An Adaptive Multi-Swarm Optimizer for Dynamic Optimization Problems

Changhe. Li changhe.lw@gmail.com
School of Computer Science, China University of Geosciences, Wuhan 430074, China

Shengxiang Yang syang@dmu.ac.uk
Centre for Computational Intelligence (CCI), School of Computer Science and Informatics, De Montfort University, Leicester LE1 9BH, U. K.

Ming Yang yangming0702@gmail.com
School of Computer Science, China University of Geosciences, Wuhan 430074, China

Abstract

The multi-population method has been widely used to solve dynamic optimization problems (DOPs) with the aim of maintaining multiple populations on different peaks to locate and track multiple changing optima simultaneously. However, to make this approach effective for solving DOPs, two challenging issues need to be addressed. They are how to adapt the number of populations to changes and how to adaptively maintain the population diversity in a situation where changes are complicated or hard to detect or predict. Tracking the changing global optimum in dynamic environments is difficult because we cannot know when and where changes occur and what the characteristics of changes would be. Therefore, we need to design algorithms that are able to adapt to changes by taking the challenging issues into account. To address the issues when multi-population methods are applied for solving DOPs, this paper proposes an adaptive multi-swarm algorithm, where the populations are enabled to be adaptive in dynamic environments without change detection. An experimental study is conducted based on the moving peaks problem to investigate the behavior of the proposed method. The performance of the proposed algorithm is also compared with a set of algorithms that are based on multi-population methods from different research areas in the literature of evolutionary computation.

Keywords

Multi-population adaptation, dynamic optimization problems, particle swarm optimization.

1 Introduction

The multi-population method has been widely used in evolutionary computation (EC) to locate and track multiple optima over environmental changes. Multi-population methods, with proper enhancements, have the potential to be efficient methods to solve dynamic optimization problems (DOPs) because they have two advantages. Firstly, they are able to maintain the population diversity at the global level. The population diversity will always be guaranteed as long as populations distribute in different sub-areas in the fitness landscape even though all of them are converging. Secondly, they are able to track a set of optima rather than a single optimum, which will increase the possibility of tracking the changing global optimum. This is because one of the

relatively good optima in the current environment has a high possibility of being the new global optimum in the next environment.

Although many algorithms based on multi-population methods have been proposed to solve DOPs (Cruz et al., 2011; Nguyen et al., 2012), several fundamental challenging issues still remain to be addressed (Li and Yang, 2009; Yang and Li, 2010), e.g., how to determine the number of populations, how to adapt populations to changing environments, when to response to changes, and how to maintain the population diversity without change detection. Some of these issues had not been discussed until the recent work in (Li and Yang, 2009; Yang and Li, 2010), where a hierarchical clustering method is employed within particle swarm optimization (PSO) to create a set of populations that are distributed in different sub-areas in the fitness landscape. The clustering PSO (CPSO) algorithm proposed in (Li and Yang, 2009; Yang and Li, 2010) and the extended version (CPSOR) in (Li and Yang, 2012) attempt to address the challenging issues when multi-population methods are applied. However, far more effort is needed to solve these issues.

To enable multiple populations to adapt to changes in dynamic environments, this paper proposes an adaptive multi-swarm optimizer (AMSO). The motivation is to provide a method to adaptively maintain the population diversity regarding tuning the number of populations for multi-population based algorithms without the assistance of change detection methods in dynamic environments. The work in this paper is based on our previous research (Li and Yang, 2012; Yang and Li, 2010), where all of them are based on the clustering method (Li and Yang, 2009) to create multiple populations. However, there are several natural and significant improvements between this research and our previous research.

Firstly, the parameter settings in the AMSO are not guided by the problem information. For our previous algorithms, some key parameters are set by directly using the information of the problem to be solved, e.g., two key parameters in CPSOR (Li and Yang, 2012) (i.e., the number of individuals $gSize$ and the diversity threshold α), are determined by the number of peaks (optima) in the moving peaks benchmark (MPB) (Branke, 1999) (see Eqs. (2) and (3) in Sect. 2.2.1 for $gSize$ and α, respectively). For the MPB, the number of peaks is available. However, for other problems, such information may be unknown, e.g., the generalized dynamic benchmark generator (GDBG) (Li et al., 2011) containing a huge unknown number of local optima. AMSO does not use such problem information to guide parameter settings due to its diversity maintaining mechanism.

Although the population diversity maintaining mechanism in AMSO is similar to the idea in CPSOR (Li and Yang, 2012) (both increase the population diversity when it drops to a threshold level), the mechanism in AMSO is adaptive while the one in CPSOR is not. In CPSOR, the number of individuals is simply restored to an initial number $gSize$ when the diversity drops to a threshold level α. The total number of individuals and the threshold value α determine the number of populations and the moment to increase diversity, respectively. However, they are fixed and not adaptive to changes particularly in situations with unknown number of peaks (see evidences in Figure 2 later). In this paper, the number of populations and the moment to increase diversity are adaptive to changes in general situations. This enables the proposed algorithm to efficiently use the available fitness evaluations to track more peaks than CPSOR (see evidences in Figure 5 later in this paper), and hence greatly improves the performance.

Secondly, like the CPSOR (Li and Yang, 2012) algorithm, change detection is not needed in AMSO. However, change detection is needed for CPSO (Yang and Li, 2010) to trigger the population increase procedure. The performance of CPSO is seriously affected in hard-to-detect environments (see evidences in Table 9 later in this paper).

Thirdly, in our previous work (Li and Yang, 2012; Yang and Li, 2010), the results of some peer algorithms were collected from the papers where they were proposed, while in this paper, all the peer algorithms are implemented, and they are run and compared based on exactly the same dynamic environments and performance measurements.

The rest of this paper is organized as follows. Sect. 2 reviews some multi-population methods developed in dynamic environments and discusses the difficulties in using multi-population methods for DOPs. The idea of adaptively maintaining the population diversity without change detection and the proposed AMSO are described in Sect. 3. The experimental studies regarding the configuration, working mechanism, and comparison of AMSO with other algorithms on the MPB problem are presented in Sect. 4. Finally, conclusions are given in Sect. 5.

2 Multiple Populations in Dynamic Environments

Due to the advantages of implicit diversity maintaining, multi-population methods have been widely used in the literature of EC for solving DOPs.

2.1 Related Research for DOPs

Branke et al. (2000) proposed a self-organizing scouts (SOS) algorithm that has shown 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.

Inspired by the SOS algorithm (Branke et al., 2000), a fast multi-swarm[1] optimization (FMSO) algorithm was proposed by Li and Yang (2008) 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 since they exclude 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 covers a single peak. Another similar idea of hibernation multi-swarm optimization (HmSO) algorithm was introduced by Kamosi et al. (2010), where a child swarm will hibernate if it is not productive anymore and will be woken up if a change is detected.

In the work in (Jiang et al., 2009), swarms are dynamic and the size of each swarm is small. The whole population is divided into many small sub-swarms. The sub-swarms are re-grouped frequently by using different re-grouping schemes and information is exchanged among sub-swarms. Several accelerating operators are applied to improve the local search ability. Changes need to be detected and adjustments are performed once changes are detected.

An atomic swarm approach was adopted by Blackwell and Branke (2004) to track multiple optima simultaneously with multiple swarms in dynamic environments. An atomic swarm is comprised of charged (or quantum) and neutral particles. The model can be depicted as a cloud of charged particles orbiting a contracting, neutral, PSO nucleus. In their approaches in (Blackwell and Branke, 2006), either charged particles (for the mCPSO algorithm) or quantum particles (for the mQSO algorithm) are used for maintaining the diversity of the swarm, and

[1] The term "swarm" is normally used in PSO, which also denotes "population".

an exclusion principle ensures that no more than one swarm surrounds a single peak. An anti-convergence principle is also introduced to detect new peaks by sharing information among all sub-swarms. This strategy was experimentally shown to be efficient for the MPB.

Borrowing the idea of exclusion from mQSO (Blackwell and Branke, 2006), Mendes and Mohais (2005) developed a multi-population differential evolution (DE) algorithm (DynDE) for DOPs. 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 (del Amo et al., 2010). One of the two rules is to increase the number of quantum particles and 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 by Lung and Dumitrescu (2007). In CESO, two swarms, which use the crowding DE (CDE) (Thomsen, 2004) 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 (Lung and Dumitrescu, 2007). Thereafter, a similar algorithm, called evolutionary swarm cooperative algorithm (ESCA), was proposed by Lung and Dumitrescu (2010) based on the collaboration between a PSO algorithm and an evolutionary algorithm (EA). In ESCA, three populations using different EAs are used. Two of them follow the rules of CDE (Thomsen, 2004) to maintain the diversity. The third population uses the rules of PSO. Three types of collaborative mechanisms were also developed to transmit information among the three populations.

Parrott and Li (2004) developed a speciation based PSO (SPSO), which dynamically adjusts the number and size of swarms by constructing an ordered list of particles, ranked according to their fitness by a "good first" rule, 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 (2006) also proposed an improved version with a mechanism to remove duplicate particles in species. Bird and Li (2006) 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 (2007) introduced another improved version of SPSO using a least square regression (rSPSO). Recently, in order to determine niche boundaries, a vector-based PSO (Schoeman and Engelbrecht, 2009) algorithm was proposed to locate and maintain niches by using additional vector operations.

An algorithm similar to SPSO (Parrott and Li, 2006), called PSO-CP, was proposed in (Liu et al., 2010). In PSO-CP, the whole swarm is partitioned into a set of composite particles by a "worst first" principle, which is opposite to the "good first" rule used in SPSO. The members of each composite particle is fixed by three particles (one pioneer particle and two elementary particles). Inspired by the composite particle phenomenon in physics, the elementary members in each composite particle interact via a velocity-anisotropic reflection (VAR) scheme to integrate valuable information. The idea behind the VAR scheme is to replace the worst particle with a refection point with better fitness through the other two particles. The diversity of each composite particle is maintained by a scattering operator. An integral movement strategy with the aim to promote the swarm diversity is introduced by moving two elementary particles with the same velocity of the pioneer particle after the pioneer particle is updated.

The clustering PSO algorithm proposed by Li and Yang (2009) applies a hierarchical clus-

tering method to divide an initial swarm into sub-swarms that cover different local regions. CPSO attempts to solve some challenging issues associated with multi-population methods, e.g., how to guide particles to move toward different promising sub-regions and how to determine the radius of sub-swarms. Recently, Li and Yang (2012) proposed a general framework for multi-population methods in undetectable dynamic environments based on the clustering method used in (Li and Yang, 2009; Yang and Li, 2010). An algorithm, called CPSOR, was implemented using the PSO technique. The CPSOR algorithm shows a superior performance compared with other algorithms, especially in dynamic environments where changes are hard to detect.

Recently, a new cluster-based differential evolution algorithm was proposed by Halder et al. (2013). In the algorithm, multiple populations are periodically generated by the k-means clustering method, and the number of clusters is decreased or increased by one over a time span according to the algorithm's performance. When a cluster is converged, it is removed with the best individual stored in an external archive. When a change is detected, all the populations are restored to an initial size and re-clustered.

A cultural framework was introduced in (Daneshyari and Yen, 2011) for PSO where five different kinds of knowledge, named situational knowledge, temporal knowledge, domain knowledge, normative knowledge, and spatial knowledge, respectively, are defined. The information is used to detect changes. Once a change is detected, a diversity based repulsion mechanism is applied among particles as well as a migration strategy among swarms. The knowledge also helps in selecting leading particles at personal, swarm, and global levels.

In (Khouadjia et al., 2011), a multi-environmental cooperative model for parallel metaheuristics was proposed to handle DOPs that consists of different sub-problems or environments. A parallel multi-swarm approach is used to deal with different environments at the same time by using different algorithms that exchange information obtained from these environments. The multi-swarm model was tested on a set of dynamic vehicle routing problems.

An adaptive PSO algorithm was proposed in (Rezazadeh et al., 2011). In the proposed algorithm, the exclusion radius and inertia weight are adaptively adjusted by a fuzzy C-means (FCM) mechanism. A local search scheme is employed for the best swarm to accelerate the search progress. When the search areas of two sub-swarms overlap, the worse one is removed. To increase diversity, all normal particles are converted to quantum particles when a change is detected.

2.2 Difficulties in Determining the Number of Populations

The number of populations is a vital factor that affects the performance of an algorithm to locate and track the multiple peaks. However, determining a proper number of populations needed in a specific environment is a very difficult task. This is because the proper number of populations needed is mainly determined by the number of peaks in the fitness landscape. In addition, the distribution and shape of peaks may also play a role in configuring the number of populations. Generally speaking, the more peaks that are in the fitness landscape, the more populations that are needed. Several experimental studies (Blackwell and Branke, 2006; Mendes and Mohais, 2005; Yang and Li, 2010) have shown that the optimal number of populations is equal to the number of peaks in the fitness landscape for the MPB with a small number of peaks (i.e., less than ten peaks). However, evidences in (du Plessis and Engelbrecht, 2012a) show that the optimal number of populations is not equal to the number of total peaks for the MPB with many peaks (i.e., more than ten peaks). Although locating and tracking each peak by a single population is theoretically right, it is not effective and hard to achieve in practice because only limited

computational resources are available. Usually in practice, a relatively low peak in the current environment has a very small chance to become the highest peak in a new environment, and thus, it will waste computational resources available to locate and track each peak by each population.

Intuitively, the optimal number of populations should be relevant to the number of promising peaks. The difficulty is how to figure out the number of such promising peaks in each specific environment. It becomes even harder when the number of peaks fluctuates over changes or in cases where the number of peaks is unknown. In addition, how to determine the search radius for each population is also a difficult issue.

2.2.1 Solutions So Far

To the best of our knowledge, little research regarding the above issue has been done so far due to the difficulties. In the literature of multi-population methods for DOPs, some researchers use pre-defined values for the number of populations and the radius of each population according to their empirical experience. For example, to effectively solve the MPB, ten populations were suggested by Blackwell and Branke (2006) for the mQSO algorithm, the radius was set to 30 in SPSO (Parrott and Li, 2006), rSPSO (Bird and Li, 2007), and HmSO (Kamosi et al., 2010), and to 25 in FMSO (Li and Yang, 2008).

