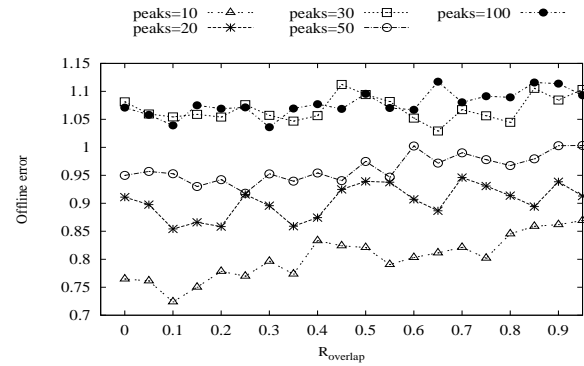| peaks	β=0.1	β=0.2	β=0.3	β=0.4	β=0.5	β=0.6	β=0.7	β=0.8	β=0.9
10	0.75	0.77	0.773	0.824	0.791	0.811	0.802	0.859	0.869
	±0.05	±0.05	±0.06	±0.05	±0.05	±0.08	±0.05	±0.07	±0.08
20	0.866	0.916	0.859	0.925	0.938	0.887	0.931	0.894	0.913
	±0.05	±0.06	±0.05	±0.05	±0.06	±0.06	±0.06	±0.06	±0.08
30	1.06	1.08	1.05	1.11	1.08	1.03	1.06	1.11	1.1
	±0.06	±0.05	±0.06	±0.06	±0.05	±0.06	±0.06	±0.06	±0.06
50	0.93	0.918	0.94	0.94	0.947	0.972	0.978	0.979	1
	±0.06	±0.05	±0.06	±0.05	±0.05	±0.06	±0.06	±0.07	±0.07
100	1.08	1.07	1.07	1.07	1.07	1.12	1.09	1.12	1.09
	±0.06	±0.04	±0.05	±0.05	±0.04	±0.05	±0.04	±0.05	±0.05

Fig. 3. Offline errors of varying the values of β with different numbers of peaks, where $\alpha = 0.3$, $gSize = 200$, and $subSize = 7$, and the default settings for the MPB problem in Table I were used.

parameter, we fix the values of the other three parameters. The default values of the four parameters are $\beta = 0.1$, $\alpha = 0.3$, $gSize = 200$, and $subSize = 7$, respectively. Each parameter was tested in a set of values. The four sets of values for the corresponding parameters are β in $\{0, 0.05, 0.1, 0.15, ... , 0.9, 0.95\}$, α in $\{0.1, 0.3, 0.7, 0.9\}$, $gSize$ in $\{30, 50, 70, 100, 150, 200, 250, 300\}$, and $subSize$ in $\{3, 5, 7, 10, 15\}$, respectively.

We first analyze the sensitivity of the parameter β. Table II and Fig. 3 present the effect of varying the values of β with different numbers of peaks. According to the combination process of two overlapping sub-populations in Algorithm 3, the larger the value of β, the more strict the condition for two overlapping sub-populations to merge together, vice versa. The two extreme cases are $\beta = 1.0$ and $\beta = 0.0$. In the case of $\beta = 1.0$, the two overlapping sub-populations merge only if all individuals are within each other's search radius. From Table II and Fig. 3, it can be seen that the performance of CPSOR is not affected too much by using different values of β for all the cases. The results also validate our assumption that the choice of β is not crucial to the performance of the framework. Therefore, we can conclude that the performance of CPSOR is not sensitive to the parameter β. However, we still suggest that the value of 0.1 for β should be used in the basic framework since CPSOR achieved relatively better
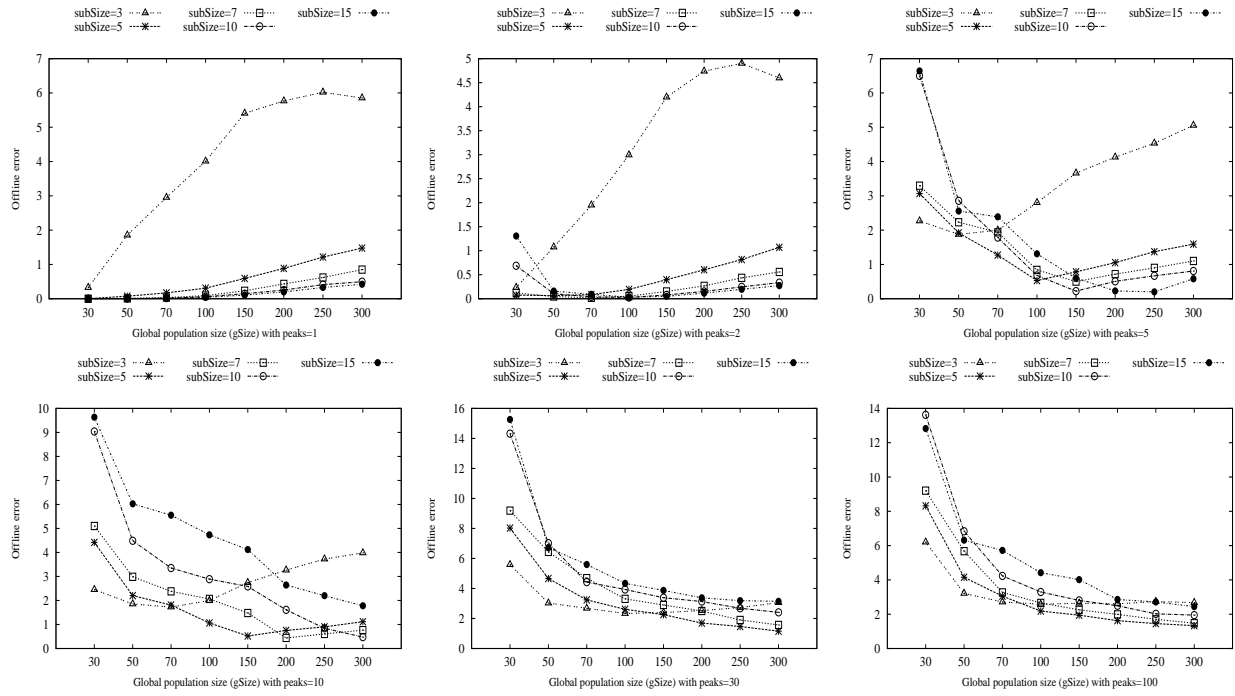
Fig. 4. Offline error of CPSOR with different values of $gSize$ and $subSize$ with different numbers of peaks, where $\alpha = 0.3$, $\beta = 0.1$, and the default settings for the MPB problem in Table I were used.

TABLE III

OFFLINE ERRORS AND $\pm$ STANDARD ERROR FOR CPSOR WITH DIFFERENT CONFIGURATIONS OF $gSzie$ AND $subSize$ WITH 10 PEAKS, WHERE $\alpha = 0.3$, $\beta = 0.1$, AND THE DEFAULT SETTINGS FOR THE MPB PROBLEM IN TABLE I WERE USED

$subSize$	$gSize$							
	30	50	70	100	150	200	250	300
3	2.45	1.85	1.73	2	2.74	3.26	3.73	3.98
	±0.2	±0.1	±0.1	±0.1	±0.2	±0.2	±0.2	±0.2
5	4.42	2.2	1.81	1.06	0.521	0.752	0.901	1.12
	±0.5	±0.2	±0.1	±0.1	±0.08	±0.06	±0.04	±0.08
7	5.1	2.99	2.38	2.06	1.48	0.44	0.607	0.764
	±0.4	±0.3	±0.3	±0.3	±0.2	±0.06	±0.05	±0.04
10	9.04	4.49	3.35	2.89	2.58	1.61	0.842	0.474
	±1	±0.6	±0.4	±0.4	±0.3	±0.1	±0.1	±0.07
15	9.63	6.03	5.55	4.73	4.12	2.65	2.2	1.78
	±1	±0.7	±0.7	±0.8	±0.9	±0.4	±0.4	±0.3

results by using 0.1 than other values for the parameter β.

Fig. 4 shows the offline errors of CPSOR with different values of $gSize$ and $subSize$ for different numbers of peaks. Table III presents the offline errors of different combinations of $gSize$ and $subSize$ on the MPB problem with 10 peaks. From Algorithm 1, the number of sub-populations obtained is determined by the value of $subSize$ if $gSize$ is fixed to a specific value. The larger the value of $subSize$, the smaller the number of sub-populations to be obtained. Hence, a too large or too small value of $subSize$ will cause too few or too many sub-populations to be created, which may be far away from the optimal number of sub-populations needed. Taking $gSize = 200$ in Table III as an example, the offline error by CPSOR is 3.26 when $subSize = 3$, decreases to the smallest value 0.44 with the value of $subSize$ increased to 7, and then

increases to 2.65 when $subSize$ is further increased to 15.

By observing the top three graphs in Fig. 4, it can be seen that the curve of $subSize = 3$ is much worse than the other curves when the number of peaks is relatively small, e.g., less than 5. One reason for this result is due to the large value of $gSize$, e.g., greater than 50, which causes too many sub-populations to be generated. On the other hand, too few individuals in each sub-population, e.g., at most three individuals because of $subSize = 3$, will result in the slow search or premature problem. From Fig. 4 and Table III, it can be seen that these two parameters greatly affect the performance of CPSOR. In order to achieve the best performance, CPSOR needs the optimal configurations in terms of the parameters $gSize$ and $subSize$.

