Multi-Population Methods in Unconstrained Continuous Dynamic Environments: The Challenges

Changhe Li[a], Trung Thanh Nguyen[b], Ming Yang[a], Shengxiang Yang[c], Sanyou Zeng[a]

[a]*School of Computer Science, China University of Geosciences, Wuhan 430074, China*
[b]*School of Engineering, Technology and Maritime Operations, Liverpool John Moores University, Liverpool L3 3AF, U. K.*
[c]*Centre for Computational Intelligence (CCI), School of Computer Science and Informatics, De Montfort University, Leicester LE1 9BH, U. K.*

Abstract

The multi-population method has been widely used to solve unconstrained continuous dynamic optimization problems with the aim of maintaining multiple populations on different peaks to locate and track multiple changing peaks simultaneously. However, to make this approach efficient, several crucial challenging issues need to be addressed, e.g., how to determine the moment to react to changes, how to adapt the number of populations to changing environments, and how to determine the search area of each population. In addition, several other issues, e.g., communication between populations, overlapping search, the way to create multiple populations, detection of changes, and local search operators, should be also addressed. The lack of attention on these challenging issues within multi-population methods hinders the development of multi-population based algorithms in dynamic environments. In this paper, these challenging issues are comprehensively analyzed by a set of experimental studies from the algorithm design point of view. Experimental studies based on a set of popular algorithms show that the performance of algorithms is significantly affected by these challenging issues on the moving peaks benchmark.

Keywords: Multi-population methods, dynamic optimization problems, evolutionary computation

1. Introduction

The key issue of addressing dynamic optimization problems (DOPs) using evolutionary algorithms (EAs) is how to maintain the population diversity while tracking the changing global optimum. The multi-population method has several properties, which make it one of the most commonly used approaches in dynamic optimization. Firstly, the overall population diversity can be maintained at the global level as long as different populations search in different sub-areas in the fitness landscape. Secondly, locating and tracking multiple changing optima simultaneously is possible, and this is very helpful to locate and track the movement of the global optimum. This is because one of the being-tracked local optima may become the new global optimum when changes occur. Thirdly, any scheme based on a single population approach can be easily extended to a multi-population version, e.g., diversity increasing and maintaining schemes [18, 24, 41, 63], memory schemes [8, 62, 68, 75], adaptive schemes [22, 41, 46, 53, 65, 66], multi-objective optimization methods [10], hybrid approaches [26, 43, 44], representation schemes [40], penalty methods [23], immune algorithms [1, 56], predator-prey simulation methods [13], change prediction methods [14, 57], and problem change detection approaches [54].

Although many multi-population algorithms were proposed to solve unconstrained continuous dynamic optimization problems (UCDOPs) [6, 7, 19, 20, 28, 33, 31, 39, 50, 52, 61, 67, 35, 64, 73, 74], most of them have the following limitations. Firstly, many of them use a fixed number of populations or a variable number of populations but with a fixed total size of populations. Secondly, the search area remains the same for each population over the run time. Thirdly, the assistance of change detection methods is needed to detect changes for increasing the diversity. These

Email addresses: changhe.lw@gmail.com (Changhe Li), T.T.Nguyen@ljmu.ac.uk (Trung Thanh Nguyen), yangming0702@gmail.com (Ming Yang), syang@dmu.ac.uk (Shengxiang Yang), sanyouzeng@gmail.com (Sanyou Zeng)

limitations may cause multi-population methods to be unable to adapt to certain dynamic environments, such as environments with an unknown number of optima, partially changed environments, or environments with noise.

The lack of attention on several crucial challenging issues, which should be taken into account in the process of algorithm design, is the cause of the above limitations. These challenging issues are: 1) when to react to changes; 2) how to determine the number of populations; 3) how to determine the search area of each population; and 4) how to avoid using change detection methods. The second and third issues were discussed in the authors' work [35] (extended in [64]). However, the works in [35, 64] have neither shown how the issues would affect the performance of an algorithm nor provided solutions on how to handle the two issues. The other two issues have not been mentioned in the works. Moreover, the proposed algorithms in [35, 64] still rely on change detection methods like many other traditional methods. In the authors' later work [36], a general framework of algorithm design without change detection was proposed. However, the issues were still left untouched except the change detection issue. In their recent work [38], an attempt of solving the first and second issue was made. The performance of the proposed algorithm [38] was improved in comparison with the algorithm proposed in [64], which is also shown in Sect. 4.1.2 in this paper later. However, the third issue was not discussed. Although there are techniques of adjusting the number of populations in [4, 59], no attempt has been made to address this challenge.

All the studies mentioned above focus on developing a specific algorithm rather than looking at a bigger picture of how the aforementioned challenges would affect algorithm performance and how they can be addressed in a generic way. Although the authors' works in [35, 64, 36, 38] partly addressed some of the challenges, there has been no experiment focusing on the investigation of these challenges and their impacts on algorithm performance. This paper is the first one to comprehensively discuss the challenging issues supported by experimental results of a set of existing popular algorithms. This paper also gives comments and suggestions on future algorithm design in dynamic environments to overcome the challenges.

The rest of this paper is organized as follows. Sect. 2 reviews multi-population based algorithms for UCDOPs in three categories. Sect. 3 introduces the selection of a set of benchmark algorithms and the moving peaks benchmark (MPB) problem [8] with a new feature. Three performance evaluation measures are also introduced in Sect. 3. The challenging issues regarding the design of multi-population methods are investigated in Sect. 4. Finally, conclusions of this paper and suggestions on future development of multi-population based EAs for UCDOPs are given in Sect. 5.

2. Multi-population Methods in Dynamic Environments

EAs for DOPs can be divided into two categories [11]: 1) finding/tracking optima over time (algorithms are mainly for DOPs in a continuous space, e.g., the MPB problem [8]) and 2) adapting current solutions against changes (algorithms are mainly for DOPs in a combinatorial space, e.g., scheduling and planning problems). Multi-population methods discussed in this paper belong to the first category and they can be further categorized into three groups in terms of the number of populations used. They are multi-population methods using a fixed number of populations, multi-population methods using a variable number (within a range) of populations, and multi-population methods using an adaptive number of populations, respectively.

2.1. Fixed Number of Populations

Most multi-population based algorithms so far fall into this group. In this group of methods, populations are organized in two different ways. The first way is that all populations use the same search operator and there is no communication among them. The second way is that different populations use different search operators and there are communications between them.

Among the methods using the first way of populations organization, one popular model is the atomic swarm[1] approach, which was proposed by Blackwell and Branke [6] to track multiple optima simultaneously. In their approach, a charged swarm is used for maintaining the diversity of the swarm, and an exclusion principle ensures that no more than one swarm surrounds a single peak. In an algorithm, called multi-swarm optimization with quantum (mQSO) in [7], anti-convergence is introduced to detect new peaks by sharing information among all sub-swarms. This strategy was experimentally shown to be effective for the MPB problem [8]. An enhanced version of mQSO was proposed in

[1]The term "swarm" is normally used to refer to a "population" in particle swarm optimization (PSO).

[20] by applying two heuristic rules to further enhance the diversity of mQSO. Borrowing the idea of exclusion from [6], Mendes and Mohais developed a multi-population differential evolution (DE) algorithm (DynDE) [45] to solve the MPB problem. In their approach, a dynamic strategy for the mutation factor F and probability factor CR in DE was introduced. A fuzzy C-means clustering technique was used to generate populations in [47], where a memory-based crowding archive method was also introduced to solve DOPs.

The representative model for the second way of organizing populations is a model called collaborative evolutionary swarm optimization (CESO), proposed in [43]. In CESO, two swarms, which use the crowding DE (CDE) [60] and PSO model, respectively, cooperate with each other by a collaborative mechanism. The CDE swarm is responsible for preserving diversity while the PSO swarm is used for tracking the global optimum. The competitive results were reported in [43]. Thereafter, a similar algorithm, called evolutionary swarm cooperative algorithm (ESCA), was proposed in [44] based on the collaboration between a PSO algorithm and an EA. In ESCA, three populations using different EAs were used. Two of them follow the rules of CDE [60] 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. Recently, a similar model to CESO [43] and ESCA [44], called cooperative dual-swarm PSO (CDPSO), was proposed [74]. CDPSO adopts a dual-swarm structure to maintain the swarm diversity and track the changing optima. Two different population topologies were used in two sub-swarms in [73], where the two sub-swarms exchange their best particles at checkpoints. One sub-swarm is used for searching the global optimum and the other is responsible for searching local optima and maintaining diversity. A similar dual-swarm structure with multiple strategies was also used in [21].

A cultural framework was introduced in [19] for PSO, where it defines five different kinds of knowledge, named situational knowledge, temporal knowledge, domain knowledge, normative knowledge, and spatial knowledge, respectively. The knowledge is used to detect changes. Once a change is detected, a diversity based repulsion mechanism is applied among particles and a migration strategy is triggered among swarms.

2.2. Variable Number of Populations

Different from the above algorithms which use a fixed number of populations created randomly in the whole fitness landscape, another way to create populations is to split off from a main population or to cluster a main population into a set of small populations.

A famous early "splitting" model is the self-organizing scouts (SOS) algorithm proposed by Branke *et al.* [9]. In SOS, the whole population is composed of a parent population that searches through the entire search space and a number of 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. The size of each population is re-adjusted according to its relative quality defined in [9]. Inspired by a forking method, a multi-swarm algorithm was proposed in [61]. Similar to the SOS algorithm, in the multi-swarm algorithm, a large main swarm is responsible for continuously exploring new peaks and a number of smaller child swarms, split off from the main swarm, are used to track the achieved peaks during the run.

Inspired by the SOS algorithm [9], a fast multi-swarm optimization (FMSO) algorithm was proposed [34] 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 changes occur, and a group of child swarms are used to search for the local optima in their own search areas, which are determined by a predefined search radius. There is no overlap among child swarms since they exclude from each other. If the distance between two child swarms is less than their radius, then the worse swarm is removed. This guarantees that no more than one child swarm covers a single peak. Another similar idea is the hibernation multi-swarm optimization (HmSO) algorithm, introduced in [30], where a child swarm hibernates if it is not productive anymore and wakes up when a change is detected. Recently, a similar hibernation scheme was employed in [70, 69], in which competitive results were reported in comparison with a set of algorithms.

For the clustering based methods, a representative model is the Speciation-based PSO (SPSO) developed by Parrott and Li [39, 50]. The model dynamically adjusts the number and size of swarms by constructing an ordered list of particles, ranked according to their fitness, with spatially close particles joining a particular species. At each generation, SPSO aims to identify multiple species seeds within a swarm. Once a species seed has been identified, all the particles within its radius are assigned to that same species. Parrott and Li also proposed an improved version with a mechanism to remove duplicate particles in species in [51]. In [2], Bird and Li developed an adaptive niching PSO

(ANPSO) algorithm which adaptively determines the radius of a species by using the population statistics. Based on their previous work, Bird and Li introduced another improved version of SPSO using a least square regression (rSPSO) in [3]. In order to determine niche boundaries, a vector-based PSO [55] algorithm was proposed to locate and maintain niches by using additional vector operations.