To alleviate the difficulty in manually tuning the two parameters, some problem information is assumed to be known and is used to guide the settings of the two parameters. For example, the exclusion radius in mQSO (Blackwell and Branke, 2006) is set by:

$$r_{excl} = 0.5 * X/peaks^{1/D} \tag{1}$$

where X is the range of the search space, D is the number of dimensions, and $peaks$ denotes the number of peaks in the search space, respectively. Thereafter, several other researchers (del Amo et al., 2010; Mendes and Mohais, 2005) also adopted the same population radius on the MPB problem. In order to get an optimized number of populations, the CPSOR algorithm (Li and Yang, 2012) uses the number of peaks to estimate the total number of individuals ($gSize$) as follows:

$$gSize = 300 \cdot (1 - exp(-0.33 \cdot peaks^{0.5})) \tag{2}$$

The threshold value of α in CPSOR is also determined by:

$$\alpha = 1 - exp(-0.2 \cdot peaks^{0.45}); \tag{3}$$

Although the number of populations varies over the runtime in SOS (Branke et al., 2000), SPSO (Parrott and Li, 2004), and CPSO (Yang and Li, 2010), it is not adaptive as the total number of individuals is fixed during the whole run. One attempt at adapting the number of populations was made by Blackwell (2007) where mQSO as extended to a self-adaptive version, called self-adaptive multi-swarm optimizer (SAMO). The SAMO algorithm starts with a single free swarm (a free swarm is one that is patrolling the search space rather than converging on a peak). The number of free swarms will decrease when some of them are converging (a swarm is assumed to be converging when the neutral swarm diameter is less than a convergence diameter of $2r_{conv}$). If there is no free swarm, a new free swarm is created. On the other hand, a maximum number of free swarms (n_{excess}) is used to prevent too many free swarms being created.

A similar population spawning and removing idea as used in SAMO was introduced and incorporated into a competitive differential evolution (CDE) algorithm (du Plessis and Engelbrecht, 2012b), which is called DynPopDE (du Plessis and Engelbrecht, 2012a), to address DOPs with an unknown number of optima. Different from the population

converging criterion used in SAMO, a simple approach is used in DynPopDE where a population k is assumed to stagnate if there is no difference between the fitness of the best individual of two succussive iterations ($\Delta f_k(t) = |f_k(t) - f_k(t-1)| = 0$). If the stagnation criterion is met, a new free population will be created and the stagnated one will be re-initialized if it is an excluded population. To prevent too many populations crowding in the search space, a population will be discarded when it is identified for re-initialization due to exclusion and $\Delta f_k(t) \neq 0$.

One major issue of the above two adaptive algorithms is that the number of converging populations is un-watched. Therefore, more and more free populations will become converging populations without considering the total number of peaks in the search space, which may be caused by an improper exclusion radius used (i.e., $r_{excl} = 0.5 * X/M^{1/D}$ where M is the number of populations). For example, the average number of populations obtained by DynPopDE (du Plessis and Engelbrecht, 2012a) on a 80 peak MPB instance almost rises to 45 when the number of changes reaches to 100 and still has a growing trend (see evidences in Figure 4 in (du Plessis and Engelbrecht, 2012a) and the fourth graph in Figure 2 in this paper). Thus, here the issue is that the 45 peaks tracked by DynPopDE may not be all promising in terms of the probability of becoming new global optima when a change happens. Thus, the performance would decrease as fitness evaluations are not effectively used due to tracking un-promising peaks. Another issue with the SAMO algorithm (Blackwell, 2007) is that the optimal value for parameter n_{excess} is problem-dependent (Blackwell, 2007; du Plessis and Engelbrecht, 2012a). For example, experimental results in Blackwell (2007) suggests $n_{excess} = 3$ is optimal for the 10-peak MPB instance while $n_{excess} = 5$ is optimal for the 200-peak MPB instance.

2.3 Difficulties in Maintaining Diversity in Dynamic Environments

So far, most EAs developed for DOPs either use some change detection methods (Li, 2004; Lung and Dumitrescu, 2007, 2010; Richter, 2009; Yang and Li, 2010) or predict changes assuming that changes have a pattern (Simoes and Costa, 2008). Once a change has been detected or predicted, different kinds of strategies are applied to increase the diversity, e.g., random immigrants strategies (Li, 2004; Li and Yang, 2009; Lung and Dumitrescu, 2007, 2010; Yang and Li, 2010), or to re-use stored useful information assuming that the new environment is closely related to the current or a previous environment, e.g., memory-based strategies (Branke, 1999). However, in order to use these strategies efficiently, a condition must be applied, that is, changes must be successfully detected. So, here comes a common question: what can these algorithms do if they fail to detect changes? For example, re-evaluating methods will fail to detect changes in a fitness landscape where a part of it changes if all evaluators are in un-changed areas. An example of completely undetectable environments is noisy environments where changes are impossible to be detected by re-evaluation methods because the noise in every fitness evaluation will be misinterpreted as changes.

Maintaining diversity without change detection throughout the run is an interesting topic. In (Grefenstette, 1992), random individuals (called random immigrants) are created every iteration. Three different mutation strategies were designed to control the diversity in (Cobb and Grefenstette, 1993). Sharing or crowding mechanisms in (Cedeño and Rao Vemuri, 1997) were introduced to ensure diversity. A genetic algorithm (GA), called thermodynamical GA (TDGA) (Mori et al., 1996), 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 by Jin and Branke (2005).

Normally, diversity-maintaining is achieved by the following three methods: a) introduce new randomly generated individuals; b) re-activate individuals via mutation operation with a

large probability or a large mutation step; c) allow some individuals to use specially designed rules to maintain diversity rather than to locate the global optimum. However, for the first and second methods, the difficulty is when to increase diversity. For the third method, the problem is how to design such effective rules for maintaining the diversity. In addition, the waste of computational resources for the third method cannot be avoided due to the function of the specialized individuals.

In fact, all the above difficulties regarding diversity maintaining in dynamic environments can be attributed to one fundamental issue, which is how to actively adapt the whole population to changes. As we know, changes in dynamic environments are usually unpredictable. We cannot predict when, where, and what kind of changes will take place. Therefore, to efficiently solve DOPs, an algorithm should be able to actively learn the information about the changes.

3 Multi-Population Adaptation in Dynamic Environments

In order to make populations adaptable to changes, we use a clustering method to create populations. All populations use the same search operator to focus on local search. An overcrowding handling scheme is applied, if certain criteria are satisfied, to remove unnecessary populations and, hence, save computational resources. To find out proper moments to increase diversity without the aid of change detection methods, a special rule is designed according to the drop rate of the number of populations over a certain period of time. In order to introduce a proper number of active individuals that are needed in each specific environment, an adaptive method is developed according to the information collected from the whole populations since the last diversity-increasing point.

3.1 Preparation for Multi-Population Adaptation

Before introducing our population adaptation method, we do some preparatory work, including the introduction of a multi-population generation scheme and an overlapping detection scheme.

3.1.1 Multi-Population Generation

In order to divide the search space into several sub-areas without overlapping, we use the single linkage hierarchical clustering method proposed in (Li and Yang, 2009). In this 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. The distance of two clusters t and s, 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. $M(t, s)$ is formulated as follows:

$$M(t, s) = \min_{i \in t, j \in s} d(i, j) \tag{4}$$

Here, we assume that each peak in the fitness landscape has a cone shape. Therefore, the search area of a population s can be defined as a circle, and accordingly, its radius can be calculated as:

$$radius(s) = \frac{1}{|s|} \sum_{i \in s} d(i, s_{center}), \tag{5}$$

where s_{center} is the central position of population s and $|s|$ is the number of individuals in s. Note that, the best individual of population s will be replaced with s_{center} if s_{center} is better than the best individual of population s in this paper.

Given an initial population pop with a number of individuals uniformly distributed in the fitness landscape, the clustering method works as follows: It first creates a list G of clusters with each cluster containing only one individual. Then, in each iteration, it finds 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 size of a sub-population), and, if successful, combines t and s into one cluster. This iteration continues until each cluster in G contains more than one individual. Finally, the cluster list G is appended to a global population list $plst$, which is empty initially. As a result, we can have a certain number of populations without overlapping with each other.

3.1.2 Overlapping Detection Scheme

Generally speaking, over-crowded populations on a single peak should not be allowed as computational resources are wasted due to redundant individuals searching on the same peak. Overlapped populations searching on different peaks should be allowed to encourage populations to track promising peaks as many as possible. In order to detect whether two populations involve a real overcrowding or overlapping situation, we adopt the following method introduced in (Yang and Li, 2010). If two 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 populations t and s are combined only when $r_{overlap}(t, s)$ is greater than a threshold value β ($\beta = 0.5$ is used in this paper). In the combination process, only $subSize$ best individuals are kept if the number of individuals in the combined population is greater than $subSize$. 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 firstly created by the clustering method rather than their current radius. It should also be noted that this method does not guarantee that every detection is able to identify a real overcrowding or overlapping situation.

In this paper, if the radius of a population is less than a small threshold value ϵ, which is set to $1.e$-4, the population is regarded as converged. A converged population will be removed from the population list $plst$, but the best individual is kept in a list $clst$.

3.2 Multi-Population Adaptation

Diversity lose is one major issue of applying EAs to solve DOPs (Blackwell, 2007). Multi-population adaptation is an alternative approach to addressing the diversity lose issue: it aims to adaptively maintain the population diversity at the multi-population level. To achieve this aim, two issues should be addressed: when to increase the population diversity when it gets low and how many populations (via clustering random individuals in this paper) should be introduced.

3.2.1 The Moment to Increase Diversity

In order to illustrate how to find out a proper moment to increase the population diversity at the multi-population level, we carried out a preliminary experimental study on the MPB with the default settings (see Table 2 in Sect. 4.1.1) based on a non-adaptive algorithm introduced above– the CPSO algorithm (Yang and Li, 2010). The algorithm is informed when a change occurs and at the same time individuals will be restored to the initial size of $gSize$. Then, the clustering method is applied to create populations. In the study, $gSize$ and $subSize$ were set to suggested values of 100 and 7, respectively. Figure 1 presents the progress of the number of populations,

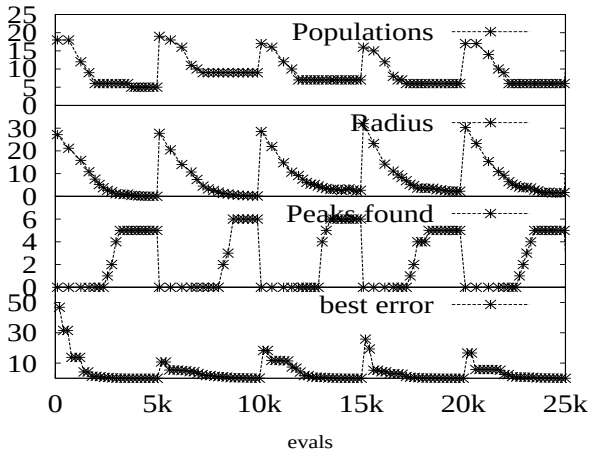


Figure 1: Progress of the number of populations, average radius, peaks tracked, and best error for CPSO on the MPB with default settings (see Table 2).

average radius, the number of peaks tracked, and best error across five changes over a typical run. A peak is assumed to be tracked/found with the MPB if the distance of any individual to the peak is less than 0.01. Here, besides current populations, converged populations are also counted to the number of populations to show the converging behavior in each environment. The best error is the fitness difference between the best solution found so far since the last change and the global optimum.

From the top graph in Figure 1, the number of populations decreases as the search goes on in each environment due to the overlapping detection scheme introduced above. It eventually stays at a certain level in all the five environments. Similar observations can be seen in the curves of average radius and best error. Opposite to the changes with the number of populations, the number of peaks tracked increases as search goes on in each environment. When the number of populations does not change, the whole populations enter a stable status, that is, all populations are converging on different peaks. As a result, no new peaks can be found any more.

This can be validated from the results of the average radius and the number of peaks tracked. From the figure, the corresponding average radius of all populations almost drops to zero in the first two environments after the number of populations converges. For the other three environments, the corresponding average radius also decreases to very small values compared with the initial values. The number of peaks tracked does not increase any more at a certain time after the number of populations becomes stable. This observation is an important clue, which indicates that when the number of populations converges, it is a proper moment to increase diversity. For example, in Figure 1, $evals \approx$3k is such a proper moment to increase the diversity by introducing new individuals as the drop rate of the number of populations almost decreases to zero and the number of peaks being tracked also converges. From that moment, as stated above, there will be no new peaks that can be found if no new individuals are introduced. Therefore, it is necessary to introduce new random individuals to explore new promising peaks no matter whether the environment changes or not.

To find out the proper moment to increase diversity, we monitor the drop rate of the number of populations over a time period (δ). If the rate is less than a threshold value, a certain number

of random individuals (see the following sub-section) are introduced to increase the population diversity. In this paper, the following formula is used to identify the moment to increase the population diversity:

$$(pop(t-\delta) - pop(t))/\delta < 0.002 \tag{6}$$

where $pop(t)$ is the number of populations at time t (measured in the number of fitness evaluations), δ is a new trace gap parameter in this paper. Note that, although the drop rate decreases till zero as overlapped populations are gradually removed for each environment, as shown in Figure 1, we should not use zero as the threshold of the drop rate. There are several reasons. Firstly, the overlapping detection scheme cannot guarantee to detect and remove all overlapped populations as search goes on due to the difficulties stated above. Secondly, the evolutionary status of all populations at the same time may be different since new populations are added repeatedly at each diversity increasing point. This would make it more difficult to detect overlapped populations. Thirdly, populations that have converged will be removed in this paper. This also suggest that the threshold value for the drop rate cannot be zero from the viewpoint of removal of converged populations at unknown time point. It should also be noted that the choice of the threshold of the drop rate also affects the choice of δ, and vice versa (the sensitivity of δ will be studied later in Sect. 4.2.1). Although the threshold of the drop rate should not be zero, obviously it should be a very small value. Based on the above considerations, we use 0.002 as the threshold value for the drop rate in this paper (the choice was made also based on our experimental results). And this would make it easy to perform the sensitivity analysis of δ later.