Generally speaking, the larger the number of peaks in the search space, the larger the number of sub-populations needed to achieve the best performance. This trend can be easily observed in the graphs when the number of peaks increases. For example, the best performance of CPSOR with a single peak was obtained by setting $gSize = 30$, which is the smallest value we tested. However, when the number of peaks increases to 100, the largest value $gSize = 300$ helps CPSOR get the best performance even for different values of $subSize$. The suggested configurations regarding the parameters $gSize$ and $subSize$ will be introduced at the end of this section.

In order to investigate what factors may affect the best choice of α, we carried out experiments on the MPB problem with different numbers of peaks and different change frequencies. Generally speaking, the more peaks in the fitness landscape, the higher diversity needed. The diversity regaining in the basic framework can only be achieved by increasing the number of random individuals. Hence, the larger the value of
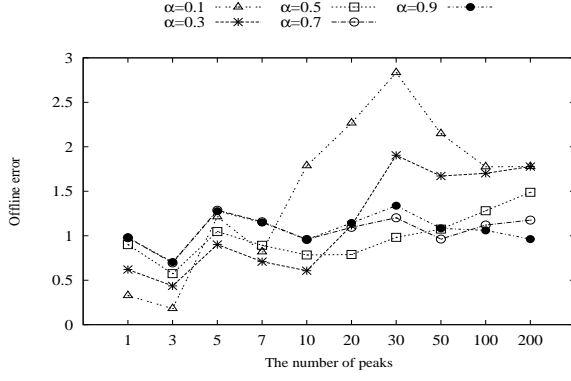
Fig. 5. Offline errors of CPSOR with varying values of α with different numbers of peaks, where $\beta = 0.1$, $gSize = 200$, $subSize = 7$, and the default settings for the MPB problem in Table I were used.

TABLE IV

OFFLINE ERRORS AND $\pm$ STANDARD ERROR FOR DIFFERENT VALUES OF α WITH DIFFERENT NUMBERS OF PEAKS, WHERE $\beta = 0.1$, $gSize = 200$, $subSize = 7$, AND THE DEFAULT SETTINGS FOR THE MPB PROBLEM IN TABLE I WERE USED

α	peaks									
	1	3	5	7	10	20	30	50	100	200
0.1	**0.327**	**0.181**	1.21	0.819	1.79	2.27	2.83	2.15	1.77	1.78
	±0.03	±0.01	±0.2	±0.1	±0.2	±0.5	±0.2	±0.3	±0.1	±0.1
0.3	0.62	0.435	**0.901**	**0.709**	**0.607**	1.12	1.9	1.67	1.7	1.78
	±0.04	±0.03	±0.05	±0.08	±0.05	±0.09	±0.2	±0.1	±0.09	±0.1
0.5	0.9	0.574	1.05	0.893	0.784	**0.789**	**0.982**	1.08	1.28	1.49
	±0.06	±0.03	±0.06	±0.1	±0.06	±0.06	±0.06	±0.06	±0.1	±0.08
0.7	0.978	0.694	1.29	1.16	0.958	1.09	1.2	**0.963**	1.12	1.18
	±0.08	±0.04	±0.06	±0.09	±0.06	±0.08	±0.06	±0.05	±0.06	±0.09
0.9	0.981	0.704	1.28	1.15	0.957	1.14	1.34	1.08	**1.06**	**0.963**
	±0.08	±0.04	±0.06	±0.1	±0.07	±0.06	±0.07	±0.05	±0.04	±0.04

α, the more often the diversity will be increased, which may be good for the environments with a large number of peaks, i.e., more than 50 peaks in the MPB problem. Another factor, which may affect the choice of α, is the change frequency U. Intuitively, the smaller the change frequency, the larger value of α may be needed. Figure 5 and Table IV show the effect of varying the value of α with different numbers of peaks, where the best results over different numbers of peaks are shown in bold font in Table IV. Table V shows the offline errors of different values of α with increasing values of U in the environments with 10 peaks.

The results of Fig. 5 and Table IV validate our assumption, i.e., the larger the number of peaks, the larger α needed. From Fig. 5 and Table IV, it can be seen that $\alpha = 0.1$ helps CPSOR obtain the best performance when the number of peaks is less than 5. Then, when the number of peaks increases, the optimal value of α also increases to get the best performance, e.g., $\alpha = 0.3$ for $peaks = 5$, 7, and 10, $\alpha = 0.5$ for $peaks = 20$ and 30, $\alpha = 0.7$ for $peaks = 50$, and $\alpha = 0.9$ for $peaks = 100$ and 200. By observing Table V, however, we do not get the corresponding results as we expected on the analysis of the relationship between α and U. $\alpha = 0.3$ seems a good choice regarding different numbers of change frequencies. From Table V, it can be seen that the performance

TABLE V

OFFLINE ERRORS AND $\pm$ STANDARD ERROR FOR CPSOR WITH DIFFERENT VALUES OF α AND INCREASING VALUES OF THE CHANGE FREQUENCY U, WHERE $\beta = 0.1$, $gSize = 200$, $subSize = 7$, AND THE DEFAULT SETTINGS FOR THE MPB PROBLEM IN TABLE I WERE USED

U	α=0.1	α=0.3	α=0.5	α=0.7	α=0.9
3000	3.08	**0.722**	1.15	1.42	1.44
	±0.4	±0.08	±0.08	±0.09	±0.09
5000	1.93	**0.44**	0.65	0.765	0.771
	±0.2	±0.06	±0.05	±0.03	±0.05
7000	1.67	**0.391**	0.477	0.561	0.559
	±0.2	±0.07	±0.04	±0.04	±0.04
10000	1.39	0.426	**0.34**	0.409	0.418
	±0.3	±0.05	±0.04	±0.03	±0.04

TABLE VI

THE BEST ERRORS AND STANDARD DEVIATION ACHIEVED BY CPSOR ON DIFFERENT NUMBERS OF PEAKS WITH CORRESPONDING SETTINGS OF $gSize$, $subSize$, AND α, WHERE $\beta = 0.1$ AND THE DEFAULT SETTINGS FOR THE MPB PROBLEM IN TABLE I WERE USED

$peaks$	Error	STD	$gSize$	$subSize$	α
1	7.84e-05	6.91e-06	150	15	0.1
2	0.00366	3.54e-04	50	15	0.9
5	0.192	0.0232	100	10	0.5
7	0.461	0.108	150	7	0.3
10	0.44	0.0563	200	7	0.3
20	0.721	0.0627	200	7	0.5
30	0.982	0.0632	250	7	0.5
50	0.951	0.0518	200	7	0.9
100	1.05	0.038	300	7	0.7
200	0.931	0.0334	300	7	0.9

gets better when U increases for most values of α, e.g., the offline error decreases from 1.15 to 0.34 when the value of U increases from 3000 to 10000 with $\alpha = 0.5$. Therefore, according to the results of Fig. 5, Table IV, and Table V, we can roughly draw a conclusion that the number of peaks may be the main factor to affect the choice of the optimal α.

So far, we have roughly recognized which parameters are sensitive to the performance of the CPSOR algorithm. β is not a key parameter in the framework, so we use a constant value of 0.1 for β in the remaining experiments. For the other three parameters, the best configurations seem to be relevant to the number of peaks. Therefore, in order to find out some hints on how to setup these parameters, we summarize the results of all the combinations of the three parameters used in the paper. The best configurations (may be not the true optimal configurations as we can not test all the possibilities) for different numbers of peaks are listed in Table VI.

It should be noted that there are of course many other factors that may affect the optimal choice of α, $gSize$, and $subSize$, such as the width severity, height severity, shift length, and the number of dimensions. The parameters of PSO algorithms (ω, η_1, and η_2) may also affect the choice. However, we do not discuss these factors as they are not the main objective in this paper.

From Table VI, it is interesting to see that the best $subSize$ is 7 for most test cases. Based on the above results and analysis of the CPSOR algorithm, we give some empirical formulae to setup these parameters for the basic framework to solve the
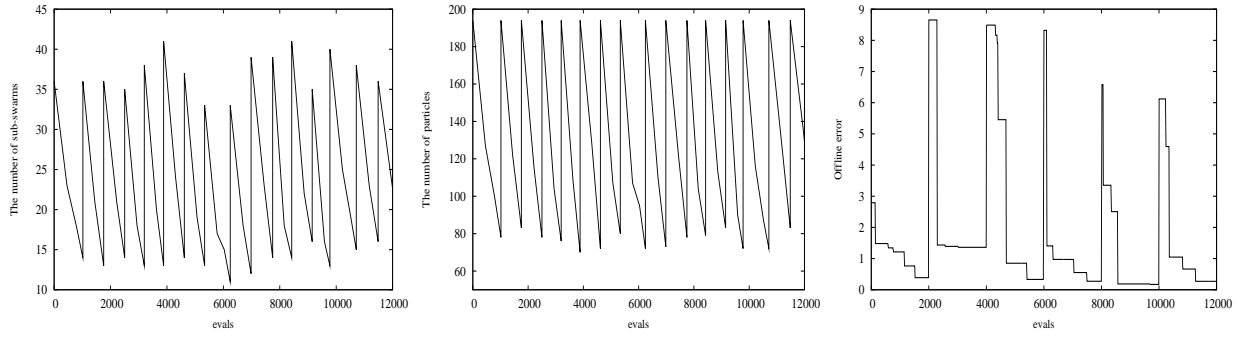
Fig. 6. Progress of the number of sub-populations (left), the number of individuals (middle), and the offline error (right), where the suggested configuration for the framework and the default settings for the MPB problem in Table I, except U=2000 and D=2, were used.