Another popular clustering based algorithm is the clustering PSO (CPSO) algorithm, proposed by Li and Yang [35, 64]. CPSO applies a hierarchical clustering method to divide an initial swarm into sub-swarms that cover different local regions. Based on their previous works in [35, 64], Li and Yang [36] proposed a general framework for multi-population methods in undetectable dynamic environments in which random individuals are added when the number of individuals drops blow a pre-defined level. A clustering PSO with random immigrants (CPSOR) was implemented in [36]. Recently, a cluster-based dynamic DE algorithm with external archive (CDDE_Ar) was proposed in [25], where the *k-means* clustering method is used to create populations when changes are detected. The number of populations is adjusted regularly with a certain time span based on the performance of the algorithm.

2.3. Adaptive Number of Populations

Different from the above methods, which start with a fixed number of populations without considering the relationship between the optima number of populations and the number of peaks, a few attempts have been made to adapt the number of populations to the changing number of peaks over changes.

The first attempt was made in [4], where the mQSO algorithm [7] was extended to a self-adaptive version, called self-adaptive multi-swarm optimizer (SAMO). SAMO starts with a single free swarm (with a small, fixed number of individuals) that patrols the search space rather than converges on a peak. Free swarms will transform themselves to normal swarms when they are converging (a swarm is assumed to be converging when its neutral swarm diameter is less than a convergence diameter). 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 dynamic population DE (DynPopDE), which addresses DOPs with an unknown number of optima, was proposed in [59]. Different from the population converging/stagnating criterion used in [4], in DynPopDE, a population k is assumed to stagnate if there is no difference between the fitness of the best individuals at two successive iterations ($\Delta f_k(t) = |f_k(t) - f_k(t-1)| = 0$). If the stagnation criterion is met, a new free population (same as the "free swarm" in SAMO [4]) is created and the stagnated one is re-initialized if it is an excluded population. To prevent too many populations from getting crowded in the search space, a population is discarded if it is set for re-initialization due to exclusion and $\Delta f_k(t) \neq 0$.

The algorithms SAMO [4] and DynPopDE [59] do not monitor the number of converging populations. As a result, more and more converging populations are formed over time from free populations without considering the total number of peaks in the search space. To address this limitation, a new approach, adaptive multi-swarm optimiser (AMSO), was proposed [38], where the number of populations is adjusted according to the differences of the number of survived populations between two successive "checking points", which are moments when the drop ratio of the number of populations decreases to a small value. This way, the number of populations is able to adapt to changes. The results in [38] showed that AMSO has a very competitive performance in comparison with other twelve algorithms on the MPB problem [8].

Note that, the literature review of PSO takes the major part of this section. This is because that published studies on multi-population methods for UCDOPs so far are mainly from the research area of PSO.

2.4. Other ways to classify multi-population methods

Besides classifying multi-population methods based on the number of populations used, we can also use the following classification criteira:

- The way to create populations.
  - Random initialization: Populations are created randomly across the whole search space at the beginning of the run [6, 7, 45, 47, 20, 43] or during the runtime [4, 59].
  - Clustering-based approaches: Populations are generated by clustering a random population [50, 51, 2, 3, 35, 64, 36, 38].

  - Splitting-off approaches: Populations are created by splitting off from a main population from generation to generation [9, 61, 34, 30].

- The role of populations.
  - All populations play the same role: All populations aim to explore promising peaks and exploit peaks [3, 7, 20, 45, 4, 59, 51, 36, 38, 64].
  - Different populations may have different roles: A part of populations are responsible for exploring new peaks and the others are for exploiting peaks that have been explored [9, 61, 34, 30, 43, 44, 21, 73, 74].

- The relationship between populations.
  - Competitive relationship: No more than one population is allowed to search in the same area and exclusion rules are normally applied [6, 7, 45, 20, 30, 64, 36, 38].
  - Collaborative relationship: Populations are allowed to search in the same area and information can be shared between each other [43, 44, 21, 73, 74].
  - Mixed relationship: Populations that play different roles cooperate with each other, but populations that play the same role compete with each other [9, 61, 34, 30].
  - No relationship: No rules are applied to populations [50, 51, 2, 3].

- Diversity handling.
  - Diversity maintaining: Diversity is maintained within each population by exclusion rules applied only to a certain type of individuals [6, 7, 45, 43, 44, 21, 9, 34, 59].
  - Diversity regaining: Diversity is regained by re-initialization when it drops to a certain level or a certain type of conditions are met [50, 51, 2, 3, 35, 64, 36, 38, 30].

Compared to the above classification methods, the one adopted in this paper highlights the challenge of determining the number of populations better. It also reflects the trend of the development of multi-population methods in dynamic environments.

2.5. Examples of real-world applications

This subsection briefly provides some examples of real-world applications to which multi-population methods can be applied. Readers are referred to [48, 49] for a comprehensive list of real-world applications solved by evolutionary dynamic optimization methods.

A contaminant source identification problem in water distribution networks is a nonlinear programming problem. The search for the location and the time history of the contaminant is carried out according to the observed data up to the current time. The difficulty in identifying the contaminant source is that solutions are not unique [71], and we do not know which solution is the correct solution, i.e., the solution representing the actual contamination source, according to the observed data. Therefore, multi-population methods can be used to locate multiple optima solutions in a hope that among the multiple optimal solutions found by the multiple populations, one would be the correct solution. Multi-population methods can also be used in other areas, such as optimizing the cluster center in subtractive clustering [16], searching for the global threshold of image [27], training perceptrons in predicting outcomes of construction claims [12], searching for the optimal set of weights in feed forward neural networks [58], predicting financial distress [15], predicting stock prices and direction [17], and finding multi-solutions for multi-layer ensemble pruning [72].

3. Benchmark Problem and Evaluations

3.1. The Moving Peaks Benchmark

The MPB problem [8] has been widely used as a benchmark in dynamic environments. A peak can be varied in three aspects: its location, height and width. For a D-dimensional landscape, the problem is defined as follows:

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

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:

$$\vec{v}_i(t) = \frac{s}{\left|\vec{r} + \vec{v}_i(t-1)\right|}((1-\lambda)\vec{r} + \lambda\vec{v}_i(t-1)), \tag{2}$$

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. A change of a single peak can be described as follows:

$$H_i(t) = H_i(t-1) + height\_severity \cdot \sigma \tag{3}$$

$$W_i(t) = W_i(t-1) + width\_severity \cdot \sigma \tag{4}$$

$$\vec{X}_i(t) = \vec{X}_i(t)(t-1) + \vec{v}_i(t) \tag{5}$$

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

Due to the simplicity of the MPB problem, many algorithms have shown good performance in tracking the global optimum, such as, mQSO [7], SPSO [51], rSPSO [3], CPSO [64], CPSOR [36], AMSO [38] and SAMO [4] from PSO, DynDE [45] and DynPopDE [59] from DE, SOS [9] from genetic algorithms (GAs), ect. In this paper, we introduce a new feature into the MPB problem. The number of peaks is allowed to change to evaluate the adaptability of multi-population methods. If this feature is enabled, the number of peaks changes by one of the following formulas:

$$peaks = peaks + sign \cdot 10 \tag{6a}$$

$$peaks = peaks + sign \cdot rand(5, 25) \tag{6b}$$

$$peaks = rand(10, 100), \tag{6c}$$

where $sign = 1$ if $peaks <= 10$ and $sign = -1$ if $peaks >= 100$ (the initial value of $sign$ is one), and $rand(a, b)$ returns a random value in $[a, b]$. This feature make the MPB more difficult to solve as the number of populations must adapt to the changes regarding the number of peaks. The default settings for the MPB used in the experiments of this paper are given in Table 1. The new feature is disabled by default, unless stated otherwise in this paper.

3.2. Benchmark Algorithms

For experiments in this paper, a set of benchmark algorithms are selected from the literature of multi-population methods for DOPs. They are mQSO [7], SPSO [51], CPSO [64], CPSOR [36], AMSO [38], SAMO [4], DynDE [45], DynPopDE [59], and CDDE_Ar [25]. These algorithms represent several research directions of multi-population methods. For example, SPSO is the origin of the re-grouping model [39, 50, 2, 3, 55, 42]; mQSO is the origin of the competing model [5, 20, 45]; CPSO is the origin of the clustering model [35, 36, 38]. Among these methods, only SAMO, DynPopDE and AMSO are adaptive algorithms. These algorithms were chosen to ensure that they represent

Table 1: Default settings for the MPB problem, 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	Parameter	Value
number of peaks ($peaks$)	10	number of dimensions (D)	5
change frequency (u)	5000	correlation coefficient (λ)	0
height severity	7.0	number of peaks change	no
width severity	1.0	S	[0, 100]
peak shape	cone	H	[30.0, 70.0]
basic function	no	W	[1, 12]
shift length (s)	1.0	I	50.0

all typical characteristics of multi-population methods. In addition to the above algorithms, we also develop two new multi-population algorithms in this paper: One is a multi-population PSO (mPSO) algorithm and the other is a multi-population DE (mDE) algorithm. The details of these two algorithms will be introduced later in Sect. 4.1.1.

The algorithms are carefully selected in each set of experiments to ensure that the results are meaningful and representative. For example, in the experimental study of varying the search radius in Sect. 4.1.3, only those algorithms that have configurable search radius are selected to investigate the usefulness of this feature.

Note that, for all experiments in this paper, for each algorithm we use the best known parameter values as suggested by the authors of the algorithms. The values of parameters of mPSO and mDE were obtained by several preliminary experiments. To investigate the impact of each challenge, we only focus on parameters that can be affected by this challenge. The values of these parameters are also carefully chosen through a thorough sensitivity analysis.

3.3. Performance Evaluation

In order to investigate the effect of the aforementioned issues on the performance of an algorithm in locating and tracking multiple optima, this paper uses three performance measures, which are described below.

3.3.1. Average Score (score)

In order to investigate the capability of an algorithm in responding to changes, i.e., how quickly the algorithm converges to the global optimum after a change occurs, we use the performance measure *score* [37]. The measure is defined as follows:

$$score = \frac{1}{K}\sum_{k=1}^{K}(r_k^{best}/(1+\sum_{p=1}^{P}(1-r_k^p)/P)), \tag{7}$$

where r_k^{best} is the relative value of the best-so-far solution to the global optimum at the end of the k-th environment (i.e., just before the k-th change), $r_k^{best} = f(x_k^{best})/f(x_k^*)$ for maximization problems and $r_k^{best} = f(x_k^*)/f(x_k^{best})$ for minimization problems; r_k^p is the relative value of the best-so-far solution to the global optimum at the p-th sampling point during the k-th environment, $r_k^p = (f(x_k^p) + offset)/(f(x_k^*) + offset)$ for maximization problems and $r_k^p = (f(x_k^*) + offset)/(f(x_k^p) + offset)$ for minimization problems, where $offset$ was set to $fabs(f(x_k^*)) + 1$ and is used to ensure that $(f(x_k^*) + offset)$ is greater than 0; $P = u/s\_f$ is the total number of sampling points for an environment, where u is the change frequency of the MPB problem and s_f is the sampling frequency, which was set to 100.