It should be noted that the monitoring operation on the drop rate of the number of populations will start over once new random individuals are introduced, i.e., populations evolve for at least δ evaluations after a diversity increasing operation. To achieve this, a queue can be used to store relevant information at each iteration, including the number of populations and the number of fitness evaluations. We keep pushing the relevant information into the back of the queue at each iteration. An element is popped out from the queue if the time difference between the front and back elements is larger than δ. This way, the moment to increase population diversity can be identified by checking the difference of the number of populations between the front and back elements. The queue is cleared once new individuals are introduced and the monitoring will start over.

3.2.2 Adaptation of the Number of Populations

Another issue of population adaptation is how many random populations should be introduced when the population diversity needs to be increased. Intuitively, the optimal number of populations needed is related to the number of peaks in the fitness landscape. However, the relationship between them is hard to know even if we have a prior knowledge of the number of peaks. And it will become harder to get such relationship in a situation where the number of peaks fluctuates over changes. To address this issue, we introduce another rule. In order to explain our idea, we again conducted a preliminary experimental study on the MPB with different numbers of peaks over 100 changes with the CPSO algorithm (Yang and Li, 2010) in this section. For CPSO, the same parameter values were used as in Sect. 3.2.1.

Table 1 presents the average number of populations at the time point before a change occurs over 30 runs. From Table 1, the average number of populations linearly increases from 5 to 14 as the number of peaks increases from 5 to 100 even though the same number of individuals ($gSize = 100$) are used in all cases. Therefore, our idea is to use the changes in the number of populations to guide the decision on the number of populations to be introduced and hence to adapt the number of populations to changes where the number of peaks is unknown.

Table 1: The average number of populations obtained by CPSO just before changes occur on the MPB with different numbers of peaks over 30 runs.

$peaks$	5	10	20	30	50	100
Populations	5.0	7.3	9.6	10.8	11.9	14.0

In AMSO, the number of populations to be increased depends on the number of random individuals to be generated for clustering. The number of random individuals to be generated (and hence the number of populations to be increased) is estimated as follows. Whenever a moment of diversity increasing is identified by Eq. (6), we compare the number of populations at the current increasing point ($curPops$) with the number of populations at the last increasing point ($prePops$). If $curPops > prePops$, the total number of individuals (and hence the number of random individuals to be generated) will be increased; otherwise, if $curPops$ is less than $prePops$ by a certain amount $\alpha > 0$ (which is set to 3 in this paper), the number of individuals is decreased in comparison with the number of individuals at the last diversity increasing point. In our experiments, we found that decreasing the number of individuals once $curPops$ is less than $prePops$ sometimes would lead to a wrong decision. This is because a few peaks sometimes become invisible in the fitness landscape when changes occur, which will cause the same effect as the number of peaks really reducing. And once a wrong decision is made to decrease the number of individuals, it will dramatically affect the performance in locating and tracking multiple peaks as only a few peaks can be located and tracked due to a small number of populations. However, a wrong decision to increase the number of individuals will not affect the performance too much as the tracking will not be lost. Therefore, we apply a harder condition on decreasing the number of individuals than that on increasing the number of individuals in this paper.

Another rule that should be noted is that the number of individuals ($gSize$) will not be changed in the following increasing point if it is changed in the current increasing point. This is because we need to give an algorithm enough time to run under a given setting to get relatively reliable feedback. If the number of individuals is changed (the current estimation value is different from the last value), a variable $counter$ will be set to an initial value of one; otherwise, it will be increased by one. There will be no change if $counter$ is equal to one. Therefore, the algorithm will be given a certain time to run under given settings.

After all the conditions are checked, an estimated number of individuals for the following search will be obtained using Algorithm 1. The number of individuals to be increased or decreased is determined by the difference between $prePops$ and $curPops$. The larger the difference between $prePops$ and $curPops$, the larger the number of individuals that will be increased or decreased accordingly, where the number is estimated by a base step of $step$ times $|curPops - prePops|$ (see Steps 5 and 8 in Algorithm 1). This way, the idea is able to adapt the number of populations to changes according to the feedback information of the whole populations. Note that the optimal number of populations for each environment is not guaranteed.

We reiterate that the aim of this paper is to locate and track as many promising peaks as possible via multi-population methods where each population locates a single peak and tracks its movement. The two issues discussed above are challenging as two tradeoffs must be considered. One tradeoff is between the frequency of increasing populations and exploitation, and the other is between the number of populations to be increased and exploitation. Increasing populations frequently or increasing a large number of populations at each increasing moment is helpful to

Algorithm 1 $getNextIndis()$

```
1: if counter = 1 then
2:     nextIndis := preIndis;
3: else
4:     if curPops − prePops > 0 then
5:         nextIndis := preIndis + step × (curPops − prePops);
6:     else
7:         if prePops − curPops > α then
8:             nextIndis := preIndis − step × (prePops − curPops);
9:         end if
10:    else
11:        nextIndis := preIndis;
12:    end if
13: end if
14: if nextIndis = preIndis then
15:    counter := counter + 1;
16:    if curPops > prePops then
17:        prePops := curPops;
18:    end if
19: else
20:    counter := 1;
21:    prePops := curPops;
22: end if
23: return nextIndis;
```

Note: $counter$ is the number of successive increasing points that have the same number of individuals; $nextIndis$ and $preIndis$ are the number of individuals for the next and previous check points, respectively; $curPops$ and $prePops$ are the number of populations in the current and previous increasing points, respectively; $step$ and α are constants with values of 10 and 3, respectively.

explore more promising peaks. However, increasing populations too frequently or increasing too many populations at each increasing moment is harmful for populations to focus on exploitation since there are limit computational resources (i.e., evaluations) available before a change occurs.

3.3 Algorithm Implementation by Particle Swarm Optimization

PSO was first introduced by Kennedy and Eberhart (1995). In PSO, each particle i (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 (Shi and Eberhart, 1998) as follows:

$$v'^d_i = \omega v^d_i + \eta_1 r_1 (x^d_{pbest_i} - x^d_i) + \eta_2 r_2 (x^d_{gbest} - x^d_i) \tag{7}$$

$$x'^d_i = x^d_i + v'^d_i, \tag{8}$$

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. Note that, the maximum velocity of each particle is set to the initial search radius of its swarm.

The PSO algorithm with the $gbest$ model is used in this paper where each particle's neighborhood is defined as the whole swarm. To speed up the local search within the PSO algorithm, we employ a learning method for the $gbest$ particle, which is an improved version of the one used in CPSO (Yang and Li, 2010) by introducing a learning probability. This learning method tries to extract useful information relevant to those potentially improved dimensions of an improved particle to update $gbest$. When a particle, say particle i, gets improved, we iteratively check each dimension d of the $gbest$ particle and replace the dimension with the corresponding dimensional value of particle i with a probability p_d if the $gbest$ particle is improved by doing so. The value of p_d is calculated by $p_d = 1 - |x^d_i - x^d_{gbest}| / \sum_{d=1}^{D} |x^d_i - x^d_{gbest}|$ (see Algorithm 2). The introduction of the heuristic learning probability greatly saves function evaluations. This

Algorithm 2 $gbestLearn(particle\ \vec{x}_i)$

1: **for** each dimension d **do**
2: $p_d = 1 - \frac{|x_i[d]-x_{gbest}[d]|}{\sum_{d=1}^{D}|x_i[d]-x_{gbest}[d]|}$;
3: **end for**
4: **for** each dimension d of $gbest$ **do**
5: **if** rand()$< p_d$ **then**
6: $\vec{x}_{t\_gbest}(a\ temporary\ particle) := \vec{x}_{gbest}$;
7: $x_{t\_gbest}[d] := x_i[d]$;
8: **if** $\vec{x}_{t\_gbest}$ is better than $\vec{x}_{gbest}$ **then**
9: $x_{gbest}[d] := x_{t\_gbest}[d]$;
10: **end if**
11: **end if**
12: **end for**

Algorithm 3 $PSO(\ )$

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. (7) and (8);
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 4 AMSO

Initialize a counter of fitness evaluation $t := 0$;
Create an initial population pop with $gSize$ particles;
Create an empty list $plst$ to store populations;
Create an empty list $clst$ to store the best individuals of converged populations;
Create an empty queue Q to store relevant information (t and $popNo$ (the number of populations));
Clustering the initial population pop;
while stop criteria is not satisfied **do**
  for each population $plst[i]$ **do**
    $plst[i].PSO()$;
  end for
  Put the best individuals of converged populations into $clst$;
  Remove populations that are converged and overcrowded;
  Q.push(t,$|plst(t)|$);
  $\bar{\delta}$:=$Q.back.t$- $Q.front.t$;
  if $\bar{\delta} >= \delta$ & $(Q.front.popNo - Q.back.popNo)/\bar{\delta} < 0.002$ **then**
    $e\_indis = getNextIndis()$;
    if $e\_indis > MAX\_INDIS$ **then**
      $e\_indis = MAX\_INDIS$;
    end if
    if $e\_indis < MIN\_INDIS$ **then**
      $e\_indis = MIN\_INDIS$;
    end if
    Count the number of individuals s_indis;
    $t\_indis := e\_indis - s\_indis - |clst|$;
    if $t\_indis > 0$ **then**
      Create a temporal population t_pop with t_indis random individuals;
      Add the individuals of $clst$ into t_pop;
      Clear $clst$;
      Clustering t_pop;
      Clear Q;
    end if
  end if
  if $\bar{\delta} > \delta$ **then**
    Q.pop();
  end if
end while

Note: e_indis is an estimated number of individuals; MAX_INDIS and MIN_INDIS are the allowed maximum and minimum number of individuals with values of 300 and 70, respectively.

way, the $gbest$ particle is able to learn some useful information from those dimensions of a particle that has been improved.

To implement an adaptive multi-swarm algorithm (AMSO) with the ideas proposed above, we use the improved PSO with learning as a local search method for each population. Algorithm 4 summarizes the framework of AMSO. Initially, populations are obtained by clustering an initial random swarm. In the evolutionary process, all populations use the improved $gbest$ PSO (see Algorithm 3) to locate different optima simultaneously. Then, they undergo the overlapping and convergence check process where redundant populations will be removed. Before

discarding converged populations, the best individuals of them will be saved into a list $clst$ for later use. To increase the population diversity at a proper moment, Eq. (6) is applied every iteration to identify that moment. If the proper moment is found, an expected number of individuals is estimated. After that, the estimated figure is amended if it goes beyond the range of the maximum and minimum number of individuals. Finally, a random immigrants scheme is applied to introduce new random populations, which are obtained by clustering a temporal random population with the estimated number of individuals and the members in $clst$.

4 Experimental Study

In this section, two groups of experiments are carried out to investigate the performance of the AMSO algorithm. The aim of the first group is to investigate the adaptability of AMSO in different perspectives in dynamic environments based on the MPB. In the second group of experiments, twelve multi-population based EAs are selected from the research areas of PSO, DE, GA, and hybrid algorithms. They are mCPSO (Blackwell and Branke, 2006), mQSO (Blackwell and Branke, 2006), SAMO (Blackwell, 2007), SPSO (Parrott and Li, 2006), rSPSO (Bird and Li, 2007), CPSO (Yang and Li, 2010), CPSOR (Li and Yang, 2012), and HmSO (Kamosi et al., 2010) from PSO, DynDE (Mendes and Mohais, 2005) and DynPopDE (du Plessis and Engelbrecht, 2012a) from DE, SOS (Branke et al., 2000) from GA, and ESCA (Lung and Dumitrescu, 2010) from the hybridization of DE and PSO, respectively. Comparison is conducted based on the MPB problem (Branke, 1999).

In order to use exactly the same fitness landscapes across all environmental changes for a fair comparison, all the peer algorithms involved in this paper were carefully implemented and examined according to their origins where they were proposed. Note that, the PSO-CP algorithm has also been implemented, but the results could not be replicated and this algorithm is therefore omitted from the comparison. The implementation of all the involved algorithms will be included in the library of EAs (EALib[2]).

4.1 Experimental Setup

4.1.1 The MPB Problem

The MPB problem, proposed by Branke (1999), has been widely used as a benchmark in the literature of dynamic optimization. Within the MPB problem, the optima can be varied by three features, i.e., the location, height, and width of the peaks. For the D-dimensional landscape, the problem is defined as follows:

$$F(\vec{x},t)=\max_{i=1,\ldots,p}\frac{H_i(t)}{1+W_i(t)\sum_{j=1}^{D}(x_j(t)-X_{ij}(t))^2}, \tag{9}$$

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:

$$\vec{v}_i(t)=\frac{s}{|\vec{r}+\vec{v}_i(t-1)|}((1-\lambda)\vec{r}+\lambda\vec{v}_i(t-1)), \tag{10}$$

[2] Available at http://cs.cug.edu.cn/teacherweb/lichanghe/pages/EAlib.html

where the shift vector $\vec{v}_i(t)$ is a linear combination of a random vector $\vec{r}$ and the previous shift vector $\vec{v}_i(t-1)$ and is normalized to the shift length s. The correlated parameter λ is set to 0, which implies that the peak movements are uncorrelated.