MPB problem. For β and $subSize$, we use 0.1 and 7 for all the following experiments in this paper, respectively, and $gSize$ and α are estimated by the following equations:

$$gSize = 300 \cdot (1 - exp(-0.33 \cdot peaks^{0.5})) \qquad (17)$$

$$\alpha = 1 - exp(-0.2 \cdot peaks^{0.45}) \qquad (18)$$

For the remaining experiments, we use the suggested configuration for algorithms that use the basic framework, i.e., CPSOR, CDER, and CGAR.

A limitation to use Eqs. (17) and (18) to set the parameters $gSize$ and α is that we have to know the number of peaks, which is assumed unknown to users. However, how to set the population size is also a general problem for most EAs. At the end of this section, we summarize a principle for setting the configurations of the basic framework: the larger the number of local optima in the fitness landscape, the larger the value of $gSize$ and α.

2) Working Mechanism: In this set of experiments, we illustrate the working mechanism of the basic framework through the CPSOR algorithm with respect to two different aspects to see how the basic framework locates and tracks multiple peaks. The experiments were carried out on the MPB problem in two dimensions with $U = 2000$ in a single typical run. The suggested configuration was used in this set of experiments. Fig. 6 shows the progress of the number of sub-populations, the number of individuals, and the offline error over six successive environments. Fig. 7 presents the distribution of particles' personal best positions of six different episodes in the run. In Fig. 7, the ten black square boxes of each graph are the locations of the ten peaks and the cross points are the $pbest$ positions of particles.

From Fig. 6, it can be clearly seen that the number of sub-populations decreases to a certain level and then increases by clustering a new random population generated by the random immigrants strategy. The similar observations can be seen for changes of the number of individuals in the middle graph. In the offline error graph (the right one), we can not see the increase when the random immigrants strategy takes place. This is because we record only the best solution so far to calculate the offline error. From the results, we can understand why this basic framework is suitable for any change type or even undetectable environments, because the random immigrants

strategy is triggered not by the environmental changes but by the current population diversity only. So, this basic framework does not consider what kind of changes or whether changes have occurred or not.

For further detailed information about the distribution of individuals, we can observe the population distribution episodes in Fig. 7. Fig. 7-(a) shows the initial 36 sub-populations with a total of 194 individuals just after the clustering process. The second one shows all the surviving individuals just before the random immigrants strategy was triggered. At this moment, only 14 sub-populations with a total of 78 surviving individuals and they were close to their own nearest peaks but not distributed across the whole fitness landscape. So far, more than half of the individuals have been removed by the redundancy control mechanism. This procedure saved a lot of computing resources for the *useful* (surviving) individuals. The third graph shows the whole populations when the third change happened. The remaining graphs in Fig. 7 show similar situations as the top three graphs in the following environments.

From the above experimental results, it can be seen that the population diversity is able to be automatically regained when the diversity decreases to a threshold level. In this framework, the diversity-maintaining technique is not the only technique for handling the dynamism, the memory with elitism technique is also involved for dealing with the dynamism. From the framework, we know that "useless" (redundant) individuals will be gradually removed before the random immigrants method is triggered. Therefore, those surviving individuals, which are usually good individuals with high fitness, will automatically go to the next environment. As we know, those surviving individuals, which represent the memory of previous environments, will help the search in the new environment if the next environment is similar to the previous one.

C. Investigation of Clustering and Random Immigrants Methods

In this section, we investigate how much benefit the multi-population method will get by using the clustering and random immigrants strategies introduced in Sect. III. In order to compare the algorithms with clustering and the algorithms without clustering, we just use a simple multi-population method where a certain number of sub-populations are created
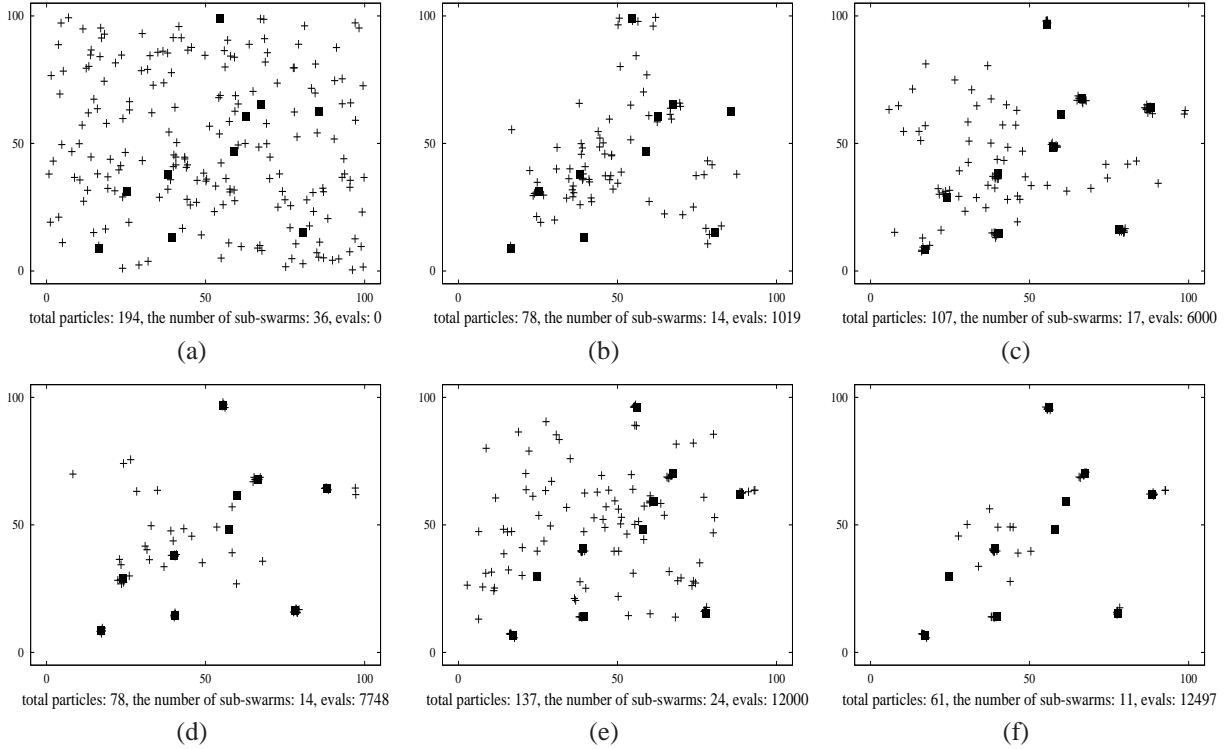
Fig. 7. Distribution of the *pbest* positions of particles in six different episodes, where the suggested configuration for the framework and the default settings for the MPB problem in Table I, except U=2000 and D=2, were used.

randomly. In the simple multi-population method, there is no overcrowding control among sub-populations because it is very hard to do that without defining their search areas. The aim of this set of experiments is just to show that the clustering method is a suitable method to create sub-populations although this may not be a fair comparison between the methods with and without clustering. In order to check the efficiency of the random immigrants strategy, we also carry out experiments on the comparison of the algorithms using the basic framework (i.e., CPSOR, CGAR, and CDER) with the algorithms using the clustering method but no mechanism of handling hard-to-detect or undetectable environments (i.e., CPSO [78], CDE, and CGA). The CDE and CGA algorithms use the same working mechanism proposed in the CPSO algorithm where they all use the elitism, memory, and random immigrants strategies once changes have been detected.

We carry out experiments to compare three sets of algorithms: (CPSOR, CPSO, and MultiPSO), (CGAR, CGA, and MultiGA), and (CDER, CDE, and MultiDE). MultiPSO, MultiGA, and MultiDE are the algorithms using the simple multi-population method without any overcrowding control. For each set, we used the same configurations regarding the global population size ($gSize$) and the sub-population size ($subSize$). In MultiPSO, MultiGA, and MultiDE, the size of each sub-population is $subSize$ and the number of sub-populations is fixed to $gSize/subSize$.

Table VII shows the comparison of offline errors of each set of algorithms with different configurations on the MPB problem with ten peaks. The best offline error of each algorithm is shown in bold font. From the results, we can get three different observations. Firstly, CPSO, CGA, and CDE obtain the best results on most test cases due to the corresponding strategies used once a change has been detected. However, we do find that the best results of the CPSOR and CGAR algorithms are 0.474 and 2.41, which are better than the best results (0.63 and 2.74) obtained by the CPSO and CGA algorithms, respectively. This means that the algorithms without change detection can work as well as the algorithms with change detection. In other words, the basic framework proposed in Sect. III is effective for DOPs.