3.3.2. The Percentage of Peaks Being Tracked (tPercent)

The aim of multi-population approaches is to locate and track multiple peaks simultaneously. Therefore, the percentage of peaks that are tracked by an algorithm is an important measure. A peak is assumed to be tracked if the distance from any solution to the peak is less than 0.1 for the MPB problem in this paper.

3.3.3. The Tracking Ratio for the Global Optimum (gRatio)

The above measures cannot show exactly how well an algorithm is able to locate and track the global optimum. Therefore, we use an additional measure: the average tracking ratio, which is the percentage of times where the global optima are successfully tracked by an algorithm over all changes.

3.3.4. t-Test Comparison

To evaluate if the difference in performance (average score and average best error) of any two peer algorithms is statistically significant, a two-tailed *t*-test with 58 degrees of freedom at a 0.05 level of significance was conducted. The *t*-test results are shown as a superscript letter of "w", "l", or "t" next to the average scores and average best-error values of each algorithm to indicate whether the performance of an algorithm is significantly better than, significantly worse than, and statistically equivalent to its peer algorithm, respectively. For example, in the first row of Table 2 in Sect. 4.1.2, CPSO has a score of 0.95^{w} against AMSO, meaning that the score of CPSO is 0.95 and it is significantly worse than that of AMSO. All the results reported in this paper are averaged over 30 independent runs.

3.3.5. Other Performance Measurements

There are several other common performance measures for EAs in dynamic environments, such as the offline error, the offline performance, the best-before-change error, and measures for robust optimization, etc. The offline error, which is used to measure the performance in response of changes, averages over the errors of the best solution found since the last change after every function evaluation. The offline performance is used in the situation where the global optimum is unknown, by simply averaging the fitness of the best solution at each evaluation. The best-before-change error, which is used to measure the solution quality, is the average error of the best solution obtained just before a change occurs. Measures for robust optimization are used to measure the robustness of solutions over a time span.

Although there are many other measures as mentioned above, for the purpose of investigating the impact of the aforementioned challenges on multi-population methods, the three performance measures selected in this paper are the most suitable. The score measure is chosen because it has the properties of both the offline error and best-before-change error. In addition, this measure makes it easy to compare the performance of a set of algorithms. From Eq. (7), for an algorithm, the better the solution obtained, the larger the value of r_k^{best} and hence, the larger the value of score. Also, the less time an algorithm spent to relocate the global optimum after a change occurs, the larger the value of score it would achieve. Similarly, the *tPercent* and *gRatio* measures are chosen over other measures because they help evaluating an algorithm's performance in tracking multiple solutions, including the global optimum.

3.4. Outdated Memory Issues

For PSO algorithms, it is necessary to update fitness values of particles' personal best positions once a change occurs. This is because there is no method that is always able to detect changes (see discussions in Sect. 4.2.6). Therefore, there will be errors in performance evaluation due to outdated fitness values of particles' personal best positions. To avoid such errors, for PSO algorithms involved in the experimental study, the fitness values of particles' personal best positions are updated automatically once a change occurs. The same procedure is applied to individuals of DE algorithms as the same issue exists in the case of DE algorithms.

4. The Challenges for Multi-population Methods

This section comprehensively analyzes the challenges for multi-population methods to locate and track multiple peaks and shows the effect through experimental results. Three major challenges and several other considerations for multi-population methods are explained and corresponding suggestions are also given from the algorithm design point of view in this section.

4.1. Major Challenges

As mentioned before, the key issue of applying EAs to dynamic environments is how to maintain population diversity. Diversity loss is generally handled by diversity increasing/regaining mechanisms in most multi-population studies, e.g., re-initializing inactive individuals or introducing extra new active individuals. Therefore, this paper focuses on analyzing the major issues encountered by the diversity increasing/regaining studies in multi-population approaches for DOPs with many peaks.

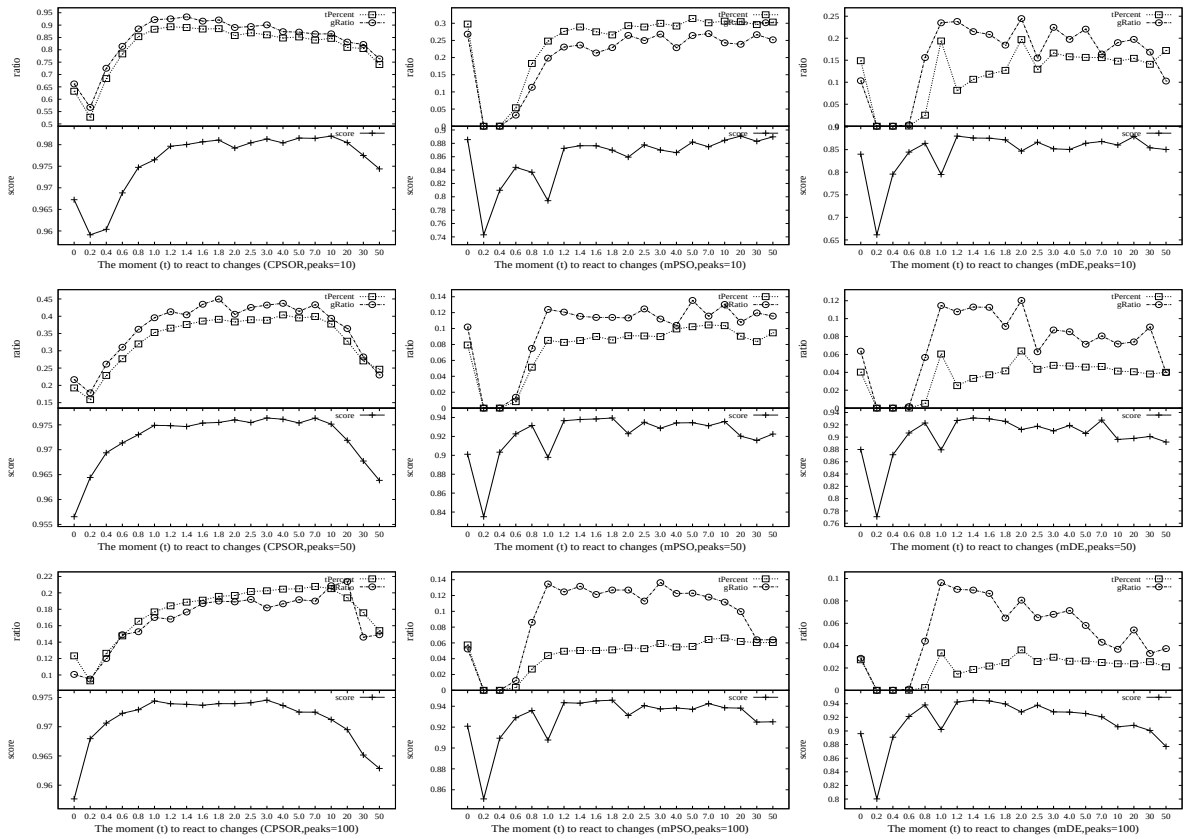

Figure 1: Effect of varying the moments to react to changes on the performance of CPSOR [36], mPSO, and mDE in terms of the average score (*score*), tracking ratio of the global optimum (*gRatio*), and percentage of tracked peaks (*tPercent*) on the MPB problem with different numbers of peaks under the default change frequency of 5,000 *evals*.

4.1.1. Determining the Moment to React to Changes

The first challenge is to determine when to react to changes. Most researchers believe that the moment when a change occurs is when the algorithm should react to changes. According to this assumption, many algorithms, including multi-population based algorithms, have been proposed to react to changes at the moment when a change occurs by increasing/introducing diversity or reusing information learnt from the past [7, 8, 39, 35, 43, 44, 64, 68].

However, we will show below that this choice might not be true in all situations, at least for three algorithms: CPSOR [36], mPSO, and mDE. mPSO and mDE are simple multi-population based PSO and DE algorithms, respectively. In these two algorithms, ten populations are used. The *gbest* model and the $DE/best/2$ mutation strategy [59, 45] are used in mPSO and mDE, respectively. To react to changes, a certain number of new random individuals are introduced in CPSOR and all populations are re-initialized in mPSO and mDE. Note that, the parameter settings of the three algorithms are made based on several preliminary experiments and several experimental studies [7, 59, 45, 64]. For example, ten populations are suggested by many studies [7, 59, 45, 30]. Figures 1 and 2 present the effect of varying the moment to react to changes on the MPB problem with the change frequency of u=5,000 and u=10,000, respectively. The horizontal axis denotes the moment to introduce new random individuals in CPSOR or to re-initialize populations in mPSO and mDE. Taking the value of 1.0 in mPSO as an example, it means that populations are re-initialized every $1.0 * u$ fitness evaluations, i.e., at every moment when a change occurs. Note that, the results for the value of 0 denote that no reaction is performed in the three algorithms throughout the run time. From the results in Fig. 1, we can have the following observations for the involved algorithms on the MPB problem.

Firstly, it is not a good option to react to changes frequently, e.g., more than once during one change interval ($0 < t < 1.0$). For all the three algorithms, the performance greatly drops in terms of all the three measures when

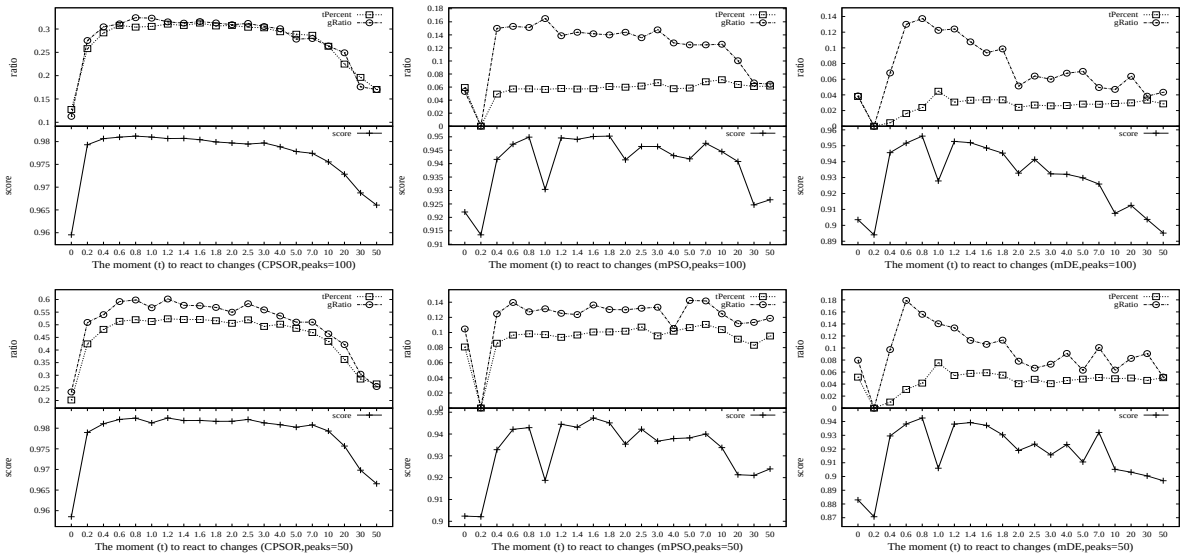

Figure 2: Effect of varying the moments to react to changes on the performance of CPSOR [36], mPSO, and mDE in terms of the average score (*score*), tracking ratio of the global optimum (*gRatio*), and percentage of tracked peaks (*tPercent*) on the MPB problem with different numbers of peaks under the change frequency of 10,000 *evals*.

the moment (t) to react to changes is between 0 and 1.0. The more frequently they response to changes, the worse performance they achieve. For example, mPSO and mDE can hardly track any peak when t decreases below 0.4. Frequently introducing/restarting individuals causes the algorithms to be unable to sufficiently exploit peaks before a change occurs. This observation is consistent with the behavior of traditional algorithms reviewed in [29], which states that continuous focus on diversity slows down the optimization process..