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

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

$$W_i(t) = W_i(t-1) + width\_severity * \sigma \tag{12}$$

$$\vec{X}_i(t) = \vec{X}_i(t)(t-1) + \vec{v}_i(t) \tag{13}$$

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

Note that different from the traditional MPB problem (Branke, 1999), two new features are introduced to make it more difficult to solve in this paper:

- **Changes in the number of peaks**. The number of peaks is allowed to change to evaluate the performance of multi-population methods in terms of the adaptation of the number of populations. If this feature is enabled, the number of peaks changes using one of the following formulas:

$$peaks = peaks + sign \cdot 10 \tag{14a}$$

$$peaks = peaks + sign \cdot rand(5, 25) \tag{14b}$$

$$peaks = rand(10, 100), \tag{14c}$$

  where $sign = 1$ if $peaks <= 10$, $sign = -1$ if $peaks >= 100$, and the initial value of $sign$ is one; $rand(a, b)$ returns a random value in $[a, b]$.

- **Changes in a part of the fitness landscape**. A ratio of changing peaks to the total number of peaks ($cPeaks$) is also introduced. This feature may cause algorithms that are based on change detection to lose their functions.

The default settings and definition of the benchmark problem used in the experiments of this paper can be found in Table 2. The new features introduced above are disabled by default unless explicitly stated otherwise in this paper.

4.1.2 Performance Evaluation

Two performance measures are used in this paper. They are the offline error ($E_{offline}$) (Branke and Schmeck, 2003) and the best-before-change error (E_{BBC}). The offline error is the average of the best error found at each fitness evaluation. The best-before-change error is the average of the best error achieved at the fitness evaluation just before a change occurs.

4.1.3 t-Test Comparison

To compare the performance of two algorithms at the statistical level, a two-tailed t-test with 58 degrees of freedom at a 0.05 level of significance was conducted between AMSO and each peer algorithm. The t-test result is given together with the average score value with superscript letter "w", "l", or "t", which denotes that the performance of AMSO is significantly better than, significantly worse than, and statistically equivalent to its peer algorithm, respectively.

Table 2: Default settings for the MPB, where the term "change frequency (u)" means that the 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
number of peaks ($peaks$)	10
change frequency (u)	5000 function evaluations
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
noise	no
time-linkage	no
number of peaks change	no
S	[0, 100]
H	[30.0, 70.0]
W	[1, 12]
I	50.0

4.1.4 Configurations of AMSO

In AMSO, the number of populations and the moments to increase diversity are adaptive. However, in order to make AMSO adaptable to changes, several non-adaptive parameters are also introduced. Table 3 lists all the parameters for AMSO. Note that constant values for most non-adaptive parameters are made by a systematical experimental study and they are reasonable. For example, the threshold radius value of $1.e$-4 is small enough for checking if a population converges or not. Making the parameters of PSO (ω, η_1, and η_2) adaptive may be helpful in dynamic environments. But, we do not investigate this aspect as it is not the main objective of this paper. To start to run the AMSO algorithm, the initial value of $gSize$ was set to 100 in all experiments unless otherwise stated in this paper. All the results obtained on the MPB problem are averaged over 30 independent runs in this paper.

All the peer algorithms use the suggested configurations from the papers where they were proposed on the MPB problem. Table 4 presents the configurations regarding the population radius and the number of populations for all the involved algorithms. Note that the population radius is not applicable for ESCA and parameter settings of all the peer algorithms for the MPB were adjusted to solve the GDBG benchmark set as necessary. For example, the search radius $r = 3$ was used for the HMSO algorithm with the GDBG benchmark set instead of $r = 30$ with the MPB due to the different search ranges of the two benchmarks.

4.2 Experimental Investigation of AMSO

In this section, the performance of AMSO is investigated with regard to several aspects, including the number of populations in dynamic environments with a variable number of peaks, the

Table 3: Parameters in AMSO, where $gSize = 100$ was used to start the AMSO algorithm.

Parameter	Type	Value
overlapping ratio (β)	Constant	0.5
convergence threshold (ϵ)	Constant	1.e-4
trace gap (δ)	Constant	1,500 $evals$
population adjustment step size ($step$)	Constant	10 individuals
population decrease threshold (α)	Constant	3
maximum individuals (MAX_INDIS)	Constant	300
minimum individuals (MIN_INDIS)	Constant	70
maximum individuals in a sub-pop ($subSize$)	Constant	7
PSO:inertia weight (ω)	Constant	0.6
PSO:acceleration constants (η_1=η_2)	Constant	1.7
initial population size ($gSize$)	Variable	100
number of populations	Adaptive	-
number of total individuals	Adaptive	-
population radius	Variable	-
frequency of diversity increase	Adaptive	-

Table 4: Comparison regarding the configurations of the radius and number of populations for all the involved algorithms, where "variable" denotes that the value of the parameter changes but not in an adaptive way.

Algorithm	Radius	Number of populations
SOS (Branke et al., 2000)	Constant	Variable
DynDE (Mendes and Mohais, 2005)	Constant (31.5: Eq. (1))	Constant (10)
mCPSO (Blackwell and Branke, 2006)	Constant (31.5: Eq. (1))	Constant (10)
mQSO (Blackwell and Branke, 2006)	Constant (31.5: Eq. (1))	Constant (10)
SPSO (Parrott and Li, 2006)	Constant (30)	Variable
rSPSO (Bird and Li, 2007)	Constant (30)	Variable
SAMO (Blackwell, 2007)	Variable	Adaptive
ESCA (Lung and Dumitrescu, 2010)	N/A	Constant (3)
CPSO (Yang and Li, 2010)	Variable	Roughly constant (70/3)
HmSO (Kamosi et al., 2010)	Constant (30)	Variable
CPSOR (Li and Yang, 2012)	Variable	Roughly constant (Eq. (2))
DynPopDE (du Plessis and Engelbrecht, 2012a)	Variable	Adaptive
AMSO, 2013	Variable	Adaptive

sensitivity to the parameter δ, and the ability of locating and tracking multiple peaks, respectively.

4.2.1 Sensitivity Analysis of Parameter δ

From Eq. (6), the frequency of increasing diversity depends on the parameter δ once the threshold of the drop rate is fixed. To make sure that "necessary" population diversity is always guaranteed, the value of δ on the one hand should not be too large as a future change may take place at any time. However, the current check is already a postponed operation, because we have to give enough time for populations to evolve into the converging status in order to achieve a precise estimation. Therefore, the value of δ on the other hand should not be too small. In order to find out a good choice of the value of δ, we carried out an experimental study with AMSO with different values of δ on the MPB problem with different numbers of peaks in this section. Table 5 presents the offline error, the best-before-change error, diversity increasing times per change ($divInc$), and the number of peaks tracked of AMSO over 30 runs.

Table 5: The offline error, the best-before-change error, the number of diversity increasing per change ($divInc$), and the number of peaks tracked by AMSO for different values of δ on the MPB, where w and t denote the best score results in bold font are significantly better than and statistically equivalent to the other results, respectively. The suggested configuration for AMSO and the default settings for the MPB in Table 2 except $peaks$ were used.

$peaks$	δ	100	300	500	700	1000	1500	2000	2500	3000	3500	4000	4500
1	$E_{offline}$	3.91^{w}	3.96^{w}	4.61^{w}	4.56^{w}	3.66^{w}	2.45^{w}	**1.6**	1.82^{t}	1.69^{t}	2^{w}	1.87^{t}	1.93^{w}
	E_{BBC}	**0.0407**	0.0691^{w}	0.117^{w}	0.168^{w}	0.22^{w}	0.13^{w}	0.098^{w}	0.114^{w}	0.0687^{w}	0.0652^{w}	0.0479^{t}	0.0542^{t}
	$divInc$	5.29	4.35	3.17	2.68	2.45	2.11	1.92	1.71	1.53	1.44	1.36	1.28
	$tPeaks$	**0.99**	0.981	0.97	0.955	0.941	0.965	0.974	0.971	0.981	0.982	0.986	0.984
2	$E_{offline}$	2.72^{w}	2.55^{w}	2.6^{w}	2.47^{w}	**2.2**	2.48^{w}	2.53^{w}	2.76^{w}	2.91^{w}	2.81^{w}	2.6^{w}	2.77^{w}
	E_{BBC}	0.791^{w}	0.745^{w}	0.611^{t}	0.66^{t}	**0.503**	0.707^{t}	0.96^{w}	1.18^{w}	1.41^{w}	1.24^{w}	1.18^{w}	1.28^{w}
	$divInc$	5.2	4.15	3.25	2.71	2.28	1.74	1.66	1.5	1.36	1.22	1.14	1.06
	$tPeaks$	1.76	1.79	1.79	1.79	**1.81**	1.78	1.58	1.5	1.46	1.47	1.45	1.42
7	$E_{offline}$	1.32^{w}	1.38^{w}	1.26^{t}	1.33^{w}	**1.24**	1.37^{w}	1.4^{w}	1.35^{w}	1.4^{w}	1.38^{w}	1.35^{w}	1.39^{w}
	E_{BBC}	0.223^{w}	0.318^{w}	0.203^{t}	0.274^{w}	**0.153**	0.231^{w}	0.238^{w}	0.212^{t}	0.241^{w}	0.267^{w}	0.248^{t}	0.298^{w}
	$divInc$	5.49	4.2	3.62	2.88	2.36	1.69	1.38	1.21	1.08	1.01	0.94	0.862
	$tPeaks$	5.95	5.93	6.06	6.01	6.34	6.3	6.28	6.28	**6.35**	6.24	6.31	6.25
10	$E_{offline}$	1.22^{t}	1.21^{t}	**1.21**	1.24^{t}	1.25^{t}	1.4^{w}	1.43^{w}	1.38^{w}	1.36^{w}	1.36^{w}	1.43^{w}	1.39^{w}
	E_{BBC}	0.109^{t}	0.123^{t}	0.108^{t}	0.131^{t}	**0.107**	0.13^{t}	0.196^{w}	0.146^{t}	0.151^{t}	0.175^{w}	0.196^{w}	0.196^{w}
	$divInc$	2.44	2.19	2.19	2.34	1.73	1.53	1.35	1.21	1.12	1.03	0.956	0.892
	$tPeaks$	9.22	9.21	9.21	**9.29**	9.25	9.25	9.12	9.23	9.18	9.07	9.08	9.09
20	$E_{offline}$	2.29^{w}	2.12^{w}	2.16^{w}	2.02^{t}	2.02^{t}	**1.97**	2.04^{t}	2.06^{t}	2.04^{t}	2.06^{t}	2.02^{t}	2.05^{t}
	E_{BBC}	1.47^{w}	1.33^{w}	1.34^{w}	1.2^{w}	1.17^{t}	**1.02**	1.04^{t}	1.03^{t}	1.04^{t}	1.09^{t}	1.08^{t}	1.13^{w}
	$divInc$	1.25	1.1	1.12	1.16	1.08	1.29	1.16	1.06	0.983	0.96	0.897	0.875
	$tPeaks$	10.9	11	11.1	11.7	12.3	13.5	13.7	**13.8**	13.7	13.6	13.3	13.5
30	$E_{offline}$	1.9^{w}	1.96^{w}	1.73^{w}	1.68^{w}	1.57^{w}	**1.48**	1.54^{w}	1.58^{w}	1.59^{w}	1.6^{w}	1.55^{w}	1.58^{w}
	E_{BBC}	0.971^{w}	1.04^{w}	0.792^{w}	0.751^{w}	0.617^{w}	**0.457**	0.458^{t}	0.49^{t}	0.479^{t}	0.541^{w}	0.512^{w}	0.527^{w}
	$divInc$	0.918	0.784	0.904	0.943	0.783	1.11	1.11	1.02	0.945	0.93	0.891	0.857
	$tPeaks$	14.6	13.8	15.2	15.7	16.9	18.9	19.2	19.1	**19.4**	18.7	18.7	18.7
50	$E_{offline}$	2.3^{w}	2.34^{w}	2.27^{w}	2.11^{w}	2.06^{w}	1.95^{t}	1.93^{t}	**1.9**	1.91^{t}	1.95^{w}	1.92^{t}	1.91^{t}
	E_{BBC}	1.33^{w}	1.37^{w}	1.29^{w}	1.08^{w}	1.04^{w}	0.964^{t}	**0.913**	0.928^{t}	0.916^{t}	0.968^{w}	0.93^{t}	0.947^{t}
	$divInc$	0.89	0.755	0.872	0.885	0.884	1.04	1.22	1.09	1.04	0.982	0.943	0.877
	$tPeaks$	16.6	15.9	16.7	18.6	19.6	20.7	**21.9**	21.3	21.4	21.2	21.1	21.1
100	$E_{offline}$	2.41^{w}	2.57^{w}	2.45^{w}	2.43^{w}	2.21^{w}	2.12^{t}	2.07^{t}	2.1^{t}	2.1^{t}	**2.06**	2.15^{w}	2.11^{t}
	E_{BBC}	1.47^{w}	1.65^{w}	1.55^{w}	1.52^{w}	1.26^{w}	1.2^{w}	**1.11**	1.16^{t}	1.16^{t}	1.16^{t}	1.25^{w}	1.22^{w}
	$divInc$	1.07	0.888	0.912	0.917	1	1.13	1.24	1.1	0.962	1.04	0.877	0.893
	$tPeaks$	18.7	17.3	18.1	19	21.6	22.4	**23.4**	22.9	22.8	22.1	21.4	21.4
200	$E_{offline}$	2.47^{w}	2.47^{w}	2.56^{w}	2.43^{w}	2.18^{w}	**1.94**	2.08^{w}	1.95^{t}	2.06^{w}	2.04^{t}	2.06^{t}	2.2^{w}
	E_{BBC}	1.53^{w}	1.55^{w}	1.62^{w}	1.48^{w}	1.26^{w}	**1.04**	1.17^{w}	1.05^{t}	1.17^{w}	1.13^{t}	1.14^{w}	1.28^{w}
	$divInc$	0.736	0.737	0.629	0.65	0.921	1.17	0.968	1.04	0.847	0.808	0.691	0.559
	$tPeaks$	18.1	17.6	17.7	19.1	21.5	**24.1**	22.2	23.1	21.6	21.6	22.4	20

From Table 5, the expected results can be observed, i.e., the choice of δ affects the performance of AMSO. A good choice of δ seems to be instance-dependant. Based on the results, we suggest that a relatively large value of δ should be used for problems with a large number of optima as AMSO achieves small E_{BBC} and $E_{offline}$ errors with a large value of δ in most cases. In this paper, $\delta = 1,500$ is used for the following experiments.