Secondly, all the results achieved by the algorithms using the clustering method are much better than the results achieved by the simple multi-population methods. However, it is hard to judge the contribution of the clustering method because we can not conduct the operation of redundancy control in the simple multi-population methods without the clustering method.

Thirdly, the results of CGAR and CDER are worse than the results of CPSOR. One reason behind this lies in the mutation operation in both GA and DE. The mutation will cause a sub-population jumping from one peak with a lower height to another peak with a higher height. This means that the algorithms using mutation operators will discard the peaks with a lower height that they have searched. Therefore, there is a trend where all sub-populations try to explore the same promising areas in the search space, which is against the aim of using the multi-population method, i.e., to maintain sub-populations in different areas. For PSO algorithms, there is no mutation. Once a sub-population is created in a sub-area in the search space, it just focuses on exploitation on that sub-area due to the working mechanism of PSO. The sub-population

		$subSize$				
$gSize$	alg	3	5	7	10	15
30	CPSOR	2.45±0.2	4.42±0.5	5.1±0.4	9.04±1	9.63±1
	CPSO	1.5±0.2	2.1±0.2	2.62±0.2	3.52±0.4	4.39±0.4
	MultiPSO	8.94±0.2	7.3±0.2	8.06±0.2	8.73±0.3	9.53±0.3
	CGAR	6.72±0.4	9.58±0.6	9.56±0.9	18.6±0.9	18.6±0.9
	CGA	4.45±0.4	3.86±0.2	4.68±0.3	5.63±0.4	6.91±0.4
	MultiGA	10.6±0.3	**9.14**±0.3	9.42±0.3	10±0.3	10.9±0.3
	CDER	12.7±1	13.7±3	11.3±1	19.6±3	18.6±3
	CDE	5.71±0.3	5.38±0.3	5.78±0.2	6.08±0.2	6.85±0.2
	MultiDE	14.6±0.3	9.69±0.3	9.77±0.2	10.1±0.2	10.3±0.2
70	CPSOR	1.73±0.1	1.81±0.1	2.38±0.3	3.35±0.4	5.55±0.7
	CPSO	0.715±0.1	0.99±0.1	1.24±0.1	1.81±0.2	2.56±0.2
	MultiPSO	8±0.2	**5.58**±0.2	5.97±0.3	6.45±0.3	7.57±0.3
	CGAR	4.7±0.3	4.27±0.3	6.55±0.9	6.18±0.7	9.36±0.8
	CGA	2.94±0.2	2.89±0.1	3.06±0.1	3.7±0.2	4.8±0.3
	MultiGA	23.5±0.7	15.9±0.4	13.5±0.4	11.5±0.3	10.9±0.3
	CDER	8.54±0.5	6±0.3	6.95±0.5	7.55±0.6	9.49±1
	CDE	3.78±0.2	3.76±0.2	4.35±0.2	5.33±0.2	6.14±0.3
	MultiDE	11.6±0.2	**9.62**±0.2	10.1±0.2	10.7±0.2	10.9±0.2
150	CPSOR	2.74±0.2	0.52±0.1	1.48±0.2	2.58±0.3	4.12±0.9
	CPSO	1.16±0.07	**0.63**±0.1	0.74±0.1	0.97±0.1	1.40±0.1
	MultiPSO	8.97±0.2	6.61±0.2	6.1±0.2	6.17±0.2	6.37±0.2
	CGAR	5.42±0.4	2.68±0.1	3.03±0.2	3.73±0.3	5.6±0.7
	CGA	3.37±0.2	2.99±0.2	**2.74**±0.1	2.95±0.2	3.55±0.2
	MultiGA	36.3±0.6	28.5±0.5	23.8±0.4	20.1±0.5	17.7±0.5
	CDER	7.72±0.5	5.58±0.2	**5.27**±0.2	5.5±0.3	6.44±0.3
	CDE	3.2±0.1	**3.1**±0.1	3.53±0.2	4.39±0.2	5.54±0.2
	MultiDE	13.6±0.2	12.9±0.2	13.5±0.1	14.2±0.1	14.8±0.2
300	CPSOR	3.98±0.2	1.12±0.1	0.76±0.04	**0.47**±0.1	1.78±0.3
	CPSO	4.39±0.1	1.99±0.1	1.16±0.04	1.12±0.1	1.24±0.1
	MultiPSO	12.7±0.2	10.4±0.1	9.46±0.1	8.99±0.2	8.65±0.2
	CGAR	6.25±0.4	3.48±0.2	2.89±0.2	**2.41**±0.2	3.4±0.3
	CGA	4.45±0.2	3.9±0.2	3.6±0.1	3.31±0.2	3.44±0.2
	MultiGA	40.4±0.6	34.2±0.6	30.4±0.6	26.3±0.4	24±0.5
	CDER	8.2±0.3	5.63±0.2	5.51±0.2	5.54±0.2	5.73±0.2
	CDE	4.74±0.1	4.38±0.1	4.12±0.1	4.55±0.1	5.43±0.2
	MultiDE	19.8±0.2	19.7±0.2	19.8±0.2	20.4±0.2	20.7±0.2

Algorithm	Severity of shift length (s)						
	0	1	2	3	4	5	6
CPSOR	**0.418**	**0.599**	0.849	0.964	1.38	1.69	2.07
	±0.08	±0.04	±0.06	±0.07	±0.09	±0.08	±0.13
CGAR	1.48	2.6	2.76	2.96	3.16	3.46	3.8
	±0.15	±0.132	±0.13	±0.13	±0.13	±0.16	±0.24
CDER	2.56	5.52	7.47	8.62	9.81	10.7	11.4
	±0.26	±0.16	±0.27	±0.25	±0.32	±0.34	±0.36
CPSO	0.465	0.715	**0.843**	**0.911**	**0.997**	**1.08**	**1.23**
	±0.08	±0.103	±0.15	±0.12	±0.12	±0.14	±0.14
mCPSO	1.18	2.05	2.80	3.57	4.18	4.89	5.53
	±0.07	±0.07	±0.07	±0.08	±0.09	±0.11	±0.13
mQSO	1.18	1.75	2.40	3.00	3.59	4.24	4.79
	±0.07	±0.06	±0.06	±0.06	±0.10	±0.10	±0.10
CESO	0.85	1.38	1.78	2.03	2.23	2.52	2.74
	±0.02	±0.02	±0.02	±0.03	±0.05	±0.06	±0.10
rSPSO	0.74	1.50	1.87	2.4	2.90	3.25	3.86
	±0.08	±0.08	±0.05	±0.08	±0.08	±0.09	±0.11
SPSO	0.95	2.51	3.78	4.96	2.56	6.76	7.68
	±0.08	±0.09	±0.09	±0.12	±0.13	±0.15	±0.16
ESCA	1.72	1.53	1.57	1.67	1.72	1.78	1.79
	±0.03	±0.01	±0.01	±0.01	±0.03	±0.06	±0.03
PSO-CP	0.87	1.31	1.98	2.21	2.61	3.20	3.93
	±0.07	±0.06	±0.06	±0.06	±0.11	±0.13	±0.14

Algorithm	0	1	2	3	4	5	6	Overall
CPSOR	1	1	2	2	2	2	3	2
CGAR	9	10	8	7	8	7	5	7
CDER	11	11	11	11	11	11	11	10
CPSO	2	2	1	1	1	1	1	1
mCPSO	7	8	9	9	10	9	9	9
mQSO	7	7	7	8	9	8	8	7
CESO	4	4	4	4	4	4	4	3
rSPSO	3	5	5	6	7	6	6	6
SPSO	6	9	10	10	5	10	10	8
ESCA	10	6	3	3	3	3	2	4
PSO-CP	5	3	6	5	6	5	7	5

will not jump to other sub-areas. This is even obvious for the PSO algorithms with the *gbest* model. This result also reminds us that we should choose or design effective local search algorithms rather than global search operators if we use multi-population methods to locate and track multiple optima. There may be some other factors that cause the inefficiency of CGAR and CDER, e.g., improper parameter settings. How to adjust the parameter settings for CGAR and CDER is not the main aim of this paper, and, hence, is not discussed here.

So far, we have investigated how the basic framework works through the three case studies (CPSOR, CGAR, and CDER) in dynamic environments. However, we have not known how they would perform in comparison with other algorithms yet. The following section will present the results of comparison with other algorithms.

D. Comparison of the Involved Peer Algorithms

In this section, experiments were carried out to compare the performance of algorithms that use the basic framework (i.e., CPSOR, CGAR, and CDER) with several peer algorithms,

including mCPSO [6], mQSO [6], SPSO [54], rSPSO [4], CESO [46], ESCA [47], CPSO [78], PSO-CP [45], HmSO [29], and RVDEA [73], on the MPB problems with different shift severities (s) and different numbers of peaks (*peaks*).