Secondly, it is neither a good option to react to changes occasionally, e.g., $t > 7.0$, especially for the MPB problem with a large number of peaks (e.g., $peaks = 100$). Taking the CPSOR algorithm as an example, the performance also drops when t increases above 7.0, where individuals are introduced every relatively large number of changes. The reason is that the risk of losing peaks that have been located will increase if diversity regaining is not carried out in a relatively long time.

Thirdly, the performance of the three algorithms without any reaction to changes seems not so bad as intuitively expected. In the case ($t = 0$), the results are similar to the results of $t = 50$. When we observe the behavior of three algorithms, we found that environmental changes always cause a small variation to the populations. This small variation enables the populations to re-locate peaks that are close to them, even when they are converging.

Fourthly, the best option to react to changes seems to be the moment corresponding to t values between 1.0 and 5.0 for all the three algorithms under the default change frequency $u = 5000$. The algorithms achieve the best performance in most cases in terms of the three measures by using the values of t in this range.

Finally, for the particular case of $t = 1.0$, where reactions are carried out just at the moment when a change occurs, it is interesting to see that mPSO and mDE have a significant drop in their score values, while CPSOR does not experience such drop in its score. An explanation for the drop in the scores of mPSO and mDE is that the two algorithms react to changes by restarting themselves without using any knowledge of previous environments. The restart happens just at every moment when a change occurs. In the cases of $t \neq 1.0$, the best solution found so far since the last change is recorded before the next change for the performance evaluation. However, the best solution does not make sense in the case of $t = 1.0$ since: 1) the best solution belongs to the previous environment and 2) restarting population means no individual survives for the new environment, i.e., the search restarts from scratch. Therefore, there is a significant performance drop in the score of mPSO and mDE in the cases of $t = 1.0$. Since CPSOR does use information of previous environment (the archived best individuals of converged populations) to accelerate its search, it does not exhibit a clear drop in its performance if the new environment somehow resembles the old environment. The results indicate that reacting to changes at the moment of change occurrence may be not the best choice for a

certain kind of response schemes, e.g,. the restart scheme in mPSO and mDE in this paper.

Compared with Fig. 1, Fig. 2 shows similar observations on these three algorithms except that a smaller t helps the algorithms achieve the best results due to the use of a larger change interval ($u = 10,000$). For example, $t = 0.8$ helps CPSOR and mDE achieve the largest *gRatio* in Fig. 2, while that value is 1.0 for the two algorithms in Fig. 1.

From the results in Figs. 1 and 2, it can be seen that the moment to react to changes plays an important role in the performance of the three algorithms. The problem is how to determine a proper moment. This is a challenging issue as a good choice depends not only on the change frequency but also on the converging status of populations. Our recent study of the AMSO algorithm in [38] suggests that a good moment for the AMSO algorithm to react to changes may be when the populations are no longer able to locate or track any new optimum, rather than when a change occurs. To estimate the moment when no new optimum is found, the formula below is used [38]:

$$(pop(t-\delta) - pop(t))/\delta < 0.002 \tag{8}$$

where $pop(t)$ is the number of populations at time t, and δ is a parameter to determine the time gap between two successive checking points. Similar idea can also be found in algorithms SAMO [4] and DynPopDE [59], where a new free population is created when all populations are converging/stagnating rather than when a change occurs.

It should be noted that this challenge exists only for algorithms with diversity increasing/regaining mechanisms, such as the three algorithms involved above. For algorithms with diversity maintaining mechanisms, such as mQSO [7] and DynDE [45], they do not have such issue (but extra evaluations are needed to maintain diversity at each iteration).

4.1.2. Determining the Appropriate Number of Populations to Deal with Changes

The second challenge in maintaining population diversity is to determine the correct number of populations to deal with changes. This issue lies in two aspects. The first aspect is to determine the number of populations for algorithms using a fixed number of populations. The proper number of populations is mainly determined by the number of peaks in the fitness landscape. Generally speaking, the more peaks in the fitness landscape, the more populations are needed for problems like the MPB. Several experimental studies [7, 45, 64] showed that the optimal number of populations is equal to the number of peaks in the fitness landscape for the MPB problem [8] with a small number of peaks (e.g., less than ten peaks). However, recent evidences in [59, 38] showed that the optimal number of populations is not equal to the number of total peaks for the MPB problem with many peaks (e.g., more than ten peaks). Although locating and tracking each peak by a single population is theoretically right, it is not efficient and hard to achieve in practice in particular for DOPs with a huge number of optima (e.g., the GDBG benchmark [37]) because it is hard to move populations to the right areas and only limited computational resources are available. In addition, the distribution and shape of peaks may also play a role in configuring the number of populations.

The second aspect is to determine the number of populations to be increased/decreased for algorithms using a variable number of populations. This is also a difficult problem. For example, in dynamic environments with an unknown and changing number of optima, the dynamic increase or decrease in the number of populations should be in line with the increase or decrease in the number of peaks. The increase/decrease in the number of peaks, however, is generally unknown to algorithms.

To illustrate the impact of changing number of populations, Fig. 3 presents the results of varying the number of populations on the performance of DynDE [45], mQSO [7], and CDDE_Ar [25] in terms of the three performance measures on four different MPB instances. Note that, both DynDE and mQSO used ten populations in their original papers and the value of k for the k-means method used in CDDE_Ar was also suggested to be ten. Experimental results in Fig. 3 show that the performance of all the three algorithms are sensitive to the total number of populations in terms of the three measures. For example, for problems with the same number of peaks, the performance of mQSO improves as the number of populations increases to a certain level, then it worsens as the number of populations further increases. For problems with different numbers of peaks, the best choice of the number of populations also varies. For example, mQSO achieves the best *gRatio* value using ten populations on the 10-peak MPB problem, while using 16 populations on the 20-peak MPB problem. The results in Fig. 3 clearly shows that the choice of the number of populations does affect the performance of the three algorithms especially in terms of *gRatio* and *tPercent* and the best choice is problem dependant.

To efficiently solve DOPs, the results above suggest that adapting the number of populations is needed. Below we will show an experimental example illustrating the need of choosing the right number of populations. We compare

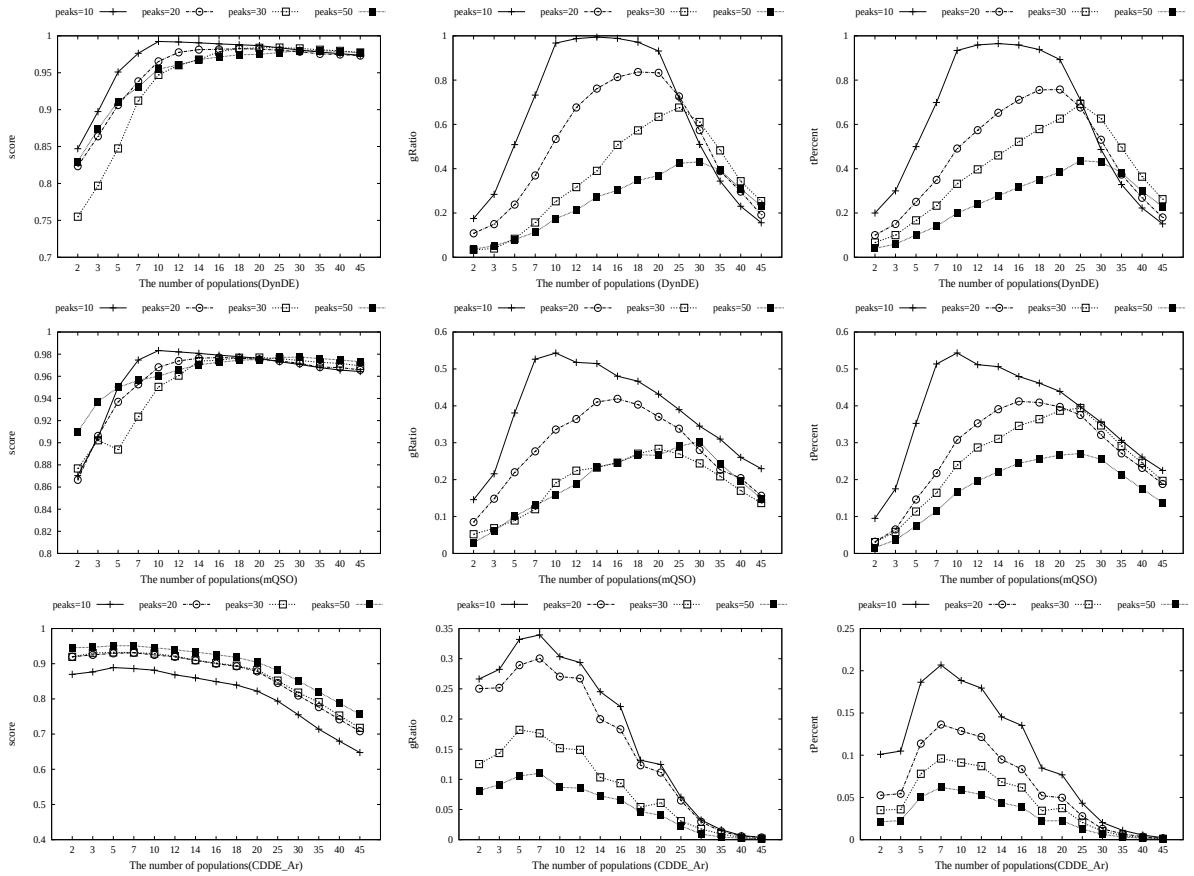

Figure 3: Effect of varying the number of populations on the performance of DynDE [45], mQSO [7], and CDDE_Ar [25] in terms of the average score (left), tracking ratio of the global optimum (middle), and percentage of tracked peaks (right) with different numbers of peaks.

two algorithms, CPSO [64] and AMSO [38]. The only difference between these two algorithms is that AMSO has a feature to adapt the number of populations. This is done by utilising a finding in [38]: the number of populations should be in synchronisation with the number of peaks. Based on this observation, AMSO first checks the variance in the number of populations at two successive checking points (see Eq. (8)), then add or delete more populations accordingly. The larger the variance, the larger the number of populations that will be increased or decreased.