Two interesting results can also be observed from Table 5. Firstly, the average number of diversity increasing per change decreases as the value of δ increases in cases where the number of peaks is less than 30. For the instances with a large number of peaks (e.g., more than 20 peaks), however, there is no such trend compared with the former cases. For example, in the case with 200 peaks, the value of $divInc$ decreases from 0.73 to 0.62 as δ increases from 100 to 500, then it increases to 1.17 when δ reaches 1500 and then the value again decreases as δ

increases. Secondly, the larger the number of peaks that are tracked by AMSO, the better the performance is for AMSO. This is obvious particularly in cases with many peaks. For example, the largest number of peaks tracked by AMSO in the case of 200-peak is 24.1, which corresponds to the smallest offline error and the best-before-change error. The explanation is that the more peaks (promising peaks) that an algorithms can track, the larger probability the algorithm will track the global optimum.

4.2.2 Adaptation in The Number of Populations

Figure 2 presents the comparison of the progress of the number of populations and the offline error between AMSO and other three algorithms (CPSOR, SAMO, and DynPopDE) on the MPB with different numbers of peaks. CPSOR is our previous algorithm and SAMO and DynPopDE are two adaptive algorithms regarding the number of populations. The optimal number of populations for a specific environment depends on the total number of peaks in the fitness landscape. Comparing the results between CPSOR and AMSO on the left graphs, we can see that AMSO shows much better adaptation capability than that of CPSOR. For example, the number of populations obtained by AMSO is slightly more than ten in the 10-peak MPB case, but that number obtained by CPSOR is much larger than ten over the whole run. In the case with 50 peaks, the number of populations achieved by CPSOR is similar to that of AMSO. For the instance with a variable number of peaks using Eq. (14a), CPSOR hardly shows any adaptation regarding the number of populations where the number of populations even grows when the number of peaks drops. The number of populations obtained by AMSO, by contrast, generally changes in synchronization with the change of number of peaks.

Due to the adaptation ability, AMSO shows much better performance than CPSOR in terms of the offline error. In the 10-peak MPB case, the average number of populations generated by CPSOR is about 25, which is much larger than the total number of peaks. Due to limited fitness evaluations for each change interval, too many populations may cause them to be unable to exploit their local areas sufficiently before new populations are introduced. In this case, it can be seen that the offline error of CPSOR is much larger than that of AMSO, which make the average offline error of CPSOR much worse than that of AMSO (see the results in Table 6). The effect of the number of populations on the performance of AMSO and CPSOR can be further seen in the 50-peak MPB case. In the graph, CPSOR and AMSO use a similar number of populations after 250k evaluations, which causes them to achieve similar offline errors as well as the best-before-change errors (see the results in Table 6). Again, in the case of varying number of peaks, the gap between the offline errors of CPSOR and AMSO increases when the peak number reaches the lowest level, due to a larger number of populations generated by CPSOR than that of AMSO.

Comparing the results obtained by SAMO and DynPopDE on the right graphs in Figure 2, although all the three algorithms show the adaptation capability, their behaviors are different. In the 10-peak case, all the three algorithms have similar behavior where the number of populations achieved by SAMO is about ten and that number achieved by DynPopDE is slightly smaller than ten. In the 100-peak case, the number of populations achieved by DynPopDE grows the fastest, followed by SAMO, and both DynPopDE and SAMO still show a growing trend at the end of the run (such behavior of DynPopDE can also be seen in Figure 4 in (du Plessis and Engelbrecht, 2012a) where DynPopDE was proposed), while that number of AMSO converges to about 32 after 200k evaluations. In the case with a variable number of peaks, the number of populations obtained by the three algorithms increases or decreases accordingly when the total number of peaks increases or decreases. However, DynPopDE uses a much larger number of populations

than AMSO and SAMO.

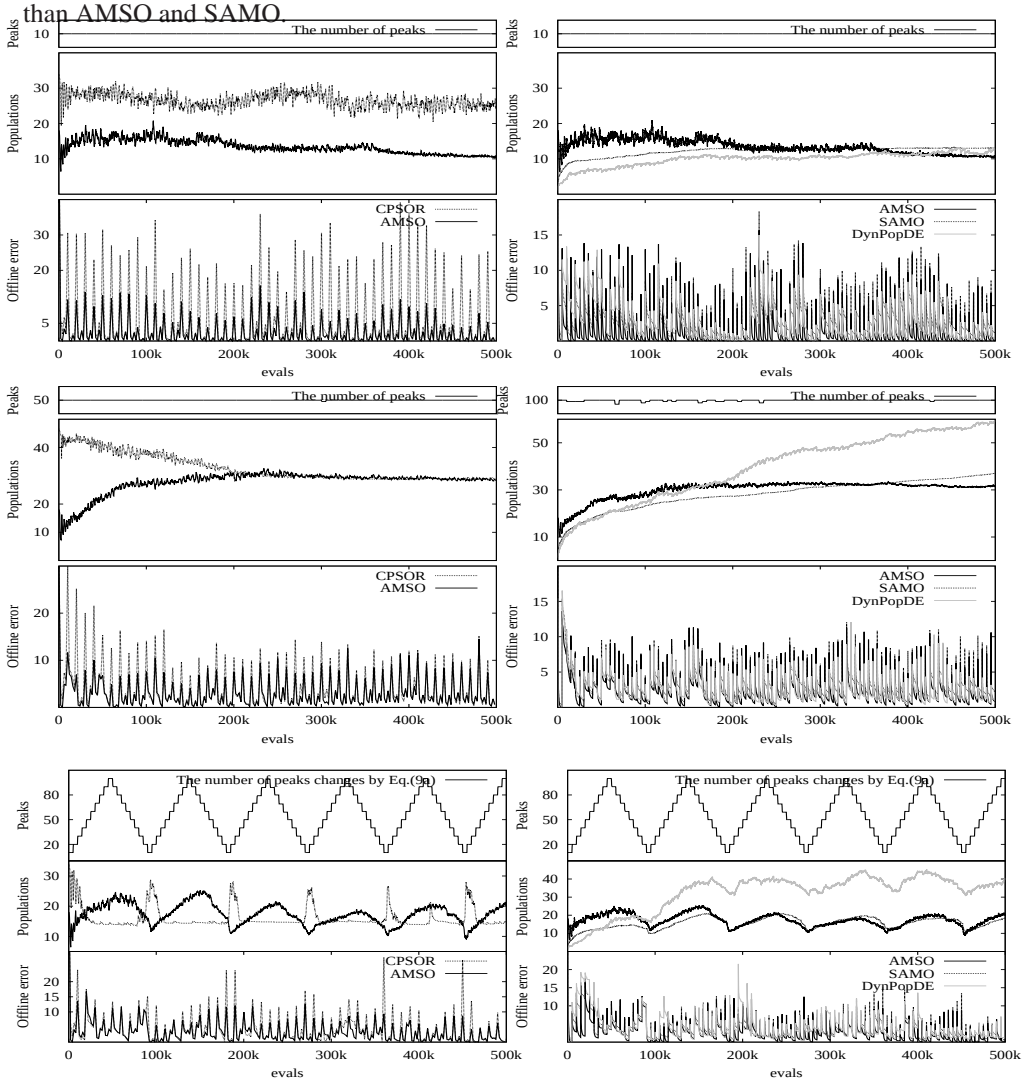
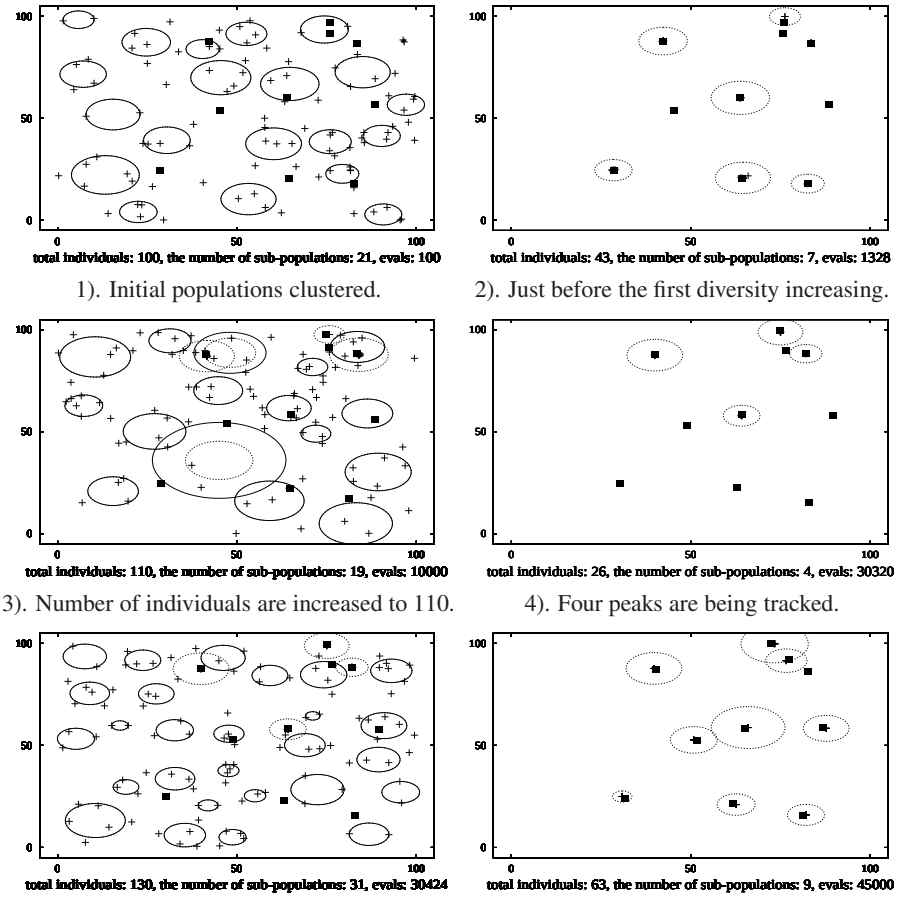


Figure 2: Comparison of the progress of the number of populations and offline error on the MPB with different numbers of peaks between AMSO and CPSOR (left) and between AMSO and other two adaptive algorithms (right).

4.2.3 Visualization of the Behavior of AMSO on Tracking Multiple Peaks

In order to show a clear working mechanism of AMSO, an experimental study was conducted on the MPB in a 2-dimensional search space. Figure 3 presents the results of $pbest$ positions of all particles over six evolving episodes of a typical run, where cross points are particles' $pbest$ positions, black squares are positions of ten peaks, and each circle represents an initial search area defined by Eq. (5) in a population.

In the first episode, 100 random individuals are randomly generated and clustered into 21 populations and there is no overlapping between them. Then, although only 7 populations with



1). Initial populations clustered. 2). Just before the first diversity increasing.

3). Number of individuals are increased to 110. 4). Four peaks are being tracked.

5). 104 individuals are added after episode 4. 6). Nine peaks are being tracked.

Figure 3: Six episodes of the distribution of all populations at different $eval$s, where the crossed points represent $pbest$ positions, the black squares are the ten peaks' locations, and each circle indicates the initial search area of a population by Eq. (5).

43 individuals survive at $eval = 1328$ in episode 2, they cover 7 different peaks, which indicates 7 peaks have been located and tracked. Due to the diversity increasing scheme, the number of individuals is increased to 110 at $eval = 10000$ in episode 3. However, there is overcrowding between the survived populations and the populations clustered from the randomly increased individuals. In episode 4, only 4 peaks are successfully tracked, and 104 individuals are added due to the random immigrants scheme, which makes the total individuals increased to 130. In addition, it can also be seen that most peaks are surrounded by populations again in episode 5. In episode 6, 9 out of 10 peaks are successfully located and tracked by 9 populations with a total of 63 individuals. Note that 2 populations, which are distributed around 2 peaks in the right top corner in episode 6, are overlapped, but they are not combined together due to the overlapping handling scheme in AMSO.