1) Effect of Varying the Shift Length Severity: Table VIII presents the comparison results of different algorithms regarding varying the severity of the shift length s. Table IX shows the rankings of algorithms with different shift length severities. The total ranking of each algorithm is calculated by the average rankings obtained for all test cases. From the results, it can be seen that when the shift severity is larger than 1, CPSO achieves the best results, which are much better than that of all the other algorithms. CPSOR obtains the best results among all the algorithms when there is no shift or small shift length (i.e., $s = 1.0$). Compared with other algorithms except CPSO, the results obtained by CPSOR are much better than those of other algorithms. For the rest of the algorithms, CESO and ESCA are the best two performers due to their

TABLE X

OFFLINE ERRORS AND ± STANDARD ERROR FOR DIFFERENT ALGORITHMS ON THE MPB PROBLEM WITH DIFFERENT NUMBERS OF PEAKS, WHERE THE SUGGESTED CONFIGURATION FOR THE FRAMEWORK AND THE DEFAULT SETTINGS FOR THE MPB PROBLEM IN TABLE I EXCEPT *peaks* WERE USED

peaks	CPSOR	CGAR	CDER	CPSO	mCPSO	mQSO	mCPSO*	mQSO*	CESO	rSPSO	SPSO	ESCA	PSO-CP	HmSO	RVDEA
1	0.0356 ±0.008	2.02 ±0.05	0.903 ±0.20	**2.29e-04** ±1.04e-04	4.93 ±0.17	5.07 ±0.17	4.93 ±0.17	5.07 ±0.17	1.04 ±0.00	1.42 ±0.06	2.64 ±0.10	0.98 ±0.0	3.41 ±0.06	0.87 ±0.05	1.02 –
2	0.0535 ±0.005	1.88 ±0.10	2.6 ±0.63	**0.00566** ±0.002	3.36 ±0.26	3.47 ±0.23	3.36 ±0.26	3.47 ±0.23	– –	1.10 ±0.03	2.31 ±0.11	– –	– –	– –	– –
5	0.549 ±0.049	2.56 ±0.1	8.02 ±0.34	**0.361** ±0.15	2.07 ±0.08	1.81 ±0.07	2.07 ±0.11	1.81 ±0.07	– –	1.04 ±0.03	2.15 ±0.07	– –	– –	1.18 ±0.04	– –
7	**0.594** ±0.08	2.98 ±0.18	6.74 ±0.39	0.675 ±0.09	2.11 ±0.11	1.77 ±0.07	2.11 ±0.11	1.77 ±0.07	– –	1.21 ±0.05	1.98 ±0.04	– –	– –	– –	– –
10	**0.599** ±0.048	2.6 ±0.13	5.52 ±0.16	0.715 ±0.103	2.08 ±0.07	1.80 ±0.06	2.05 ±0.07	1.75 ±0.06	1.38 ±0.02	1.50 ±0.08	2.51 ±0.01	1.54 ±0.02	1.31 ±0.06	1.42 ±0.04	3.54 –
20	**0.796** ±0.05	3.66 ±0.14	7.49 ±0.27	1.18 ±0.09	2.64 ±0.07	2.42 ±0.07	2.95 ±0.08	2.74 ±0.07	1.72 ±0.02	2.20 ±0.07	3.21 ±0.07	1.89 ±0.04	– –	1.5 ±0.06	3.87 –
30	**1.05** ±0.06	3.12 ±0.1	5.51 ±0.12	1.34 ±0.07	2.63 ±0.08	2.48 ±0.07	3.38 ±0.11	3.27 ±0.11	1.24 ±0.01	2.62 ±0.07	3.64 ±0.07	1.52 ±0.02	2.02 ±0.07	1.65 ±0.04	3.92 –
50	**0.986** ±0.05	3.26 ±0.11	5.79 ±0.15	1.42 ±0.07	2.65 ±0.06	2.50 ±0.06	3.68 ±0.11	3.65 ±0.11	1.45 ±0.01	2.72 ±0.08	3.86 ±0.08	1.67 ±0.02	2.14 ±0.08	1.66 ±0.06	3.78 –
100	**1.06** ±0.04	2.68 ±0.07	4.12 ±0.1	1.09 ±0.03	2.49 ±0.04	2.36 ±0.04	4.07 ±0.09	3.93 ±0.08	1.28 ±0.02	2.93 ±0.06	4.01 ±0.07	1.61 ±0.01	2.04 ±0.07	1.68 ±0.03	3.37 –
200	**0.949** ±0.04	2.39 ±0.07	3.71 ±0.11	0.955 ±0.04	2.44 ±0.04	2.26 ±0.03	3.97 ±0.08	3.86 ±0.07	– –	2.79 ±0.05	3.82 ±0.05	– –	– –	1.71 ±0.02	3.54 –

TABLE XI

ALGORITHM RANKINGS ON THE MPB PROBLEM WITH DIFFERENT NUMBERS OF PEAKS

peaks	CPSOR	CGAR	CDER	CPSO	mCPSO	mQSO	mCPSO*	mQSO*	CESO	rSPSO	SPSO	ESCA	PSO-CP	HmSO	RVDEA
1	2	9	4	1	12	14	12	14	7	8	10	5	11	3	6
2	2	4	6	1	7	9	7	9	–	3	5	–	–	–	–
5	2	10	11	1	7	5	7	5	–	3	9	–	–	4	–
7	1	9	10	2	7	4	7	4	–	3	6	–	–	–	–
10	1	13	15	2	11	9	10	8	4	6	12	7	3	5	14
20	1	12	14	2	8	7	10	9	4	6	11	5	–	3	13
30	1	10	15	3	9	7	12	11	2	8	13	4	6	5	14
50	1	10	15	2	8	7	12	11	3	9	14	5	6	4	13
100	1	9	15	2	8	7	14	12	3	10	13	4	6	5	11
200	1	5	9	2	6	4	12	11	–	7	10	–	–	8	3
Overall	1	10	15	2	9	8	12	11	3	6	13	5	7	4	14

diversity maintaining and local search strategies.

2) Effect of Varying the Number of Peaks: Table X shows the offline errors of all the algorithms when varying the number of peaks. The corresponding rankings of each algorithm are shown in Table XI. In the two tables, the "–" sign means that there is no result available in the original paper. The algorithms mCPSO* and mQSO* denote mCPSO without anti-convergence and mQSO without anti-convergence, respectively.

From the results, it can be easily seen that CPSOR outperforms all the other peer algorithms when the number of peaks is larger than five. By looking at the ranking table, the CPSO algorithm takes the second place followed by CESO, HmSO, ESCA, rSPSO, and PSO-CP, which are recently proposed.

From the comparison results of CPSOR with the other peer algorithms on the MPB problem with different numbers of peaks and shift severities, we can draw a conclusion that the CPSOR algorithm is a competitive optimizer for DOPs.

E. CPSOR in Hard-to-Detect Environments

In all the above experiments, we assume that all peaks change over time in the fitness landscape. Therefore, changes can be easily detected even by a single random point in

the fitness landscape. In this section, we investigate the performance of CPSOR in hard-to-detect environments in comparison with the CPSO algorithm. Fig. 8 shows the effect of varying the ratio of changing peaks $cPeaks$ with different numbers of peaks for the CPSOR algorithm. Table XII and Table XIII show the comparison between CPSOR and CPSO under different ratios of changing peaks with different shift severities and different numbers of peaks, respectively. In both tables, the best results of the two algorithms are shown in bold font.

From the results, it can be seen that different ratios of changing peaks do bring different levels of difficulty for both algorithms. Generally speaking, the smaller the ratio of changing peaks, the harder it is for an algorithm to detect changes, and hence, the more difficult it is for an algorithm to track the changing peaks. From Fig. 8, it can be seen that $cPeaks = 0.1$ brings the biggest difficulty for CPSOR in most test cases, except the case of $peaks = 50$. By observing Table XII, although CPSO outperforms CPSOR when $cPeaks = 1$, the results of CPSOR are much better than those of CPSO on all shift severities when $cPeaks$ is less than 0.5.

From the results of Table XIII, we can make two observations. Firstly, CPSOR outperforms CPSO on most test

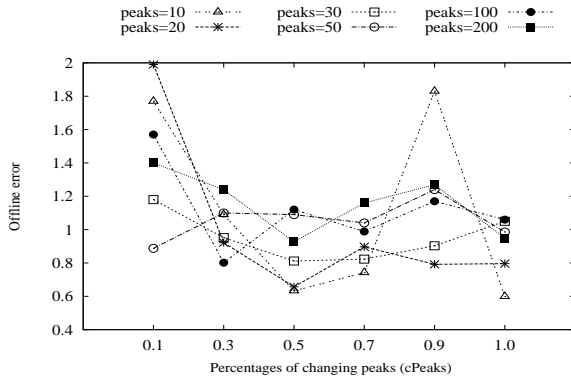peaks=10 △ peaks=30 □ peaks=100 ● peaks=200 ■
peaks=20 ✳ peaks=50 ○

Fig. 8. Offline errors of CPSOR on the MPB problem with different ratios of changing peaks $cPeaks$ and different numbers of peaks, where the suggested configuration for CPSOR and the default settings for the MPB problem in Table I were used.