Figure 4 presents the comparison of progress on the number of populations and *tPercent* for CPSO and AMSO on different MPB instances. From the graphs in Fig. 4 with a fixed number of peaks ($peaks = 10$ and $peaks = 30$), AMSO shows a better adaptability: It is able to adaptively choose different numbers of populations for itself in relation to different problems. For example, on the 10-peak MPB instance, the number of populations is about 15, while on the 30-peak MPB instance the number of populations stabilizes at 28. Due to the limitation of CPSO and the difficulties in determining the correct number of populations, CPSO does not show such population adaptation capability. In CPSO, the number of populations drops during each change interval and is simply restored to an initial level after a change occurs. Comparing the results of CPSO between the two MPB instances, we cannot observe the behavior change as AMSO shows. Due to the adaptation capability of AMSO, its performance is much better than that of CPSO regarding *tPercent*. On the 10-peak MPB instance, the percentage of peaks tracked by AMSO is much higher than that of CPSO and it reaches almost 100% after the 60th change. For the 30-peak MPB instance, although *tPercent* of AMSO is smaller than that of CPSO for the first 40 changes, the value gradually improves and finally overtakes the value achieved by CPSO at the 40th change.

For the problem instances with a changing number of peaks, AMSO shows a certain level of adaptability to the

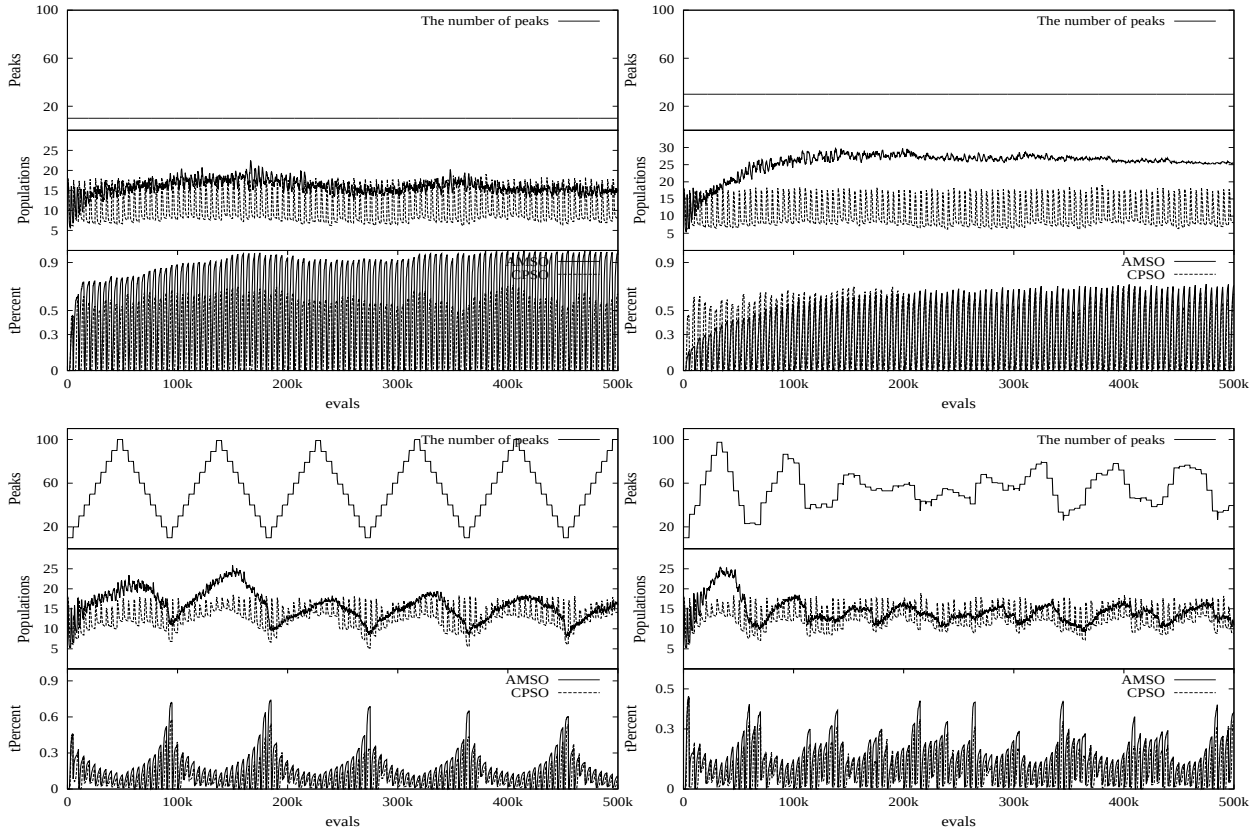

Figure 4: Comparison of the number of populations and the percentage of tracked peaks for AMSO [38] and CPSO [64] on the MPB problem with different instances, where CPSO is AMSO without population adaptation.

changing environments where the number of populations is generally synchronous with the change pattern of the number of peaks. However, we cannot observe such a clear adaptation in CPSO. Again, AMSO shows a better performance than CPSO on these two instances in terms of *tPercent* measure.

To show the advantage of adapting the number of populations, Table 2 presents the comparison of the average score, *gRatio*, and average number of populations (*Pops*) between three pairs of algorithms. The algorithms AMSO, SAMO, and DynPopDE are three adaptive versions of the algorithms CPSO, mQSO, and DynDE, respectively, regarding the number of populations. For each pair of these algorithms, the only difference between the adaptive version and the non-adaptive one is that the earlier has a population adaptation mechanism. The results in Table 2 show that the adaptive versions chose a very different number of populations, compared to their non-adaptive peers. As stated above, the larger the number of peaks, the more number of populations are used in the adaptive algorithms. It can be seen that the results of the three adaptive versions are much better than that of the three non-adaptive algorithms in most cases in terms of *score* except for the pair of DynPopDE [59] and DynDE [45], which were proposed by different authors, in the cases of variable number of peaks. Note that, the tracking ratio of the global optimum achieved by SAMO is worse than that of mQSO. The reason is that SAMO spends a much larger number of evaluations for charged particles to maintain the population diversity at every iteration due to a larger number of populations used.

Although the adaptive versions have much better results than the non-adaptive versions, the number of populations grows as the number of peaks increases (e.g., DynPopDE). As a result, for problems with a huge number of peaks, it is likely that a large number of populations is generated, which in turn requires many evaluations per iteration. This computational cost issue should be addressed in future research.

4.1.3. Search Areas of Populations

For multi-population approaches, ideally each population should only cover the area surrounding one optimum. This way, the populations are able to locate and track multiple optima in different sub-areas simultaneously. Normally,

Table 2: Average score (*score*) ± standard error, tracking ratio for the global optimum (*gRatio*), and average number of populations for three pairs of algorithms on the MPB problem with different numbers of peaks, where AMSO, SAMO, and DynPopDE are the adaptive versions of CPSO, mQSO, and DynDE, respectively, and var1, var2, and var3 are problems with changing number of peaks by Eq. (6a), Eq. (6a), and Eq. (6a), respectively. The *t*-test comparison is performed between each pair of algorithms

peaks	Evaluation	CPSO	AMSO	mQSO	SAMO	DynDE	DynPopDE
10	*score*	0.95^{w}±0.02	**0.99**±0.008	**0.96**±0.01	0.96^{t}±0.01	**0.98**±0.004	0.97^{w}±0.03
	gRatio	0.71^{w}±0.3	**0.96**±0.09	**0.17**±0.3	0.086^{t}±0.2	0.51^{t}±0.5	**0.63**±0.4
	Pops	7.4	17	10	12	10	10
20	*score*	0.95^{w}±0.02	**0.98**±0.01	0.95^{t}±0.02	**0.96**±0.01	0.96^{t}±0.02	**0.97**±0.02
	gRatio	0.61^{w}±0.2	**0.74**±0.2	**0.053**±0.1	0.01^{w}±0.04	0.26^{w}±0.3	**0.47**±0.4
	Pops	9.7	21	10	17	10	17
30	*score*	0.96^{w}±0.02	**0.98**±0.008	0.95^{w}±0.02	**0.96**±0.01	0.96^{w}±0.02	**0.98**±0.02
	gRatio	0.48^{w}±0.3	**0.73**±0.3	**0.097**±0.2	0.013^{w}±0.04	0.16^{w}±0.2	**0.38**±0.4
	Pops	11	25	10	22	10	23
50	*score*	0.96^{w}±0.01	**0.98**±0.01	0.95^{w}±0.01	**0.96**±0.01	0.95^{w}±0.01	**0.97**±0.01
	gRatio	0.3^{t}±0.2	**0.45**±0.3	**0.076**±0.1	0.0093^{w}±0.04	0.11^{w}±0.1	**0.25**±0.3
	Pops	12	28	10	24	10	28
100	*score*	0.96^{w}±0.01	**0.98**±0.01	0.94^{w}±0.02	**0.96**±0.01	0.95^{w}±0.02	**0.97**±0.02
	gRatio	0.2^{t}±0.2	**0.21**±0.2	**0.031**±0.06	0.0027^{w}±0.01	0.042^{t}±0.06	**0.12**±0.2
	Pops	14	30	10	28	10	40
200	*score*	0.97^{t}±0.01	**0.98**±0.01	0.94^{w}±0.03	**0.96**±0.01	0.95^{w}±0.03	**0.98**±0.01
	gRatio	0.13^{t}±0.2	**0.13**±0.2	**0.025**±0.06	0.001^{w}±0.005	0.034^{t}±0.07	**0.064**±0.1
	Pops	15	35	10	33	10	62
var1	*score*	0.95^{w}±0.03	**0.97**±0.02	0.94^{w}±0.03	**0.95**±0.02	**0.95**±0.03	0.95^{t}±0.03
	gRatio	0.41^{t}±0.3	**0.45**±0.2	**0.055**±0.1	0.029^{t}±0.06	**0.17**±0.2	0.099^{t}±0.2
	Pops	11	18	10	16	10	33
var2	*score*	0.94^{w}±0.04	**0.96**±0.03	0.94^{t}±0.03	**0.95**±0.03	**0.95**±0.03	0.94^{t}±0.04
	gRatio	0.33^{t}±0.4	**0.4**±0.4	**0.11**±0.2	0.045^{t}±0.1	**0.22**±0.3	0.094^{t}±0.2
	Pops	11	16	10	15	10	34
var3	*score*	0.94^{w}±0.04	**0.97**±0.03	0.94^{t}±0.04	**0.95**±0.03	**0.95**±0.03	0.93^{t}±0.06
	gRatio	0.37^{t}±0.4	**0.45**±0.4	**0.1**±0.2	0.049^{t}±0.1	**0.25**±0.4	0.09^{w}±0.2
	Pops	11	15	10	15	10	40

all individuals belonging to a population are restricted to the search area covered by that population only. Therefore, identifying a proper search area for each population is very important to locate the optima within that area. However, determining a proper search area for an initial population with a given number of individuals is a very challenging task. The challenges lie in that: 1) the population may cover several optima instead of one, 2) the population may cover no optimum at all, and 3) the size of the search area is very hard to define due to the irregular shape of the basin of attraction (e.g., see the rotated landscapes in the GDBG benchmark [37]).