Table 6: The offline error ($E_{offline}$) and best-before-change error (E_{BBC}) for different algorithms on the MPB with different numbers of peaks, where the suggested configurations for AMSO in Table 3 and the default settings of the MPB in Table 2 were used.

peaks		AMSO	CPSOR	CPSO	rSPSO	SPSO	mCPSO	mQSO	SAMO	DynDE	DynPopDE	ESCA	HmSO	SOS
1	$E_{offline}$	2.4 ±0.71	6.1^{w} ±0.84	7.8^{w} ±1	4.1^{w} ±0.71	4.4^{w} ±0.72	41^{w} ±9.5	5.9^{w} ±0.98	4.2^{w} ±0.61	5.2^{w} ±1	**0.44**^{l} ±0.12	6.5^{w} ±0.93	4^{w} ±0.4	2.8^{t} ±1.6
	E_{BBC}	0.13	0.062^{l}	**2.01e-04**^{l}	1.6^{w}	1.9^{w}	39^{w}	3.2^{w}	2.9^{w}	1.3^{w}	0.034^{l}	4.9^{w}	2.2^{w}	1.5^{w}
2	$E_{offline}$	2.5 ±0.37	5^{w} ±0.79	5.3^{w} ±0.45	2.6^{t} ±0.15	2.6^{w} ±0.2	14^{w} ±2.2	6.1^{w} ±0.46	2.9^{w} ±0.17	5.7^{w} ±0.41	**1.2**^{t} ±0.4	7.4^{w} ±0.8	2.6^{t} ±0.43	4.3^{w} ±0.93
	E_{BBC}	**0.71**	1.4^{w}	1.1^{w}	0.93^{w}	0.99^{w}	12^{w}	4.3^{w}	1.7^{w}	4.1^{w}	0.85^{t}	6.5^{w}	1.2^{w}	3.5^{w}
5	$E_{offline}$	**1.6** ±0.28	2.9^{w} ±0.34	4.2^{w} ±0.32	2.5^{w} ±0.29	2.5^{w} ±0.25	7.3^{w} ±1.2	2.6^{w} ±0.24	2.6^{w} ±0.17	1.9^{w} ±0.12	2^{w} ±0.68	13^{w} ±1.8	5.2^{w} ±0.75	6.5^{w} ±0.96
	E_{BBC}	**0.42**	0.66^{w}	1.1^{w}	1.5^{w}	1.5^{w}	6.3^{w}	1.6^{w}	1.6^{w}	1.1^{w}	1.7^{w}	13^{w}	4.4^{w}	6^{w}
7	$E_{offline}$	**1.4** ±0.15	3^{w} ±0.22	4^{w} ±0.28	2.3^{w} ±0.21	2.3^{w} ±0.17	5.3^{w} ±0.54	2.2^{w} ±0.11	2.2^{w} ±0.089	1.5^{w} ±0.064	1.6^{t} ±0.77	14^{w} ±1.8	3.9^{w} ±0.31	7^{w} ±1.5
	E_{BBC}	**0.23**	0.54^{w}	0.63^{w}	1.2^{w}	1.2^{w}	4.4^{w}	1.2^{w}	1.2^{w}	0.69^{w}	1.1^{w}	13^{w}	2.1^{w}	6.2^{w}
10	$E_{offline}$	**1.4** ±0.11	2.6^{w} ±0.2	4.5^{w} ±0.26	3.5^{w} ±0.41	3.6^{w} ±0.47	8.6^{w} ±1.1	2.8^{w} ±0.19	3^{w} ±0.15	1.5^{w} ±0.067	2.3^{w} ±0.79	15^{w} ±1.8	5.1^{w} ±0.31	8.6^{w} ±1.4
	E_{BBC}	**0.13**	0.36^{w}	1.3^{w}	2.2^{w}	2.3^{w}	7.7^{w}	1.7^{w}	2^{w}	0.68^{w}	1.7^{w}	14^{w}	3.6^{w}	7.8^{w}
20	$E_{offline}$	**2** ±0.19	2.6^{w} ±0.3	4^{w} ±0.16	4.3^{w} ±0.38	4.3^{w} ±0.38	8.6^{w} ±0.96	3.4^{w} ±0.24	3.2^{w} ±0.16	2.8^{w} ±0.61	2.3^{w} ±0.27	11^{w} ±1.5	4.2^{w} ±0.12	6.4^{w} ±0.98
	E_{BBC}	1	**1**^{t}	1.6^{w}	3.6^{w}	3.6^{w}	7.9^{w}	2.7^{w}	2.5^{w}	2.2^{w}	1.8^{w}	10^{w}	3.4^{w}	5.7^{w}
30	$E_{offline}$	**1.5** ±0.1	2^{w} ±0.14	3.5^{w} ±0.16	3.9^{w} ±0.25	4^{w} ±0.3	6.4^{w} ±0.72	3.8^{w} ±0.46	2.8^{w} ±0.1	3.1^{w} ±0.3	1.9^{w} ±0.28	9.9^{w} ±1.1	3.9^{w} ±0.11	6.1^{w} ±1.1
	E_{BBC}	**0.46**	0.6^{w}	1.3^{w}	3^{w}	3.1^{w}	5.7^{w}	2.9^{w}	2^{w}	2.5^{w}	1.4^{w}	9.1^{w}	3.3^{w}	5.3^{w}
50	$E_{offline}$	**2** ±0.16	2.4^{w} ±0.12	3.5^{w} ±0.13	4.3^{w} ±0.29	4.3^{w} ±0.31	6.4^{w} ±0.74	3.7^{w} ±0.19	3^{w} ±0.11	3.5^{w} ±0.28	2.1^{w} ±0.24	10^{w} ±1.6	4.1^{w} ±0.11	5.8^{w} ±1.2
	E_{BBC}	**0.96**	0.98^{l}	1.4^{w}	3.3^{w}	3.4^{w}	5.5^{w}	2.8^{w}	2.2^{w}	2.9^{w}	1.5^{w}	9.5^{w}	3.3^{w}	5^{w}
100	$E_{offline}$	**2.1** ±0.16	2.5^{w} ±0.1	3.2^{w} ±0.13	4.3^{w} ±0.3	4.4^{w} ±0.35	6.4^{w} ±0.46	4.2^{w} ±0.33	3.1^{w} ±0.11	3.9^{w} ±0.29	2.2^{t} ±0.33	11^{w} ±1.3	4.1^{w} ±0.11	6.1^{w} ±1
	E_{BBC}	**1.2**	1.2^{t}	1.5^{w}	3.5^{w}	3.5^{w}	5.6^{w}	3.4^{w}	2.4^{w}	3.2^{w}	1.6^{w}	9.6^{w}	3.6^{w}	5.3^{w}
200	$E_{offline}$	**1.9** ±0.17	2.3^{w} ±0.097	2.5^{w} ±0.091	4.5^{w} ±0.38	4.5^{w} ±0.47	6^{w} ±0.85	4.3^{w} ±0.39	2.9^{w} ±0.15	3.8^{w} ±0.35	2^{t} ±0.21	8.5^{w} ±0.7	3.4^{w} ±0.12	5.2^{w} ±0.89
	E_{BBC}	1	1.1^{t}	**0.98**^{t}	3.6^{w}	3.6^{w}	5.2^{w}	3.3^{w}	2.2^{w}	3.2^{w}	1.4^{w}	7.7^{w}	3^{w}	4.4^{w}

4.2.4 Discussion

There is no explicit action when a change occurs because AMSO does not need to detect changes. The adaptive diversity maintaining is achieved just based on the evolving status of all populations rather than the changing information from the environment. Thanks to the ideas proposed in Sect. 3.2, the AMSO algorithm is able to adapt to environmental changes even though it has no knowledge about the changes at all.

4.3 Comparison on the MPB Problem

So far, the working mechanism of AMSO has been investigated. In this section, the performance of AMSO is compared with other twelve algorithms on the MPB with different scenarios.

4.3.1 Effect of Varying the Number of Peaks

Table 6 presents the comparison of all the involved algorithms on the MPB with different numbers of peaks. From the results, AMSO achieves the best offline error and best-before-change error in most cases and its performance is significantly better than that of all other algorithms in

Table 7: The offline error ($E_{offline}$) and best-before-change error (E_{BBC}) for different algorithms on the MPB with different shift severities, where the suggested configurations for AMSO in Table 3 and the default settings of the MPB in Table 2 were used.

s		AMSO	CPSOR	CPSO	rSPSO	SPSO	mCPSO	mQSO	SAMO	DynDE	DynPopDE	ESCA	HmSO	SOS
1	$E_{offline}$	**1.4**	2.6^{w}	4.5^{w}	3.5^{w}	3.6^{w}	8.6^{w}	2.8^{w}	3^{w}	1.5^{w}	2.3^{w}	15^{w}	5.1^{w}	8.6^{w}
		±0.11	±0.2	±0.26	±0.41	±0.47	±1.1	±0.19	±0.15	±0.067	±0.79	±1.8	±0.31	±1.4
	E_{BBC}	**0.13**	0.36^{w}	1.3^{w}	2.2^{w}	2.3^{w}	7.7^{w}	1.7^{w}	2^{w}	0.68^{w}	1.7^{w}	14^{w}	3.6^{w}	7.8^{w}
2	$E_{offline}$	2.2	3.7^{w}	5.4^{w}	4.9^{w}	5^{w}	9.1^{w}	3.3^{w}	3.7^{w}	$\mathbf{2}^{t}$	2.5^{w}	13^{w}	5.9^{w}	11^{w}
		±0.13	±0.19	±0.24	±0.29	±0.27	±0.91	±0.14	±0.12	±0.1	±0.49	±1.1	±0.45	±1.3
	E_{BBC}	**0.22**	0.59^{w}	1.3^{w}	2.8^{w}	2.9^{w}	7.4^{w}	1.8^{w}	2.2^{w}	0.78^{w}	1.7^{w}	12^{w}	3.9^{w}	8.9^{w}
3	$E_{offline}$	2.9	4.5^{w}	6.1^{w}	5.9^{w}	5.9^{w}	10^{w}	3.8^{w}	4.2^{w}	$\mathbf{2.4}^{l}$	2.9^{t}	13^{w}	6.7^{w}	21^{w}
		±0.18	±0.24	±0.19	±0.26	±0.26	±0.69	±0.14	±0.12	±0.1	±0.62	±0.94	±0.38	±4
	E_{BBC}	**0.27**	0.81^{w}	1.3^{w}	3.3^{w}	3.3^{w}	8.2^{w}	2.1^{w}	2.4^{w}	0.88^{w}	1.9^{w}	12^{w}	4.3^{w}	15^{w}
4	$E_{offline}$	3.4	5.5^{w}	6.7^{w}	6.9^{w}	7.1^{w}	12^{w}	4.4^{w}	4.7^{w}	$\mathbf{2.9}^{l}$	3.8^{w}	13^{w}	7.1^{w}	26^{w}
		±0.19	±0.26	±0.21	±0.21	±0.26	±0.63	±0.14	±0.12	±0.13	±0.61	±0.92	±0.54	±3.7
	E_{BBC}	**0.3**	1.1^{w}	1.5^{w}	4^{w}	4.1^{w}	8.9^{w}	2.3^{w}	2.6^{w}	0.91^{w}	2.7^{w}	12^{w}	4.4^{w}	18^{w}
5	$E_{offline}$	3.8	6.1^{w}	7.2^{w}	7.7^{w}	7.8^{w}	13^{w}	5^{w}	5.2^{w}	$\mathbf{3.6}^{l}$	4.4^{w}	13^{w}	7.3^{w}	32^{w}
		±0.16	±0.21	±0.15	±0.23	±0.24	±0.77	±0.14	±0.18	±0.11	±0.54	±0.78	±0.37	±6.4
	E_{BBC}	**0.36**	1.3^{w}	1.5^{w}	4.4^{w}	4.4^{w}	9.9^{w}	2.6^{w}	2.8^{w}	1^{w}	3.2^{w}	11^{w}	4.4^{w}	23^{w}
6	$E_{offline}$	**4.2**	6.6^{w}	7.6^{w}	8.8^{w}	8.9^{w}	14^{w}	5.5^{w}	5.8^{w}	4.3^{t}	4.7^{w}	13^{w}	8^{w}	37^{w}
		±0.15	±0.25	±0.2	±0.21	±0.24	±0.61	±0.13	±0.13	±0.17	±0.58	±0.78	±0.44	±7.2
	E_{BBC}	**0.48**	1.5^{w}	1.7^{w}	5.1^{w}	5.2^{w}	11^{w}	2.9^{w}	3.1^{w}	1.1^{w}	3.3^{w}	11^{w}	5.1^{w}	28^{w}

most cases as well.

Generally speaking, the more peaks in the fitness landscape, the harder it is for an algorithm to locate and track the global optimum. This trend can be observed in the best-before-change error for most algorithms where the E_{BBC} error increases as the number of peaks increases. However, it is interesting to see that the offline error is very large for most algorithms in the 1-peak and 2-peak MPB cases. It seems difficult for these multi-population algorithms to solve the MPB with a few number of peaks. This is because that competition between populations on a single peak becomes more serious as the number of peaks decreases. This would slowdown the search. Therefore, a worse offline error is achieved for these competing models, such as AMSO, CPSOR, CPSO, mQSO, SAMO, SPSO, DynDE, and HmSO.

4.3.2 Effect of Varying the Shift Length

Table 7 shows the comparison of the $E_{offline}$ error and E_{BBC} error of all the algorithms on the MPB with ten peaks under different shift lengths (s). Generally speaking, the difficulty for an algorithm to locate and track a changing optimum will increase as the shift length increases. The larger the shift length, the further a peak moves, and hence, the harder for an algorithm to re-locate and track the new peak. This trend can be observed from the comparison of both $E_{offline}$ and E_{BBC} with different shit lengths for all algorithms except ESCA. The motivation of the ESCA algorithm is to use a swarm with sufficient diversity to re-start a new search for the global optimum whenever a change is detected. A DOP is treated as a series of static problems by the ESCA algorithm and hence the moving distance of peaks will not affect its performance too much. AMSO outperforms our previous algorithm CPSOR on all instances in terms of the t-test results. It also outperforms all the other algorithms except DynDE in cases with $s > 1$.

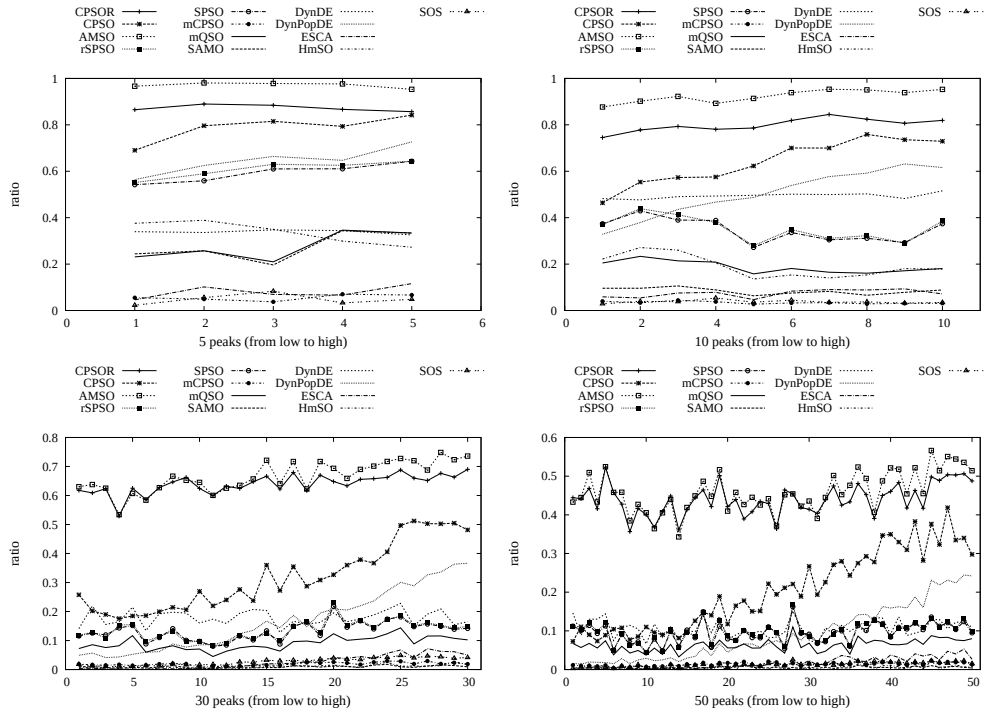


Figure 4: Comparison of the average tracking rate for each peak of all the algorithms on the MPB problem, where the height increases as the peak number increases in the x-axis. The suggested configurations of AMSO in Table 3 and the default settings of the MPB problem in Table 2 were used.