TABLE XII

COMPARISON OF CPSOR AND CPSO ON THE MPB PROBLEM WITH DIFFERENT RATIOS OF CHANGING PEAKS $cPeaks$ AND DIFFERENT SHIFT SEVERITIES s, WHERE THE SUGGESTED CONFIGURATION FOR CPSOR AND THE DEFAULT SETTINGS FOR THE MPB PROBLEM IN TABLE I, EXCEPT $cPeaks$ AND s, WERE USED. RESULTS REPORTED ARE THE OFFLINE ERRORS AND $\pm$ STANDARD ERRORS

s	Algorithm	$cPeaks$					
		0.1	0.3	0.5	0.7	0.9	1.0
0	CPSOR	**1.47**	**0.535**	**0.5**	**0.6**	1.72	**0.418**
		±0.3	±0.2	±0.06	±0.1	±0.2	±0.08
	CPSO	3.01	2.7	0.904	0.765	**1.68**	0.465
		±0.4	±0.5	±0.09	±0.1	±0.2	±0.09
1	CPSOR	**1.77**	**1.09**	**0.633**	**0.742**	**1.83**	**0.599**
		±0.3	±0.3	±0.05	±0.08	±0.2	±0.05
	CPSO	3	2.76	0.912	1.02	2.11	0.715
		±0.4	±0.4	±0.1	±0.1	±0.2	±0.1
2	CPSOR	**1.89**	**1.17**	**0.781**	**0.99**	**2.04**	0.849
		±0.3	±0.2	±0.06	±0.09	±0.1	±0.07
	CPSO	3.24	2.96	0.939	1.19	2.24	**0.843**
		±0.4	±0.4	±0.1	±0.1	±0.2	±0.2
3	CPSOR	**1.94**	**1.61**	0.995	**1.24**	**2.35**	0.964
		±0.3	±0.3	±0.05	±0.1	±0.1	±0.07
	CPSO	3.11	3.38	**0.928**	1.32	2.58	**0.911**
		±0.3	±0.4	±0.08	±0.2	±0.2	±0.1
4	CPSOR	**2.06**	**2.09**	1.25	1.58	**2.6**	1.38
		±0.3	±0.2	±0.07	±0.1	±0.1	±0.09
	CPSO	3.17	3.29	**1.11**	**1.43**	2.66	**0.997**
		±0.3	±0.3	±0.09	±0.2	±0.2	±0.1
5	CPSOR	**2.05**	**2.6**	1.34	1.72	2.88	1.69
		±0.2	±0.2	±0.05	±0.09	±0.1	±0.08
	CPSO	3.27	3.79	**1.14**	**1.45**	**2.69**	**1.08**
		±0.3	±0.3	±0.1	±0.2	±0.1	±0.1
6	CPSOR	**2.17**	**3.04**	1.43	2.03	3.13	2.07
		±0.3	±0.2	±0.07	±0.08	±0.1	±0.1
	CPSO	3.31	3.84	**1.21**	**1.44**	**2.87**	**1.23**
		±0.4	±0.3	±0.1	±0.1	±0.2	±0.1

cases. Secondly, for each particular number of peaks, i.e., $peaks = 50$, the offline error for CPSO basically increases with the decrease of the ratio of changing peaks, but not so obviously for CPSOR.

The comparison results with the peer algorithms in this paper clearly show that the performance of the proposed clustering framework is competitive in detectable dynamic environments. Most importantly, the results also show the

TABLE XIII

COMPARISON OF CPSOR AND CPSO ON THE MPB PROBLEM WITH DIFFERENT RATIOS OF CHANGING PEAKS $cPeaks$ AND DIFFERENT NUMBERS OF PEAKS, WHERE THE SUGGESTED CONFIGURATION FOR CPSOR AND THE DEFAULT SETTINGS FOR THE MPB PROBLEM IN TABLE I, EXCEPT $cPeaks$ AND $peaks$, WERE USED. RESULTS REPORTED ARE THE OFFLINE ERRORS AND $\pm$ STANDARD ERRORS

$peaks$	Algorithm	$cPeaks$					
		0.1	0.3	0.5	0.7	0.9	1.0
7	CPSOR	**0.931**	**0.761**	**1.02**	0.952	**0.752**	**0.594**
		±0.2	±0.1	±0.2	±0.1	±0.1	±0.09
	CPSO	3.94	1.66	1.54	**0.822**	0.932	0.675
		±1	±0.3	±0.2	±0.2	±0.2	±0.1
10	CPSOR	**1.77**	**1.09**	**0.633**	**0.742**	**1.83**	**0.599**
		±0.3	±0.3	±0.05	±0.08	±0.2	±0.05
	CPSO	3	2.76	0.912	1.02	2.11	0.715
		±0.4	±0.4	±0.1	±0.1	±0.2	±0.1
20	CPSOR	**1.99**	**0.922**	**0.657**	**0.897**	**0.792**	**0.796**
		±0.5	±0.1	±0.07	±0.07	±0.1	±0.05
	CPSO	4.58	1.33	0.998	1.52	1.34	1.18
		±0.5	±0.2	±0.1	±0.1	±0.1	±0.09
30	CPSOR	**1.18**	**0.952**	**0.812**	**0.823**	**0.903**	**1.05**
		±0.2	±0.1	±0.09	±0.07	±0.07	±0.06
	CPSO	4.43	1.45	1.47	1.12	1.05	1.34
		±0.6	±0.2	±0.1	±0.09	±0.07	±0.07
50	CPSOR	**0.888**	**1.1**	**1.09**	**1.04**	**1.24**	**0.986**
		±0.09	±0.07	±0.07	±0.06	±0.07	±0.06
	CPSO	2.53	1.87	1.42	1.32	1.78	1.42
		±0.4	±0.1	±0.1	±0.1	±0.09	±0.08
100	CPSOR	**1.57**	**0.802**	**1.12**	**0.989**	**1.17**	**1.06**
		±0.1	±0.05	±0.07	±0.04	±0.05	±0.04
	CPSO	3.4	1.22	1.37	1.26	1.37	1.09
		±0.6	±0.1	±0.1	±0.08	±0.06	±0.04
200	CPSOR	**1.4**	**1.24**	**0.927**	**1.16**	1.27	**0.949**
		±0.2	±0.08	±0.05	±0.05	±0.04	±0.04
	CPSO	3.06	1.93	1.08	1.17	**1.18**	0.955
		±0.3	±0.1	±0.08	±0.05	±0.03	±0.04

unique advantages of the proposed clustering framework in hard-to-detect environments.

VI. DISCUSSION IN REAL-WORLD APPLICATIONS

The general framework proposed in this paper is for dynamic environments where dynamism is hard to detect or totally undetectable, which is close to real-world problems. Here, hard-to-detect means the dynamism can be detected but with a huge price, and hence, is impractical in real-world problems. Fig. 1 in Sect. III-A is such an example. The change can be detected with the cost of enumerating all the points in the fitness landscape, which is impossible for problems in the continuous space. Another example is the dynamic TSP [37]. For a TSP with a small number of cities, e.g., less than 100, it is easy to check the change when the position of one city changes. However, it will become impractical to check the position change of a single city among millions of cities. Therefore, this framework is particularly useful for the environments where dynamism is hard-to-detect or undetectable.

Below, we first present some discussions regarding the usage of the clustering framework and then give some discussions regarding the potential application of the framework in several real-world problems.

A. Usage of the Framework

The clustering framework is proposed for EAs in dynamic environments with any change properties to locate and track multiple optima. Hence, we believe that it would work for any optimization problems that have multiple optima in the fitness landscape, which is one of the most common features of many optimization problems. To apply this framework, users only need to define a proper distance metric for a particular problem to be solved.

Although this framework is easy to apply, it does not have any domain knowledge. So, to effectively solve a particular problem with this framework, users should design or choose search operators that may make use of specific domain knowledge. Regarding the parameter settings in this framework, users can follow the suggested settings in this paper. But, we suggest that users should re-adjust them because they are problem dependent.

B. Path Planning in Dynamic Environments

In the real world, hard-to-detect environments widely exist. Taking the design of the path planning system as an example, the optimal path planning is an important issue in navigation of autonomous mobile robots. The objective is to find an optimal collision-free path from a starting point to a goal in a given environment according to some criteria, e.g., distance, time, or energy. In the real world, the environment changes over time due to moving obstacles with unknown trajectories, e.g., a large public square full of people moving in different directions and a factory full of moving robots and human workers, etc. The distance that the sensors of a robot can reach is limited. Furthermore, the ways of moving obstacles are unknown. Therefore, the robot can not detect changes of a moving obstacle that is beyond its detectable range.

There are many classic approaches for the path planning problem, e.g., the potential field methods [22], visibility graph methods [42], and grid methods [8]. These classic approaches have a big disadvantage of being trapped in local minima, which makes them inefficient in practice [48]. There are also methods for path planning based on EAs [7]. The methods proposed in [74], [84] have shown that it is effective to apply the general idea of evolutionary computation to solve a problem in a more natural and suitable representation. Our approach can be also applied to solve the path planning problem since we do not need to detect the changes in order to maintain the population diversity.