Due to the above challenges, most existing multi-population algorithms just use pre-defined values, which are based on empirical experience, to determine the search area for populations. For example, the size of each search area for all populations is set to 30 in rSPSO [3] and HmSO [30] and 25 in FMSO [34]. Some other studies assume that some information of the problem to be solved is known. In such cases, problem information can be used to guide the configuration of the search area. For example, assuming that in the MPB problem we know such information as the number of peaks, the number of variables, and the domain range, Blackwell [7] suggested that the exclusion radius of each population is determined by:

$$r_{excl} = 0.5 * S / peaks^{1/D} \tag{9}$$

where S 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 [20, 45] also adopted the same population radius to solve the MPB problem. To avoid being relied on difficult-to-know, problem-dependent information such as the number of peaks, *peaks* in Eq. (9) was replaced by the number of populations in SAMO [4]. The algorithm DynPopDE [59] also adopted this idea.

Although problem information was not needed in [4, 59], two limitations still exist: 1) The size of the search area for each population is not adaptive to changes and 2) all populations use the same size of the search area.

To overcome the limitations of the above ideas, a clustering based idea were proposed in [35]. Thereafter, CPSOR [36] and AMSO [38] also adopted a similar idea to [35]. The idea is that spatially close individuals are clustered into one population. A unique size of the search area will be calculated for that population according to the distribution of individuals. Initial populations are trained for several iterations to allow individuals within one population to move to

Table 3: Average score ± standard error, and the number of cases where *score* values and *tPercent* values are significantly better (*w*) than, significantly worse (*l*) than, and statistically equivalent (*t*) to peer cases with different exclusion radius (r_{excl}), r_{excl} = 31.5 is obtained by Eq. (9)

Algorithm		0	1	10	20	30	31.5	40	50	60	70
mCPSO	score	0.82±0.07	0.85±0.05	0.87±0.06	0.87±0.05	**0.88**±0.05	0.88±0.05	0.87±0.06	0.83±0.06	0.83±0.07	0.83±0.07
	w,t,l	0,4,6	1,8,1	4,6,0	4,6,0	5,5,0	4,6,0	4,6,0	0,5,5	0,5,5	0,5,5
	tPercent	**0.038**±0.05	0.026±0.04	0.026±0.05	0.028±0.05	0.033±0.05	0.034±0.05	0.037±0.05	0.028±0.05	0.026±0.04	0.028±0.05
	w,t,l	0,10,0	0,10,0	0,10,0	0,10,0	0,10,0	0,10,0	0,10,0	0,10,0	0,10,0	0,10,0
mQSO	score	0.87±0.07	0.96±0.02	0.96±0.01	0.96±0.01	**0.96**±0.01	0.96±0.01	0.96±0.01	0.94±0.02	0.93±0.02	0.92±0.04
	w,t,l	0,1,9	4,6,0	4,6,0	4,6,0	4,6,0	4,6,0	4,6,0	2,2,6	1,3,6	1,2,7
	tPercent	0.13±0.08	0.14±0.09	0.15±0.09	0.17±0.1	0.17±0.1	0.17±0.1	**0.18**±0.1	0.1±0.08	0.11±0.07	0.099±0.07
	w,t,l	0,10,0	1,9,0	3,7,0	3,7,0	3,7,0	3,7,0	3,7,0	0,5,5	0,5,5	0,4,6
SPSO	score	0.37±0.1	0.37±0.1	0.5±0.2	0.89±0.04	0.96±0.02	**0.96**±0.02	0.96±0.02	0.94±0.02	0.94±0.02	0.92±0.04
	w,t,l	0,2,8	0,2,8	2,1,7	3,1,6	7,3,0	7,3,0	7,3,0	5,2,3	5,2,3	4,1,5
	tPercent	**0**±0	**0**±0	**0**±0	**0**±0	**0**±0	**0**±0	**0**±0	**0**±0	**0**±0	**0**±0
	w,t,l	0,10,0	0,10,0	0,10,0	0,10,0	0,10,0	0,10,0	0,10,0	0,10,0	0,10,0	0,10,0
DynDE	score	0.89±0.06	0.97±0.01	0.98±0.01	0.98±0.008	0.98±0.006	**0.98**±0.005	0.98±0.006	0.96±0.02	0.95±0.02	0.93±0.03
	w,t,l	0,1,9	4,2,4	4,6,0	5,5,0	5,5,0	5,5,0	5,5,0	3,1,6	2,1,7	1,1,8
	tPercent	0.23±0.08	0.43±0.1	0.46±0.1	0.48±0.1	**0.48**±0.2	0.48±0.1	0.47±0.2	0.21±0.1	0.16±0.1	0.09±0.08
	w,t,l	2,2,6	4,6,0	4,6,0	4,6,0	4,6,0	4,6,0	4,6,0	1,3,6	1,2,7	0,1,9

an area nearby where an optimum is located. Then, the search area for each population is determined after the training process. Therefore, the obtained search area of each population is adaptive to the local fitness landscape.

To show the effect of varying the search radius, we carried out an experimental study on algorithms mCPSO [7], mQSO [7], SPSO [51], and DynDE [45] with different exclusion radius (r_{excl} - the size of the search area for each population), where mCPSO and mQSO have the same structure [7] except that mCPSO uses charged swarms instead of quantum swarms used by mQSO. In mCPSO, mQSO, and DynDE, a population is re-initialized if its best individual is within r_{excl} of another population with a better best individual. SPSO regroups all individuals at every iteration and assigns individuals that are within r_{excl} from its seed to one group. Table 3 presents the results of the four algorithms on the 10-peak MPB problem, where r_{excl}=31.5 was set by Eq. (9). From the results, it can be seen that both *score* and *tPercent* are sensitive to the size of the search area for populations (i.e., the exclusion radius) for all the algorithms. Note that, due to the small threshold (0.1) for checking a tracked peak, *tPercent* achieved by SPSO is zero in all cases. The estimated radius by Eq. (9) does work on the MPB with default settings in Table 1.

Although improvements have been achieved via adapting the search areas to the fitness landscape in [35], far more effort is still needed as a symmetric peak shape, e.g., the cone shape in the MPB problem, is assumed in current research of EAs for UCDOPs. For complex peak shapes, e.g., the rotated peak shape in the GDBG benchmark [37], there is no research yet.

It should be noted that this challenge only exists for multi-population methods that aim to locate and track multiple peaks simultaneously (most existing multi-population methods belong to this category), such as SPSO [51], mQSO [7], CPSO [64], CPSOR [36], DynDE [45], and SOS [9], etc. However, for algorithms like CESO [43] where overlapping search is allowed, there is no such issue.

4.2. Other Considerations

Besides the above major challenges, several other concerns should be also addressed when multi-population methods are considered for addressing DOPs. These concerns are discussed below.

4.2.1. Communication

Many researchers believe that communication among populations (communication here means exchanging individuals among different populations) is helpful for the search since information is able to transmit among populations and, hence, this will accelerate the search and promising solutions may be found as well. However, interestingly, most multi-population algorithms for UCDOPs have no communication, especially for algorithms aiming to locate and track multiple peaks, e.g., mQSO [7], SPSO [51], rSPSO [3], CPSO [64], CPSOR [36], AMSO [38], SAMO [4], DynDE [45], DynPopDE [59], and SOS [9].

Table 4: Average score (*score*) ± standard error and the number of tracked peaks (*tPeaks*) for algorithms CPSOR [36], CPSOR*, mQSO [7], and mQSO* on the MPB problem with different numbers of peaks (*peaks*), where CPSOR* and mQSO* are CPSOR and mQSO with communication, respectively. The *t*-test comparison is performed between the two algroithms of each pair

	Algorithm	1	2	5	7	10	20	30	50	100	200
score	CPSOR	**0.96**±0.02	**0.93**±0.04	**0.96**±0.02	**0.96**±0.02	**0.97**±0.02	**0.97**±0.02	**0.98**±0.01	**0.97**±0.01	**0.97**±0.01	**0.98**±0.009
	CPSORC*	0.85^w±0.1	0.81^w±0.09	0.85^w±0.05	0.81^w±0.1	0.81^w±0.06	0.87^w±0.05	0.89^w±0.05	0.87^w±0.05	0.88^w±0.05	0.9^w±0.04
tPercent	CPSOR	**0.98**±0.06	**0.59**±0.3	**0.81**±0.2	**0.6**±0.2	**0.73**±0.1	**0.48**±0.1	**0.42**±0.09	**0.3**±0.06	**0.16**±0.04	**0.086**±0.02
	CPSORC*	0.041^w±0.2	0.02^w±0.08	0.019^w±0.05	0.01^w±0.03	0.0093^w±0.03	0.0057^w±0.01	0.0043^w±0.01	0.0027^w±0.006	0.0014^w±0.003	6.35e-04^w±0.002
score	mQSO	**0.92**±0.03	**0.91**±0.04	**0.96**±0.01	**0.97**±0.01	**0.96**±0.01	**0.95**±0.02	0.95±0.02	**0.95**±0.01	0.94±0.02	0.94^w±0.03
	mQSOM*	0.91±0.03	0.89^w±0.04	0.94^w±0.01	0.95^w±0.01	0.94^w±0.01	0.95^w±0.01	**0.95**±0.01	0.95±0.01	**0.95**±0.02	**0.96**±0.01
tPercent	mQSO	**0.027**±0.1	**0.16**±0.2	**0.27**±0.2	**0.33**±0.2	**0.18**±0.1	**0.099**±0.05	**0.086**±0.04	**0.06**±0.03	**0.03**±0.01	**0.016**±0.007
	mQSOM*	0±0	0^w±0	4.0e-04^w±0.002	2.86e-04^w±0.002	1.33e-04^w±7.2e-04	3.33e-05^w±1.8e-04	2.22e-05^w±1.2e-04	2.67e-05^w±1.4e-04	3.33e-06^w±1.8e-05	5.00e-06^w±2.7e-05

Experimental results in this section reveal that certain type of communication between populations may not be useful to locate and track multiple peaks for the algorithms CPSOR [36] and mQSO [7]. We do so by adding a special type of communication to the two above algorithms and name the communication-equipped algorithms CPSOR* and mQSO*, respectively. The communication method is a ring-type topology where a random individual of a population migrates to one of its neighbour population with a migration frequency of ten iterations. We compare CPSOR* with CPSOR, and mQSO* with mQSO to see which perform better. Table 4 presents the results regarding the average score and the percentage of peaks tracked by these two pairs of algorithms.