4.3.3 Comparison of the Peaks Tracked

To further investigate the performance of algorithms in tracking optima, Figure 4 presents the results of the average tracking ratio for each peak of all the involved algorithms. In the experiment, we sort all peaks according to their heights when a change occurs and then monitor if any peak is tracked by an algorithm according to the criterion stated above in Sect. 3.2.1. Figure 5 presents the average ratio of peaks found over the total number of peaks on the MPB with different change frequencies. From the results, several observations can be made.

Firstly, the performance of AMSO is the best among all the algorithms. The superiority becomes obvious when the number of peaks is large. The other two clustering based algorithms (CPSOR and CPSO) outperform the remaining algorithms that are based on other multi-population methods. DynDE and DynPopDE also show relatively good results in comparison with other algorithms. Compared with other algorithms, the clustering method is able to locate and track a larger number of optima.

Secondly, comparing the results on the MPB problem with different numbers of peaks, the tracking ratios drop seriously when the number of peaks increases for all the involved algorithms. This is understandable because it will become harder for algorithms to track the global optimum when the number of local optima increases. The offline error may get smaller when the number of peaks increases on the MPB. However, the tracking ratio for each peak decreases

Table 8: The offline error ($E_{offline}$) and the best-before-change error (E_{BBC}) for all the peer algorithms on the MPB problem with ten peaks in different dimensions (D), where the suggested configuration for AMSO and the default settings for the MPB problem in Table 2 except D were used.

D		AMSO	CPSOR	CPSO	rSPSO	SPSO	mCPSO	mQSO	SAMO	DynDE	DynPopDE	ESCA	HmSO	SOS
5	$E_{offline}$	**1.4**	2.6^w	4.5^w	3.5^w	3.6^w	8.6^w	2.8^w	3^w	1.5^w	2.3^w	15^w	5.1^w	8.6^w
		±0.11	±0.2	±0.26	±0.41	±0.47	±1.1	±0.19	±0.15	±0.067	±0.79	±1.8	±0.31	±1.4
	E_{BBC}	**0.13**	0.36^w	1.3^w	2.2^w	2.3^w	7.7^w	1.7^w	2^w	0.68^w	1.7^w	14^w	3.6^w	7.8^w
10	$E_{offline}$	**4.2**	9.6^w	8.9^w	1.1e+02^w	1.1e+02^w	21^w	7.4^w	7.5^w	5.1^w	8.4^w	17^w	13^w	15^w
		±0.5	±1	±0.53	±11	±11	±2.3	±0.36	±0.26	±0.25	±1.4	±4.1	±0.67	±1.5
	E_{BBC}	**2.3**	3.1^w	4.2^w	1.1e+02^w	1.1e+02^w	20^w	6.5^w	6.7^w	4^w	7.7^w	16^w	12^w	14^w
20	$E_{offline}$	**6.5**	37^w	17^w	2.2e+02^w	2.2e+02^w	70^w	14^w	15^w	9.9^w	13^w	47^w	23^w	77^w
		±1.4	±5.6	±1.4	±10	±10	±15	±0.52	±0.63	±0.46	±2.1	±6.9	±0.97	±53
	E_{BBC}	**2.6**	8.1^w	9^w	2.2e+02^w	2.2e+02^w	68^w	13^w	13^w	8.2^w	11^w	46^w	22^w	73^w
30	$E_{offline}$	**8.1**	70^w	25^w	2.9e+02^w	2.9e+02^w	1.9e+02^w	17^w	19^w	11^w	13^w	72^w	45^w	1.4e+02^w
		±1.3	±15	±2.4	±9.5	±9.5	±17	±0.31	±0.69	±0.34	±2.4	±20	±5.3	±85
	E_{BBC}	**3.3**	19^w	10^w	2.9e+02^w	2.9e+02^w	1.8e+02^w	16^w	16^w	10^w	11^w	72^w	44^w	1.4e+02^w
50	$E_{offline}$	25	1.4e+02^w	89^w	4.3e+02^w	4.3e+02^w	2.7e+02^w	33^w	35^w	24^l	**18**^{l}	1.2e+02^w	87^w	1.4e+02^w
		±3.7	±4.4	±5.1	±10	±10	±20	±0.92	±1.9	±0.85	±1.4	±10	±8.8	±47
	E_{BBC}	**9.2**	77^w	22^w	4.3e+02^w	4.3e+02^w	2.7e+02^w	30^w	31^w	21^w	17^w	1.2e+02^w	85^w	1.4e+02^w

obviously as the number of peak increases in this paper.

Thirdly, it is interesting to observe that the AMSO algorithm prefers to track promising peaks with relatively high heights. The CPSO, CPSOR, and DynPopDE algorithms also show the similar behavior. However, this trend cannot be observed in the other algorithms. This is also an advantage of such kind of algorithms.

Fourthly, from the comparison results of algorithms on the MPB with different change frequencies in Figure 5, most algorithms are able to track more peaks when the change frequency increases except mCPSO and ESCA. This is reasonable as algorithms are given more computing resources to re-locate peaks before a change occurs when the change frequency increases. As a result, more peaks should be tracked. Among all involved algorithms, the average ratio of peaks tracked by the AMSO algorithm is the largest across all cases.

4.3.4 Comparison in High Dimensional Spaces

Experiments were carried out to compare the performance of algorithms in high dimensional spaces. Table 8 shows the results of algorithms on the 10-peak MPB instance with the number of dimensions of 5, 10, 20, 30, and 50, respectively. From the results, although the performance of AMSO decreases as the number of dimensions increases, the results are still significantly better than that of all the peer algorithms across all test cases except DynDE and DynPopDE in the case of D=50. The performance of all the algorithms decreases, which is understandable: The difficulty in tracking changing optima will increase as the number of dimensions increases, and thus normally a larger number of evaluations is required. However, the number of evaluations is fixed to 5,000 before a change occurs in all dimensional cases here.

4.3.5 Comparison in Hard-to-Detect Environments

So far, all the comparisons are in the environments where changes are easy to detect. This section presents the comparison of involved algorithms in the environments where changes are hard to detect. This kind of environments is simulated by introducing $cPeaks$ to the MPB (see Sect. 4.1.1). Note that the highest peak (the global optimum) is guaranteed to change in order to test the performance of algorithms in tracking the global optimum in this experimental study.

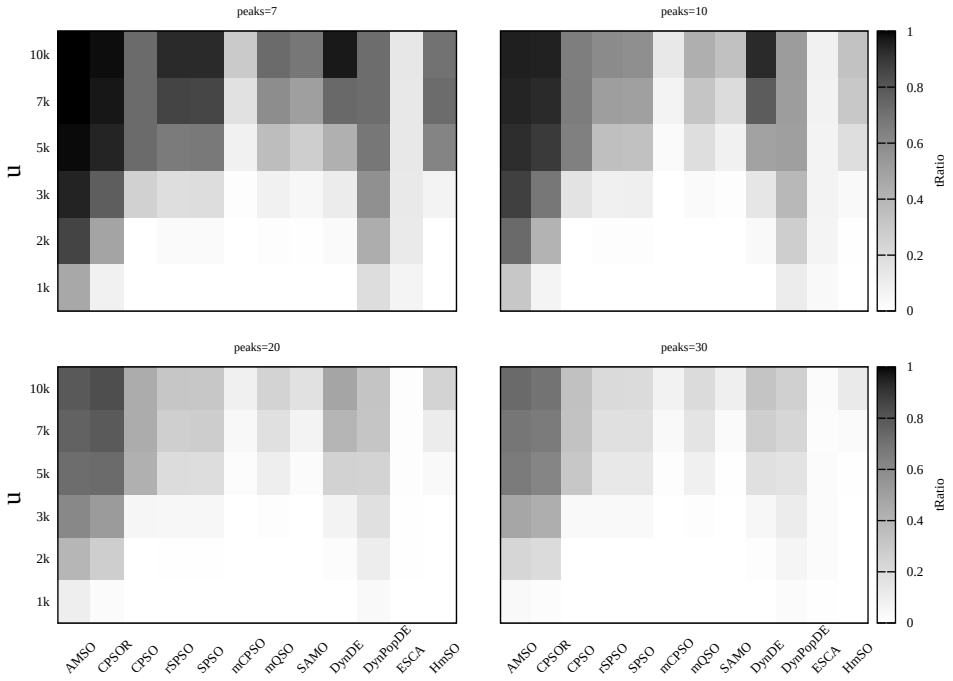


Figure 5: Average tracking ratio of all algorithms on the MPB with different change frequencies, where $tRatio$ is the ratio of the number of peaks tracked to the total number of peaks.

Table 9: The offline error ($E_{offline}$) and the best-before-change error (E_{BBC}) for different algorithms on the MPB problem with different change ratios ($cPeaks$), where the suggested configurations for AMSO in Table 3 and the default settings for the MPB problem in Table 2 were used.

$cPeaks$		AMSO	CPSOR	CPSO	rSPSO	SPSO	mCPSO	mQSO	SAMO	DynDE	DynPopDE	ESCA	HmSO	SOS
0.1	$E_{offline}$	2.2 ±0.71	4^{w} ±0.49	8^{w} ±1	4.9^{w} ±0.51	5^{w} ±0.52	5.7^{w} ±0.54	2.8^{w} ±0.66	2.6^{w} ±0.46	**1.6**^{l} ±0.45	7.7^{w} ±0.93	9.2^{w} ±1.4	8.5^{w} ±2	6.7^{w} ±1.2
	E_{BBC}	1.3	2.1^{w}	7.9^{w}	4.6^{w}	4.6^{w}	5.4^{w}	2^{w}	1.8^{w}	**0.87**^{l}	7.5^{w}	9.2^{w}	8.3^{w}	6.5^{w}
0.3	$E_{offline}$	**1.7** ±0.25	3.2^{w} ±0.32	8.5^{w} ±1.9	6^{w} ±1.2	5.5^{w} ±1.2	8^{w} ±1.3	3.1^{w} ±0.26	3.2^{w} ±0.22	1.9^{w} ±0.3	6.7^{w} ±1.6	18^{w} ±4.5	9.6^{w} ±2.1	11^{w} ±3.2
	E_{BBC}	**0.7**	0.92^{w}	7.4^{w}	5.3^{w}	4.8^{w}	7.2^{w}	2.3^{w}	2.4^{w}	1.2^{w}	5.7^{w}	17^{w}	9^{w}	11^{w}
0.5	$E_{offline}$	1.5 ±0.32	2.5^{w} ±0.25	4.7^{w} ±0.7	2.7^{w} ±0.25	2.7^{w} ±0.28	5.6^{w} ±0.56	2.2^{w} ±0.23	2.2^{w} ±0.23	**1.4**^{l} ±0.23	3.9^{w} ±0.68	11^{w} ±0.9	5.1^{w} ±0.57	6.4^{w} ±1.3
	E_{BBC}	0.54	**0.5**^{l}	3.2^{w}	1.9^{w}	1.9^{w}	4.9^{w}	1.4^{w}	1.4^{w}	0.84^{w}	3.2^{w}	10^{w}	4.4^{w}	5.9^{w}
0.7	$E_{offline}$	**1.5** ±0.17	2.8^{w} ±0.27	3.8^{w} ±0.69	3.9^{w} ±0.44	3.9^{w} ±0.41	6.4^{w} ±0.69	2.6^{w} ±0.26	2.7^{w} ±0.25	1.6^{t} ±0.26	2.9^{w} ±0.74	15^{w} ±2.2	4.7^{w} ±0.69	6.7^{w} ±0.98
	E_{BBC}	**0.52**	0.68^{w}	1.9^{w}	3^{w}	3^{w}	5.6^{w}	1.8^{w}	1.9^{w}	0.99^{w}	2.3^{w}	14^{w}	3.6^{w}	6.1^{w}
0.9	$E_{offline}$	**1.7** ±0.29	2.8^{w} ±0.26	3.7^{w} ±0.27	4.1^{w} ±0.43	4^{w} ±0.44	7.8^{w} ±1.1	3.1^{w} ±0.2	3.2^{w} ±0.18	1.8^{t} ±0.28	2.9^{w} ±0.67	9.9^{w} ±0.92	5^{w} ±0.68	6.5^{w} ±0.69
	E_{BBC}	**0.66**	0.73^{t}	1.2^{w}	3.2^{w}	3.2^{w}	7.2^{w}	2.3^{w}	2.4^{w}	1.1^{w}	2.4^{w}	9.4^{w}	3.9^{w}	5.9^{w}
1	$E_{offline}$	**1.4** ±0.11	2.6^{w} ±0.2	4.5^{w} ±0.26	3.5^{w} ±0.41	3.6^{w} ±0.47	8.6^{w} ±1.1	2.8^{w} ±0.19	3^{w} ±0.15	1.5^{w} ±0.067	2.3^{w} ±0.79	15^{w} ±1.8	5.1^{w} ±0.31	8.6^{w} ±1.4
	E_{BBC}	**0.13**	0.36^{w}	1.3^{w}	2.2^{w}	2.3^{w}	7.7^{w}	1.7^{w}	2^{w}	0.68^{w}	1.7^{w}	14^{w}	3.6^{w}	7.8^{w}

Table 9 shows the comparison of all the algorithms over 30 runs on the MPB where a part of the fitness landscape changes.