C. Dynamic Scheduling Problems

For static scheduling problems, all resources and activities are given in advance. Constraints are also fixed, i.e., there are no uncertainties in the behavior of resources and activities [35]. However, in the real world, every schedule is subject to unexpected events, such as unexpected resource failures (e.g., machine breakdown), the arrival of new activities during the solving phase, and shorter/longer processing time than expected, etc. Among these uncertain constraints, some are hard-to-detect. For example, if the time of completion of a particular activity is re-scheduled later than its original time, all current solutions will still be feasible regarding this change. In this case, it is not easy to detect the change by monitoring the fitness or feasibility of individuals in the current population. As a result, we fail to respond to the change even though we should. In a dynamic scheduling problem, if a constraint change does not affect the current population, that is, the change expands the feasible areas in the search space, that change will be difficult to detect. Two examples of dynamic scheduling problems in the real world are timetabling problems and flight assignment problems.

A timetabling problem can be described as the scheduling of a certain number of activities (lectures, labs, surgeries, etc.), which involves a particular group of people (teachers, students, etc.) over a finite period of time, requiring certain resources (rooms, projectors, etc) in conformity with the availability of resources so that it maximizes the possibility of allocation or minimizes the violation of constraints [21]. There are various timetabling problems, including, school timetabling, examination timetabling, employment timetabling, and university course timetabling, etc. There are also different methods to solve timetabling problems, such as, graph coloring [15], GAs [67], integer programming [61], tabu search [26], and constraint programming approaches [24], [30]. A reviewed work for dynamic timetabling problems with changes and uncertainties was reported in [13]. A complex timetabling problem was investigated in [59], and the system designed in [59] is currently used for many varied course timetabling problems encountered each term at Purdue University [59].

There are some advantages using the framework proposed in this paper to solve dynamic scheduling problems. Firstly, the framework together with specialized knowledge in timetabling is able to find new solutions due to the diversity maintaining mechanism, even for undetectable constraint changes. Secondly. the new solutions would be close to the original ones because of the memory with elitism scheme in the framework. In the framework, some old individuals carrying information of previous environments will survive in the new environment. Therefore, the new solutions are very likely to contain information from previous environments based on the survived individuals. This is an important issue in dynamic scheduling problems. Usually, a new solution, which is close to an original one (minimal changes), is the best choice for decision makers because it will reduce the cost in re-scheduling the resources and activities from the current solution to the new solution.

D. Optimization in Dynamic Environments with Noise

One of the most important applications of our framework is for optimization problems in dynamic environments with noise. Noise in the real world comes from many different sources, e.g., sensory measurement errors and randomized simulations. One example is the evolutionary structure optimization of neural networks using indirect encoding schemes [83], where noise is inevitable to evaluate the network structure. Different fitness values can be obtained from the same genotype (network structure) due to random initialization of the weights and the multi-modality of the error function.

Another good example is the multi-rover coordination problem in dynamic and noisy environments [70], [69]. The signals that rovers receive from the environment are not reliable due to the noise caused by the rovers' sensors and actuators.

To address optimization problems in noisy environments, many studies have been reported [1], [2]. And many methods have also been proposed to reduce the detrimental effects of noise, such as population sizing [23], fitness averaging and fitness estimation [60], specific selection mechanism [58], and Kalman filtering [66]. The proposed framework is able to search optimal solutions in noisy environments since it does not need to detect the changes that are very difficult to detect in noisy environments. In other words, we may consider the proposed framework as a noise-proof method from the detection-free point of view in dynamic environments with uncertainties.

VII. Conclusions and Future Work

Multiple population approaches are effective methods to locate and track multiple optima in dynamic environments. However, how to effectively use multi-population methods is a difficult question. The difficulty lies in several issues that need to be addressed in dynamic environments, e.g., how to create sub-populations, how to remove redundant individuals, and how to deal with the dynamism that is difficult to detect. Especially for the last issue, there is little research that has been identified to address it [27].

This paper proposes a simple, general, and effective multi-population method to solve DOPs in undetectable dynamic environments. The basic framework employs a single linkage hierarchical clustering method to generate sub-populations whose search radius and size are self-deterministic. The redundancy control is achieved by removing the individuals from the overlapping, overcrowded, or converged sub-populations. Finally, the population diversity is regained by a random immigrants strategy that is triggered when the population diversity decreases to a certain level.

Using this framework to solve DOPs has several advantages. Firstly, there is no need to detect changes throughout the whole run. So, it can be used in any dynamic environments, e.g., detectable, hard-to-detect, or undetectable, with any kind of dynamism, e.g., small step change, large step change, recurrent change, and even chaotic change. Secondly, it can be easily extended into different classes of EAs, such as GA, PSO, and DE. Thirdly, it is simple to implement. There are no complex techniques applied in this framework. Only clustering, redundancy control, and random immigrants procedures are involved in the whole process.

In order to investigate the efficiency of the basic framework, we instantiate it into three different research areas: PSO, GA, and DE. The three corresponding algorithms are called CPSOR, CGAR, and CDER, respectively. In order to effectively use this framework, we systematically carried out experiments on the parameter sensitivity study through the CPSOR algorithm, including the global population size ($gSize$), the sub-population size ($subSize$), the overlapping ratio (β), and the diversity degree (α). This paper also compares the performance of the three algorithms with a set of

algorithms under different test environments. In order to test how the framework performs in hard-to-detect environments, we conducted comparison of CPSOR with the CPSO algorithm, which is a competitive optimizer in detectable dynamic environments.

To summarize the experimental study in this paper, we can draw two conclusions. First, CPSOR is the best performer among all the involved algorithms in detectable environments and also an effective optimizer for DOPs in hard-to-detect environments. Second, CPSOR possesses an outstanding adaptability to different dynamic environments in terms of whether the changes can be detected or not. Generally speaking, this paper has achieved the main objective: to develop a multi-population method in undetectable environments.

In the future, there are several interesting areas to pursue. The first is to improve the performance of the clustering method. The current version can not detect the situation when a single peak is covered by only one individual. More work should be done to resolve this problem. The second is to design effective local search algorithms within this framework in more research areas. The third is to test the performance of the basic framework in completely undetectable dynamic environments. The last one is the application of this framework in real-world problems.

References

[1] D. V. Arnold and H.-G. Beyer, "Local performance of the $(1+1)$-es in a noisy environment," *IEEE Trans. Evol. Comput.*, vol. 6, no. 1, pp. 30–41, 2002.

[2] H.-G. Beyer, "Evolutionary algorithms in noisy environments: Theoretical issues and guidelines for practice," *Comput. Methods Appl. Mech. Eng.*, vol. 186, no. 2-4, pp. 239–267, 2000.

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

[4] S. Bird and X. Li, "Using regression to improve local convergence," in *2007 Congr. Evol. Comput.*, 2007, pp. 592–601.

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

[6] 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, 2006.

[7] P. P. Bonissone, R. Subbu, N. Eklund, and T. R. Kiehl, "Evolutionary algorithms + domain knowledge = real-world evolutionary computation," *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 256–280, 2006.

[8] V. Boschian and A. Pruski, "Grid modeling of robot cells: a memory efficient approach," *J. of Intell. and Robot. Syst.*, vol. 8, no. 2, pp. 201–223, 1993.

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

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

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

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

[13] K. N. Brown and I. Miguel, "Uncertainty and change," in F. Rossi, P. van Beek, and T. Walsh (eds.), *Handbook of Constraint Programming*, Elsevier, Amsterdam, 2006, Chapter 21, pp. 731–760.

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

[15] E. K. Burke, D. G. Elliman, and R. F. Weare, "A university timetabling system based on graph colouring and constraint manipulation," *J. of Res. on Comput. in Education*, vol. 27, pp. 1–18, 1994.

[16] W. Cedeño and V. Rao Vemuri, "On the use of niching for dynamic landscapes," in *IEEE Int. Conf. Evol. Comput.*, 1997, pp. 361–366.

[17] H. Chen, J. C. Miles, and A. S. K. Kwan, "Two-human cardiffga: A new multiple population genetic algorithm," in *2nd Int. Conf. Comput. & Auto. Eng., ICCAE 2010*, vol. 1, 2010, pp. 267–271.

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

[19] I. del Amo, D. Pelta, Gonzáez, and J. lez, "Using heuristic rules to enhance a multiswarm pso for dynamic environments," in *2010 Congr. Evol. Comput.*, 2010, pp. 1–8.

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

[21] A. Elkhyari, C. Guéret, and N. Jussien, "Solving dynamic resource constraint project scheduling problems using new constraint programming tools," in *Practice and Theory of Automated Timetabling IV*, LNCS 2740, 2003, pp. 39–59.

[22] S. S. Ge and Y. J. Cui, "New potential functions for mobile robot path planning," *IEEE Trans. Robot. and Autom.*, vol. 16, no. 5, pp. 615–620, 2000.

[23] D. E. Goldberg, *The Design of Innovation: Lessons from and for Competent Genetic Algorithm*. Kluwer Academic Publishers: Dordrecht, 2002.

[24] H.-J. Goltz, "Combined automatic and interactive timetabling using constraint logic programming," *Int. Conf. on the Practice and Theory of Automated Timetabling*, 2000, pp. 78–95.

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

[26] A. Hertz, "Tabu search for large scale timetabling problems," *Europ. J. of Oper. Res.*, vol. 54, pp. 39–47, 1991.