From the results, it can be seen that the number of tracked peaks greatly drops to a very low level when communications is applied to both algorithms. Accordingly, the results of both algorithms with communication (CPSOR* and mQSO*) are significantly worse than those of the algorithms without communications (CPSOR and mQSO) on all instances in terms of both performance measures. The reason for this behaviour can be explained as follows. The motivation of multi-population methods is to divide the search space into different sub-areas. Each population locates and tracks peaks within its own search area. It would be easy to track the global optimum if one of the traced peaks becomes the new global optimum or the global optimum moves to one of the search areas of populations. However, when communications are used between populations, migrants keep moving from one population to another population during the search progress. In such case, when a migrant moves from a high-quality peak to a sub-population currently tracking a lower quality peak, it can potentially dominate that sub-population and consequently force that sub-population to abandon the lower quality peak. Such an abandon will eventually reduce the number of peaks that can be tracked by the algorithm. Therefore, the number of peaks tracked greatly drops when the communication scheme is applied. In environments like the MPB where a low-quality peak can become the global optimum in the future when changes occur, reducing the number of peaks being tracked obviously has a negative impact. It contradicts the aim of using multi-population methods.

It should be noted that although communication may not help multi-population methods like CPSOR and mQSO where populations aim to search in different sub-areas, it is necessary for multi-population methods like CESO [43] where populations transmit information between each other to cooperate to track the changing global optimum.

4.2.2. Avoiding Overlapping Search

In order to efficiently locate and track multiple peaks in different sub-areas, overlapping control is necessary on problems with many peaks. This is because overlapping between two populations searching in the same sub-area firstly wastes valuable computational resources and secondly is not helpful for exploring new optima. Overlapped populations normally will be re-initialized (e.g., in mQSO [7] and DynDE [45]) or removed (e.g., in CPSO [64]). Table 5 presents the comparison between three pairs of algorithms where each pair has one algorithm with and another without overlapping control. From the comparison, it can be seen that the score values of the algorithms without overlapping control are significantly worse than those of the algorithms with overlapping control on most MPB instances with many peaks (e.g., more than 5 peaks), except for the pair of CPSO algorithms, where CPSO$^-$ achieves slightly worse *score* values than CPSO. Like the comparison of the other two pairs of algorithms (mQSO and mQSO$^-$, DynDE and DynDE$^-$), the average percentage of peaks tracked by CPSO with overlapping control is much larger than that of CPSO$^-$ without overlapping control.

However, overlapping control seems to be not good to solve problems with a few peaks. Examples can be seen in Table 5, where the algorithm in each pair with overlapping control performs worse than its peer algorithm without overlapping control on the 1-peak and 2-peak MPB instances in terms of both the *score* and *tPercent* measures.

Table 5: Average score (*score*) and *tPercent* for three pairs of algorithms on the MPB problem with different numbers of peaks (*peaks*), where CPSO$^-$ is the CPSO algorithm without overlapping control, mQSO$^-$ and DynDE$^-$ are mQSO and DynDE without exclusion (i.e., $r_{excl} = 0$ in Eq. (9)), respectively. The *t*-test is performed between the two algorithms in each pair

Algorithm		the number of peaks (*peaks*)									
		1	2	5	7	10	20	30	50	100	200
score	CPSO	**0.95**±0.02	0.94±0.04	**0.95**±0.03	**0.96**±0.02	**0.95**±0.02	0.95±0.02	**0.96**±0.02	**0.96**±0.01	**0.96**±0.01	**0.97**±0.01
	CPSO$^-$	0.95±0.01	**0.95**±0.02	0.95±0.02	0.96±0.01	0.95±0.02	**0.96**±0.01	0.96±0.01	0.96±0.01	0.96±0.01	0.97±0.009
tPercent	CPSO	0.12^w±0.3	0.36^w±0.3	**0.72**±0.2	**0.62**±0.1	**0.61**±0.1	**0.41**±0.05	**0.3**±0.04	**0.2**±0.02	**0.1**±0.01	**0.053**±0.008
	CPSO$^-$	**1**±0.02	**0.79**±0.2	0.58^w±0.2	0.48^w±0.1	0.41^w±0.1	0.26^w±0.07	0.19^w±0.05	0.12^w±0.03	0.066^w±0.02	0.034^w±0.01
score	mQSO	0.92^w±0.03	0.91^w±0.04	**0.96**±0.01	**0.97**±0.01	**0.96**±0.01	**0.95**±0.02	**0.95**±0.02	**0.95**±0.01	**0.94**±0.02	**0.94**±0.03
	mQSO$^-$	**0.95**±0.01	**0.95**±0.04	0.81^w±0.05	0.87^w±0.04	0.89^w±0.08	0.89^w±0.04	0.88^w±0.06	0.92^w±0.05	0.92^w±0.04	0.92^w±0.05
tPercent	mQSO	0.027±0.1	0.16^w±0.2	**0.27**±0.2	**0.33**±0.2	**0.18**±0.1	**0.099**±0.05	**0.086**±0.04	**0.06**±0.03	**0.03**±0.01	**0.016**±0.007
	mQSO$^-$	**0.14**±0.3	**0.31**±0.3	0.036^w±0.08	0.2^w±0.09	0.13^w±0.07	0.044^w±0.03	0.057^w±0.03	0.046^w±0.02	0.026±0.01	0.012^w±0.006
score	DynDE	0.96^w±0.01	0.91^w±0.04	**0.97**±0.007	**0.98**±0.005	**0.98**±0.004	**0.96**±0.02	**0.96**±0.02	**0.95**±0.01	**0.95**±0.02	**0.95**±0.03
	DynDE$^-$	**0.98**±0.003	**0.96**±0.04	0.81^w±0.04	0.84^w±0.002	0.89^w±0.06	0.87^w±0.04	0.87^w±0.03	0.9^w±0.06	0.92^w±0.04	0.91^w±0.05
tPercent	DynDE	0.16^w±0.4	0.12^w±0.2	**0.33**±0.2	**0.4**±0.2	**0.48**±0.2	**0.25**±0.08	**0.17**±0.05	**0.1**±0.03	**0.052**±0.02	**0.026**±0.008
	DynDE$^-$	**0.99**±0.009	**0.71**±0.2	0.21^w±0.05	0.27^w±0.04	0.23^w±0.08	0.14^w±0.04	0.1^w±0.02	0.06^w±0.02	0.039^w±0.01	0.018^w±0.006

Table 6: Average score ± standard error, and the number of cases where *score* value is significantly better (*w*) than, significantly worse (*l*) than, and statistically equivalent (*t*) to its peer cases for CPSO and AMSO on the MPB problem with 10 peaks

Algorithm		size of individual population (*subS ize*)												
		3	5	7	9	11	13	15	17	20	25	30	40	50
CPSO	score	0.92±0.03	0.95±0.03	**0.95**±0.02	0.95±0.03	0.95±0.02	0.95±0.02	0.94±0.02	0.94±0.02	0.94±0.02	0.94±0.03	0.93±0.03	0.93±0.04	0.91±0.05
	w,t,l	0,5,8	3,10,0	4,9,0	3,10,0	3,10,0	3,10,0	2,11,0	2,11,0	2,11,0	1,12,0	0,12,1	0,8,5	0,4,9
	tPercent	0.48±0.2	**0.68**±0.1	0.61±0.1	0.55±0.1	0.5±0.09	0.47±0.09	0.43±0.09	0.4±0.08	0.37±0.08	0.31±0.08	0.28±0.08	0.23±0.08	0.19±0.08
	w,t,l	6,5,2	12,1,0	11,1,1	8,3,2	7,4,2	6,4,3	5,4,4	4,3,6	4,2,7	2,2,9	2,2,9	0,2,11	0,2,11
AMSO	score	0.96±0.02	0.98±0.01	**0.99**±0.008	0.99±0.009	0.99±0.009	0.99±0.01	0.98±0.02	0.98±0.02	0.98±0.03	0.96±0.04	0.95±0.05	0.92±0.07	0.9±0.09
	w,t,l	2,3,8	5,8,0	6,7,0	6,7,0	6,7,0	5,8,0	5,8,0	5,8,0	4,6,3	2,4,7	2,3,8	0,2,11	0,2,11
	tPercent	0.3±0.1	0.89±0.09	**0.92**±0.06	0.89±0.07	0.88±0.07	0.82±0.1	0.78±0.1	0.72±0.1	0.67±0.1	0.57±0.1	0.5±0.1	0.38±0.09	0.31±0.1
	w,t,l	0,2,11	9,4,0	10,3,0	9,4,0	9,3,1	7,2,4	7,2,4	5,2,6	5,2,6	4,1,8	3,1,9	2,1,10	0,2,11

Table 7: Average score ± standard error, and the number of cases where the *score* value is significantly better (*w*) than, significantly worse (*l*) than, and statistically equivalent (*t*) to its peer cases for CPSO and AMSO on the MPB problem with 50 peaks

Algorithm		size of individual population (*subS ize*)												
		3	5	7	9	11	13	15	17	20	25	30	40	50
CPSO	score	0.95±0.02	0.96±0.01	**0.96**±0.01	0.96±0.01	0.96±0.02	0.96±0.02	0.96±0.02	0.95±0.02	0.95±0.02	0.95±0.02	0.95±0.03	0.94±0.03	0.94±0.04
	w,t,l	1,11,1	4,9,0	5,8,0	3,10,0	2,11,0	2,11,0	2,11,0	1,12,0	1,12,0	0,11,2	0,10,3	0,7,6	0,4,9
	tPercent	0.14±0.05	**0.22**±0.03	0.2±0.02	0.17±0.02	0.16±0.02	0.14±0.02	0.13±0.02	0.12±0.02	0.11±0.02	0.091±0.02	0.078±0.02	0.06±0.02	0.048±0.02
	w,t,l	6,4,3	12,1,0	11,1,1	10,1,2	8,2,3	7,2,4	6,2,5	5,1,7	4,1,8	3,1,9	2,1,10	1,1,11	0,1,12
AMSO	score	0.96±0.02	0.98±0.01	**0.98**±0.01	0.98±0.01	0.97±0.01	0.97±0.02	0.96±0.02	0.96±0.03	0.95±0.03	0.94±0.03	0.93±0.04	0.91±0.06	0.88±0.08
	w,t,l	4,4,5	9,4,0	9,4,0	9,4,0	8,5,0	7,3,3	5,4,4	4,4,5	3,4,6	2,3,8	1,3,9	0,3,10	0,2,11
	tPercent	0.047±0.03	0.37±0.06	**0.4**±0.05	0.37±0.05	0.31±0.04	0.24±0.04	0.19±0.04	0.17±0.04	0.14±0.03	0.12±0.02	0.096±0.02	0.074±0.02	0.062±0.02
	w,t,l	0,1,12	10,3,0	11,2,0	10,2,1	9,1,3	8,1,4	6,2,5	6,2,5	5,1,7	4,1,8	3,1,9	2,1,10	1,1,11

4.2.3. The Size of Individual Population

In order to investigate how the population size impacts the performance of a multi-population based EA, a simple experiment was carried out with the algorithms CPSO and AMSO on the MPB problem with 10 and 50 peaks. Tables 6 and 7 present the results reagrding *score* and *tPercent* for both algorithms with different numbers of individual population size (*subS ize*) on the MPB with 10 and 50 peaks, respectively. From the results in each table, it can be seen that varying the individual population size has a significant impact on the performance of the two algorithms, especially regarding the performance *tPercent* (see the *t*-test results in the tables). However, comparing the results in the two tables, it can be seen that the population size that helps the two algorithms achieve the best results does not change. For example, $subSize = 7$ helps AMSO achieve the best performance on both MPB instances.