Table 10: The offline error ($E_{offline}$) and the best-before-change error (E_{BBC}) for all the peer algorithms on the MPB where the number of peaks changes, where Var1-Var3 are changes following Eq. (14a)-Eq. (14c), respectively.

	Error	AMSO	CPSOR	CPSO	rSPSO	SPSO	mCPSO	mQSO	SAMO	DynDE	DynPopDE	ESCA	HmSO	SOS
Var1	$E_{offline}$	**2.3**	2.8^{w}	4^{w}	6^{w}	5.9^{w}	7.8^{w}	4.5^{w}	3.5^{w}	3.6^{w}	3.7^{w}	13^{w}	4.5^{w}	11^{w}
		±0.25	±0.17	±0.28	±0.55	±0.58	±0.62	±0.27	±0.24	±0.38	±0.26	±1.3	±0.19	±3.2
	E_{BBC}	**1.5**	1.6^{w}	1.9^{w}	5.2^{w}	5.1^{w}	7^{w}	3.7^{w}	2.8^{w}	3^{w}	3.1^{w}	12^{w}	3.5^{w}	9.7^{w}
Var2	$E_{offline}$	**2.9**	3.3^{w}	5^{w}	4.6^{w}	4.9^{w}	7.3^{w}	4.4^{w}	4^{w}	3.5^{w}	4.2^{w}	13^{w}	5.4^{w}	9.4^{w}
		±0.74	±0.63	±1	±0.65	±0.67	±0.89	±0.92	±0.68	±0.81	±0.75	±1.2	±0.69	±4.3
	E_{BBC}	2	**1.9**^{t}	2.6^{w}	3.7^{w}	3.9^{w}	6.3^{w}	3.4^{w}	3.1^{w}	2.9^{w}	3.6^{w}	13^{w}	4^{w}	8.4^{w}
Var3	$E_{offline}$	**2.7**	2.9^{w}	4.5^{w}	4.9^{w}	4.8^{w}	7.4^{w}	4.1^{w}	3.7^{w}	3.4^{w}	4.8^{w}	13^{w}	5.3^{w}	9.6^{w}
		±0.45	±0.29	±0.36	±0.68	±0.66	±0.98	±0.55	±0.32	±0.5	±0.59	±2	±0.4	±3.5
	E_{BBC}	1.7	**1.6**^{t}	2.2^{w}	3.9^{w}	3.7^{w}	6.4^{w}	3.3^{w}	2.8^{w}	2.7^{w}	4.1^{w}	12^{w}	4.1^{w}	8.5^{w}

From Table 9, it can be seen that the performance of several algorithms on the problems with $cPeaks < 1.0$ is worse than that on the problem with $cPeaks = 1.0$. For problems with $cPeaks < 1.0$, only a part of peaks are allowed to change. Therefore, a successful change detection depends on the location of detectors. The change detection will fail when detectors are in unchanged areas of the fitness landscape, and hence algorithms that are based on change detection, such as HmSO and CPSO, do not work well in such dynamic environments.

For HmSO and CPSO, successful change detection is very important for good performance. Normally, the smaller the value of $cPeaks$, the harder it is for the two algorithms to detect changes. From the corresponding results, it can be seen that the errors obtained by the two algorithms gets worse as $cPeaks$ decreases. For the other algorithms that do not heavily rely on change detection, the effect is not as serious as for HmSO and CPSO. AMSO and DynDE show the competitive performance among all the algorithms.

4.3.6 Comparison in Environments with Changing Number of Peaks

Table 10 presents the comparison of all algorithms on the MPB with changing number of peaks. Var1-Var3 are changes corresponding to Eq. (14a) to Eq. (14c), respectively. From the results, the performance of all the algorithms greatly drops on the MPB with this new feature in comparison with the MPB under the default settings. Among the algorithms, AMSO shows the best performance due to the adaptive mechanism. The adaptive algorithm SAMO also achieves better performacne than its non-adaptive version of mQSO. However, such improvement cannot be observed between DynPopDE and DynDE, where DynPopDE is an adaptive version of DynDE but was proposed by different authors.

5 Conclusions

In order to effectively use multi-population methods to solve complex DOPs where changes are complicated or hard to detect, this paper proposes an adaptive multi-population algorithm, which is able to adapt to changes by increasing a "necessary" number of populations at proper moments over changes. The proposed AMSO algorithm employs a single-linkage hierarchical clustering method to generate populations. An overlapping detection scheme is introduced to remove redundant populations during the running process. In this scheme, in order to avoid losing peaks that are being tracked, populations, which are overlapped but cover different peaks, will not be removed. In order to find proper moments to increase the population diversity, a

special technique is proposed by monitoring the drop rate of the number of populations. In order to deal with DOPs with complicated changes, e.g., the number of peaks fluctuates, a novel idea is introduced to figure out a proper number of populations that are really needed. The idea is to compare the number of populations in the current and previous diversity increasing points. If the number of populations in the current increasing point is larger than that in the previous increasing point, the total number of individuals will be increased; Otherwise, the total number of individuals will be decreased. By using these methods, AMSO is able to adaptively maintain the population diversity over changes. Therefore, this adaptive algorithm has basically solved the difficulties in applying multi-population methods for DOPs, including how to determine the number of populations and when to increase diversity over changes. In addition, the population diversity is maintained automatically based on only the information of populations without the assistance of change detection methods.

From the working mechanism investigation and the comparison of a set of algorithms based on multi-population methods on the MPB and the GDBG benchmark, we can draw two conclusions. Firstly, the proposed algorithm is able to adapt to changes by adaptively adjusting the number of populations that are really needed without change detection. Secondly, the performance of the proposed AMSO algorithm is competitive compared with other peer algorithms on the tested problems in terms of both the successful tracking rate and the average score, especially for the fitness landscape with a large number of local optima and for situations where changes are complicated or even hard to detect.

For the future work, an interesting topic is how to adaptively determine the search radius of each population. Although the radius of each population is different in this paper, it is not adaptive to changes.

Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grant 61203306 and the Engineering and Physical Sciences Research Council (EPSRC) of U.K. under Grant EP/K001310/1.

References

Bird, S. and Li, X. (2006). Adaptively choosing niching parameters in a PSO. In *Proceedings of the 2006 Genetic and Evolutionary Computation Conference*, pages 3–10.

Bird, S. and Li, X. (2007). Using regression to improve local convergence. In *Proceedings of the 2007 IEEE Congress on Evolutionary Computation*, pages 592–599.

Blackwell, T. (2007). Particle swarm optimization in dynamic environments. In Yang, S., Ong, Y.-S., and Jin, Y. (eds.), *Evolutionary Computation in Dynamic and Uncertain Environments*, Studies in Computational Intelligence, chapter 2, pages 29–49. Spinger.

Blackwell, T. M. and Branke, J. (2004). Multi-swarm optimization in dynamic environments. In *Applications of Evolutionary Computation*, *Lecture Notes in Computer Science 3005*, pages 489–500.

Blackwell, T. M. and Branke, J. (2006). Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 10(4):459–472.

Branke, J. (1999). Memory enhanced evolutionary algorithms for changing optimization problems. In *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, volume 3, pages 1875–1882.

Branke, J., Kaußler, T., Schmidth, C., and Schmeck, H. (2000). A multi-population approach to dynamic optimization problem. In *Proceedings of the 4th International Conference on Adaptive Computing in Design and Manufacturing*, pages 299–308.

Branke, J. and Schmeck, H. (2003). Designing evolutionary algorithms for dynamic optimization problems. In Ghosh, A. and Tsutsui, S., editors, *Advances in Evolutionary Computing*, Natural Computing Series, pages 239–262. Springer Berlin Heidelberg.

Cedeño, W. and Rao Vemuri, V. (1997). On the use of niching for dynamic landscapes. In *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, pages 361–366.

Cobb, H. G. and Grefenstette, J. J. (1993). Genetic algorithms for tracking changing environments. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 523–530.

Cruz, C., González, J. R., and Pelta, D. A. (2011). Optimization in dynamic environments: A survey on problems, methods and measures. *Soft Computing*, 15(7):1427–1448.

Daneshyari, M. and Yen, G. (2011). Dynamic optimization using cultural based pso. In *Proceedings of the 2011 IEEE Congress on Evolutionary Computation*, pages 509–516.

del Amo, I., Pelta, D. A., and González, J. R. (2010). Using heuristic rules to enhance a multiswarm pso for dynamic environments. In *Proceedings of the 2010 IEEE Congress on Evolutionary Conputation*, pages 1–8.

du Plessis, M. C. and Engelbrecht, A. P. (2012a). Differential evolution for dynamic environments with unknown numbers of optima. *Journal of Global Optimization*, 55(1):73–99.

du Plessis, M. C. and Engelbrecht, A. P. (2012b). Using competitive population evaluation in a differential evolution algorithm for dynamic environments. *European Journal of Operational Research*, 218(1):7–20.

Halder, U., Das, S., and Maity, D. (2013). A cluster-based differential evolution algorithm with external archive for optimization in dynamic environments. *IEEE Transactions on Cybernetics*, 43(3):881–897.

Grefenstette, J. J. (1992). Genetic algorithms for changing environments. In *Proceedings of the 2nd International Conference on Parallel Problem Solving From Nature (PPSN II)*, pages 137–144.

Jiang, Y., Huang, W., and Chen, L. (2009). Applying multi-swarm accelerating particle swarm optimization to dynamic continuous functions. In *Proceedings of the 2nd International Workshop on Knowledge Discovery and Data Mining (WKDD 2009)*, pages 710–713.

Jin, Y. and Branke, J. (2005). Evolutionary optimization in uncertain environments: a survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317.

Kamosi, M., Hashemi, A. B., and Meybodi, M. R. (2010). A hibernating multi-swarm optimization algorithm for dynamic environments. In *Proceedings of the 2010 World Congress on Nature and Biologically Inspired Computing (NaBIC2010)*, pages 363–369.

Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, pages 1942–1948.

Khouadjia, M., Sarasola, B., Alba, E., Jourdan, L., and Talbi, E. (2011). Multi-environmental cooperative parallel metaheuristics for solving dynamic optimization problems. In *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW)*, pages 395–403.

Li, C. and Yang, S. (2008). Fast multi-swarm optimization for dynamic optimization problems. In *Proceedings of the 4th International Conference on Natural Computing (ICNC 2008)*, volume 7, pages 624–628.

Li, C. and Yang, S. (2009). A clustering particle swarm optimizer for dynamic optimization. In *Proceedings of the 2009 IEEE Congress on Evolutionary Computation*, pages 439–446.

Li, C. and Yang, S. (2012). A general framework of multipopulation methods with clustering in undetectable dynamic environments. *IEEE Transactions on Evolutionary Computation*, 16(4):556–577.

Li, C., Yang, S., Nguyen, T. T., Yu, E. L., Yao, X., Jin, Y., Beyer, H.-G., and Suganthan, P. N. (2009). Benchmark generator for CEC'2009 competition on dynamic optimization. Technical report, Department of Computer Science, University of Leicester, U.K.

Li, C., Yang, S., and Pelta, D. (2011). Benchmark generator for CEC'2012 competition on evolutionary computation for dynamic optimization problems. Technical report, School of Computer Science, China University of Geosciences, Wuhan, China.

Li, X. (2004). Adaptively choosing neighborhood bests using species in a particle swarm optimizer for multimodal function optimization. In *Proceedings of the 2004 Genetic and Evolutionary Computation Conference (GECCO 2004)*, pages 105–116.

Liu, L., Yang, S., and Wang, D. (2010). Particle swarm optimization with composite particles in dynamic environments. *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, 40(6):1634–1648.

Lung, R. I. and Dumitrescu, D. (2007). A collaborative model for tracking optima in dynamic environments. In *Proceedings of the 2007 IEEE Congress on Evolutionary Computation*, pages 564–567.

Lung, R. I. and Dumitrescu, D. (2010). Evolutionary swarm cooperative optimization in dynamic environments. *Natural Computing*, 9(1):83–94.

Mendes, R. and Mohais, A. S. (2005). Dynde: a differential evolution for dynamic optimization problems. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, pages 2808–2815.

Mori, N., Kita, H., and Nishikawa, Y. (1996). Adaptation to a changing environment by means of the thermodynamical genetic algorithm. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature (PPSN V)*, *Lecture Notes in Computer Science 1141*, pages 513–522.

Nguyen, T. T., Yang, S., and Branke, J. (2012). Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, 6:1–24.

Parrott, D. and Li, X. (2004). A particle swarm model for tracking multiple peaks in a dynamic environment using speciation. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 98–103.

Parrott, D. and Li, X. (2006). Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Transactions on Evolutionary Computation*, 10(4):440–458.

Rezazadeh, I., Meybodi, M. R., and Naebi, A. (2011). Adaptive particle swarm optimization algorithm for dynamic environments. In *Proceedings of the 2nd International Conference on Advances in Swarm Intelligence (ICSI'11)*, volume Part I, pages 120–129.

Richter, H. (2009). Detecting change in dynamic fitness landscapes. In *Proceedings of the 2009 IEEE Congress on Evolutionary Computation*, pages 1613–1620.

Schoeman, I. L. and Engelbrecht, A. P. (2009). A novel particle swarm niching technique based on extensive vector operations. *Natural Computing*, 9(3):683–701.

Shi, Y. and Eberhart, R. (1998). A modified particle swarm optimizer. In *Proceedings of the 1998 IEEE Conference Evolutionary Computation*, pages 69–73.

Simoes, A. and Costa, E. (2008). Evolutionary algorithms for dynamic environments:prediction using linear regression and markov chains. In *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature (PPSN X)*, *Lecture Notes in Computer Science 5199*, pages 306–315.

Thomsen, R. (2004). Multimodal optimization using crowding-based differential evolution. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, volume 2, pages 1382–1389.

Yang, S. and Li, C. (2010). A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 14(6):959–974.