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

[28] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 2, pp. 241–254, 1967.

[29] M. Kamosi, A. B. Hashemi, and M. R. Meybodi, "A hibernating multi-swarm optimization algorithm for dynamic environments," in *World Congr. on Nature and Biologically Inspired Computing, NaBIC2010*, 2010, pp. 363–369.

[30] L. Kang and G. M. White, "A logic approach to the resolution of constraint in timetabling," *Europ. J. of Oper. Res.*, vol. 112, pp. 322–345, 1999.

[31] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problem*. Springer Press, Heidelberg, 2004.

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

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

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

[35] R. Klein, *Scheduling of Resource Constraints Project*. Kluwer Academic Publishers, Boston, 1999.

[36] C. Li, "Particle swarm optimization in stationary and dynamic environments," Ph.D. Dissertation, Department of Computer Science, University of Leicester, U.K., 2011.

[37] C. Li, M. Yang, and L. Kang, "A new approach to solving dynamic tavelling salesman problem," in *6th Int. Conf. on Simulated Evolution and Learning*, 2006, pp. 236–243.

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

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

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

[41] 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," Tech. Rep., Department of Computer Science, University of Leicester, U.K., 2009.

[42] L. Li, T. Ye, and M. Tan, "Present state and future development of mobile robot technology research," *Robot.*, vol. 24, no. 5, pp. 475–480, 2002.

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

[44] L. Liu and S. R. Ranjithan, "An adaptive optimization technique for dynamic environments," *Eng. Appl. Artif. Intell.*, vol. 23, no. 5, pp. 772–779, 2010.

[45] L. Liu, S. Yang, and D. Wang, "Particle swarm optimization with composite particles in dynamic environments," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 40, no. 6, pp. 1634–1648, 2010.

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

[47] R. I. Lung and D. Dumitrescu, "Evolutionary swarm cooperative optimization in dynamic environments," *Natural Comput.*, vol. 9, no. 1, pp. 83–94, 2010.

[48] E. Masehian and D. Sedighizadeh, "Classic and heuristic approaches in robot motion planning: A chronological review," in *World Academy of Science, Engineering and Technology*, vol. 23, 2007, pp. 101–106.

[49] R. Mendes and A. S. Mohais, "Dynde: a differential evolution for dynamic optimization problems," in *2005 IEEE Congr. Evol. Comput.*, 2005, pp. 2808–2815.

[50] N. Mori, H. Kita, and Y. Nishikawa, "Adaptation to a changing environment by means of the thermodynamical genetic algorithm," in *Parallel Problem Solving from Nature PPSN IV*, LNCS 1141, 1996, pp. 513–522.

[51] R. W. Morrison and K. A. De John, "A test problem generator for non-stationary environments," in *1999 Congr. Evol. Comput.*, 1999, pp. 2047–2053.

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

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

[54] 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, 2006.

[55] A. Passaro and A. Starita, "Particle swarm optimization for multimodal functions: A clustering approach," *J. of Artif. Evol. and Appl.*, vol. 2008, 2008.

[56] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization: An overview," *Swarm Intell.*, vol. 1, no. 1, pp. 33–58, 2007.

[57] H. Richter, "Detecting change in dynamic fitness landscapes," in *2009 Congr. Evol. Comput.*, 2009, pp. 1613–1620.

[58] G. Rudolph, "A partial order approach to noisy fitness functions," in *2001 Congr. Evol. Comput.*, vol. 1, 2001, pp. 318–325.

[59] H. Rudová, T. Müller, and K. Murray, "Complex university course timetabling," *J. of Scheduling*, vol. 14, no. 2, pp. 187–207, 2011.

[60] Y. Sano and H. Kita, "Optimization of noisy fitness functions by means of genetic algorithms using history of search with test of estimation," in *2002 Congr. Evol. Comput.*, vol. 1, 2002, pp. 360–365.

[61] K. Schimmelpfeng and S. Helber, "Application of a real-world university course timetabling model solved by integer programming," *OR Spectrum*, vol. 29, no. 4, pp. 783–803, 2007.

[62] I. L. Schoeman and A. P. Engelbrecht, "A novel particle swarm niching technique based on extensive vector operations," *Natural Comput.*, vol. 9, no. 3, pp. 683–701, 2009.

[63] A. Simoes and E. Costa, "Evolutionary algorithms for dynamic environments:prediction using linear regression and markov chains," in *Parallel Problem Solving from Nature*, 2008, pp. 306–315.

[64] W. M. Spears, *Evolutionary Algorithms: The Role of Mutation and Recombination*. Springer-Verlag, Berlin, 2000.

[65] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous space," *J. of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[66] P. D. Stroud, "Kalman-extended genetic algorithms for search in nonstationary environments with noisy fitness evaluations," *IEEE Trans. Evol. Comput.*, vol. 5, no. 1, pp. 66–77, 2001.

[67] N. D. Thanh, "Solving timetabling problem using genetic and heuristics algorithms," *J. of Scheduling*, vol. 9, no. 5, pp. 403–432, 2006.

[68] R. Thomsen, "Multimodal optimization using crowding-based differential evolution," in *2004 Congr. Evol. Comput.*, vol. 2, 2004, pp. 1382–1389.

[69] K. Tumer and A. Agogino, "Evolving multi rover systems in dynamic and noisy environment," in S. Yang, Y.-S. One, and Y. Jin (eds.), *Evolutionary Computation in Dynamic and Uncertain Environments*, Springer, 2007, Chapter 16, pp. 371–387.

[70] K. Tumer and A. Agogino, "Coordinating multi-rover systems:evaluation functions for dynamic and noisy environments," in *2005 Genetic and Evol. Comput. Conf.*, 2005, pp. 591–598.

[71] R. K. Ursem, "Multinational gas: Multimodal optimization techniques in dynamic environments," in *2nd Genetic and Evol. Comput.Conf.*, 2000, pp. 19–26.

[72] F. van den Bergh, "An analysis of particle swarm optimizers," Ph.D. dissertation, Department of Computer Science, University of Pretoria, South Africa, 2002.

[73] Y. G. Woldesenbet and G. G. Yen, "Dynamic evolutionary algorithm with variable relocation," *IEEE Trans. Evol. Comput.*, vol. 13, no. 3, pp. 500–513, 2009.

[74] J. Xiao, Z. Michalewicz, L. Zhang, and K. Trojanowski, "Adaptive evolutionary planner/navigator for mobile robots," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 18–28, 1997.

[75] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *1998 IEEE Conf. Evol. Comput.*, 1998, pp. 69–73.

[76] S. Yang, "Associative memory scheme for genetic algorithms in dynamic environments," in *EvoWorkshops 2006: Appl. Evol.Comput.*, LNCS 3907, 2006, pp. 788–799.

[77] S. Yang, "Genetic algorithms with memory- and elitism-based immigrants in dynamic environment," *Evol. Comput.*, vol. 16, no. 3, pp. 385–416, 2008.

[78] S. Yang and C. Li, "A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments," *IEEE Trans. Evol. Comput.*, vol. 14, no. 6, pp. 959–974, 2010.

[79] S. Yang and H. Richter, "Hyper-learning for population-based incremental learning in dynamic environments," in *2009 Congr. Evol. Comput.*, 2009, pp. 682–689.

[80] S. Yang and R. Tinos, "Hyper-selection in dynamic environments," in *2008 Congr. Evol. Comput.*, 2008, pp. 3185–3192.

[81] 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, 2005.

[82] 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, 2008.

[83] X. Yao and Y. Liu, "A new evolutionary systems for evolving artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 694–713, 1997.

[84] C. Zheng, L. Li, F. Xu, F. Sun, and M. Ding, "Evolutionary route planner for unmanned air vehicles," *IEEE Trans. Robot.*, vol. 21, no. 4, pp. 609–620, 2005.

Changhe Li received the B.Sc. and M.Sc. degrees in computer science from China University of Geosciences, Wuhan, China in 2005 and 2008, respectively, and the Ph.D. degree in computer science from the University of Leicester, U.K. in July 2011. He is currently a lecturer with the School of Computer Science, China University of Geosciences, Wuhan, China.

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

Shengxiang Yang (M'00) received the B.Sc. and M.Sc. degrees in automatic control and the Ph.D. degree in systems engineering from Northeastern University, Shenyang, China in 1993, 1996, and 1999, respectively. He is currently a Senior Lecturer with the Department of Information Systems and Computing, Brunel University, U.K. He has over 130 publications. He has given invited keynote speeches in several international conferences and co-organised several symposiums, workshops and special sessions in conferences. He serves as the area editor, associate editor or editorial board member for four international journals. He has co-edited several books and conference proceedings and co-guest-edited several journal special issues. His major research interests include evolutionary and genetic algorithms, swarm intelligence, computational intelligence in dynamic and uncertain environments, artificial neural networks for scheduling and real-world applications. He is the chair of the Task Force on Evolutionary Computation in Dynamic and Uncertain Environments, Evolutionary Computation Technical Committee, IEEE Computational Intelligence Society and the founding chair of the Task Force on Intelligent Network Systems, Intelligent Systems Applications Technical Committee, IEEE Computational Intelligence Society.