Table 8: Average score (*score*) ± standard error for algorithms CPSOR, CPSOR', and CDER on the MPB problem with different numbers of peaks (*peaks*), where the CPSOR′ and CDER algorithms use the same algorithm framework as used in the CPSOR algorithm except that the local search operator is replaced by PSO using the *lbest* model [32] and the simple DE algorithm with DE/best/2/bin suggested by [59, 45]

Algorithm		The number of peaks								
		1	2	5	7	10	20	30	50	100	200
score	CPSOR	**0.96**±0.02	0.93±0.04	**0.96**±0.02	0.96±0.02	**0.97**±0.02	**0.97**±0.02	**0.98**±0.01	**0.97**±0.01	**0.97**±0.01	**0.98**±0.009
	CPSOR′	0.96±0.03	**0.95**±0.04	0.96±0.03	**0.96**±0.03	0.96±0.02	0.96±0.02	0.96^{w}±0.02	0.96^{w}±0.02	0.96^{w}±0.01	0.97^{w}±0.01
	CDER	0.31^{w}±0.2	0.56^{w}±0.3	0.58^{w}±0.2	0.49^{w}±0.2	0.54^{w}±0.1	0.46^{w}±0.2	0.42^{w}±0.2	0.45^{w}±0.2	0.53^{w}±0.2	0.5^{w}±0.2
tPercent	CPSOR	**0.98**±0.06	**0.59**±0.3	**0.81**±0.2	**0.6**±0.2	**0.73**±0.1	**0.48**±0.1	**0.42**±0.09	**0.3**±0.06	**0.16**±0.04	**0.086**±0.02
	CPSOR′	0.8^{w}±0.4	0.59±0.4	0.52^{w}±0.2	0.42^{w}±0.2	0.41^{w}±0.2	0.25^{w}±0.1	0.19^{w}±0.07	0.11^{w}±0.04	0.044^{w}±0.02	0.017^{w}±0.009
	CDER	0.012^{w}±0.02	0.0088^{w}±0.02	0.0055^{w}±0.008	0.0025^{w}±0.007	9.67e-04^{w}±0.001	9.18e-04^{w}±0.003	4.44e-04^{w}±8.3e-04	2.00e-04^{w}±2.7e-04	2.40e-04^{w}±7.7e-04	5.50e-05^{w}±1.3e-04

4.2.4. Local Search Operator

Since each population in multi-population approaches usually focuses on one peak only, it might be helpful to hybridise them with a local search operator so that the population can quickly converge to the peak. This helps relocating the moving optima quickly using just a relatively small number of evaluations.

The question is how to choose a suitable local search operator for a particular algorithm. Table 8 presents the comparison of the CPSOR algorithm with three different local search operators with different levels of bias toward exploitation. These are the PSO with the *gbest* model (CPSOR - bias toward exploitation), the PSO with the *lbest* model [32] (CPSOR′ - bias toward exploration), and the DE with the mutation scheme of DE/best/2/bin [59, 45] (CDER - bias toward exploration).

The comparison in Table 8 shows that CPSOR with the *gbest* model, which is the one that biases toward exploitation, achieves the best results in most cases. So, the experimental results in this subsection confirms that, for this class of PSO to solve problems similar to the MPB problem, we should choose a local search operator that focuses more on exploitation rather than exploration.

4.2.5. The Way to Create Populations

How to create populations is also one inevitable question when multi-population methods are applied. As mentioned above, methods for creating multiple populations can be roughly classified into three approaches. The first approach simply uses a certain number of randomly generated populations across the whole search space (e.g., ESCA [44], CESO [43], and mQSO [7]). The second approach starts with a main population and maintains it to generate sub-populations by splitting off from the main population (e.g., SOS [9], FMSO [34], and HmSO [30]) if some predefined criteria are satisfied (e.g., the best individual in the main population does not improve for a certain number of iterations). The third approach divides a large randomly generated population into a set of sub-populations to make them cover different sub-areas in the search space (e.g., the *k*-means PSO [31], SPSO [39], and CPSO [35, 64]).

It is difficult to give an answer to the question of which way to create populations is the best one from an experimental point of view. All the three approaches have their own advantages and disadvantages, as explained below:

- Random initialization approaches.
  - Advantages: It is simple and easy to implement.
  - Disadvantages: Populations have over-lapping search areas and it is difficult to define the search area of each population.
- Clustering-based approaches.
  - Advantages: Populations have no over-lapping search areas and defining the search area of different populations becomes possible (an appropriate search area for each population is still very hard to determine).
  - Disadvantages: It is difficult to develop an effective clustering approach. For example, the population size or the population search radius must be given before the clustering operation in [51, 64]. Moreover, these parameters are problem dependant.
- Splitting-off approaches.

Table 9: Average score (*score*) and percentages of peaks tracked for eight algorithms on the 10-peak MPB with and without noise

		CPSOR	CPSO	AMSO	SPSO	mCPSO	mQSO	SAMO	DynPopDE
score	Origin	**0.97**±0.02	**0.95**±0.02	**0.99**±0.009	**0.95**±0.02	**0.87**±0.05	**0.96**±0.01	**0.96**±0.01	**0.97**±0.03
	Noise	0.96±0.02	0.48^w±0.09	0.94^w±0.04	0.013^w±0.03	0.33^w±0.1	0.33^w±0.1	0.15^w±0.1	0.77^w±0.1
tPercent	Origin	**0.73**±0.1	**0.61**±0.1	**0.86**±0.08	0±0	**0.032**±0.05	**0.18**±0.1	**0.084**±0.08	**0.51**±0.2
	Noise	0.67±0.2	0^w±0	0.3^w±0.1	0±0	0^w±0	0^w±0	0^w±0	0.1^w±2.2e-008

- Advantages: Populations have no over-lapping search areas.
- Disadvantages: It is hard to design an effective splitting rule and special rules need to be designed to prvent the main population from being empty.

From the comparison, generating populations without overlapping seems to be the future trend. This also reflects the divide-and-conquer idea of multi-population methods. This way, individuals that are close to each other will be likely assigned to one group and individuals that are far away from each other will be assigned to different groups. Hence, populations will distribute in different sub-areas without overlapping, and they can locate and track several optima in parallel without any guidance for them to move to different sub-areas.

However, techniques are needed to handle the difficulties in the usage of such methods to create multiple populations, especially learning techniques, which are able to discover the characteristics of the fitness landscape (e.g., the number of peaks, the shape of peaks, and the basion of attraction), are needed. Fortunately, we have seen the start of such work. For example, SAMO [4], DynPopDE [59], and AMSO [38] have been proposed to try to adjust their behaviour in the number of populations.

4.2.6. Change Detection

Change detection is an important issue for EAs in dynamic environments and many studies so far are based on change detection or prediction techniques. Currently, change detection is realized by directly monitoring the fitness change using re-evaluating methods, or by indirectly checking the population average fitness value or other algorithm behaviors [49]. However, there is no change detection or prediction method that is able to guarantee a successful detection or prediction when changes occur in a certain situation, e.g., the dynamic environments with noise. Table 9 shows the comparison of eight algorithms on the 10-peak MPB problem with and without noise. Noise is added to a solution when it is to be evaluated as follows:

$$\vec{x} = \vec{x} + 0.01 \cdot \vec{\sigma} \tag{10}$$

where $\vec{\sigma}$ is a vector of normal distributed random numbers with mean 0 and variation 1.

From the comparison, it can be seen that noise does have a significant effect on the performance of all the involved algorithms, especially on the algorithms which highly depend on change detection methods (e.g., CPSO, SPSO, mCPSO, and mQSO). In such algorithms, noise is misinterpreted as changes. Therefore, diversity increasing operations are being triggered continuously. As a result, the performance significantly drops due to continuously focusing on diversity [29]. Such algorithms hardly track any peak in noisy environments. So, here raises a question: whether we should really need change detection methods as they do not work in such situations.

To answer the above question, we should re-consider the motivation behind change detection. In order to achieve a good performance for an algorithm, many people believe that the moment when a change occurs is a very important time point to react to changes in order to trigger different mechanisms, e.g., diversity increasing, random immigrant, memory, adaptive schemes, and so on. However, experimental results in Sect. 4.1.1 show that for the tested algorithms, it might be not necessary to increase population diversity at the time point when changes occur. What is worse, current change detection techniques may not guarantee a successful detection in all cases. Therefore, it may be a good idea to focus more on identifying new methods that do not heavily rely on change detection in future research.

4.3. Difficulties in Evaluating Algorithm Behavior

There are many performance evaluation methods for EAs in dynamic environments [49]. Many of them are optimization based approaches, such as the score measure used in the paper and several other measures mentioned in

Sect. 3.3. However, for multi-population methods, researchers may be interested in behavior-based measures, such as measures for the moment to increase diversity, measures for the number of populations, and measures for the search areas. It would be helpful for researchers to design algorithms if we can measure algorithms in such behaviors.

However, such behaviors are difficult to measure. The measure for determining the moment to increase diversity is related to algorithm diversity behavior. Although there are several diversity based measures [49], it is still hard to know what level of diversity is optimal. For measures of the number of populations, although we know the fact that a good choice of the number of populations depends on the number of peaks, we do not know their relationship even though we know the number of peaks. For measures of the search areas, it is even harder than the two measures aforementioned.

The behavior-based measures used in this paper (e.g., *gRatio* and *tPercent*) are able to indicate the capability of an algorithm for tracking multiple peaks. However, how to evaluate the quality of peaks that has been tracked by an algorithms is still an open and important question. Developing new behavior-based measures to answer this open question would be very helpful for researchers to develop new algorithms. This direction, however, has been largely overlooked by the community.

5. Conclusions

The multi-population approach, which aims at locating and tracking multiple peaks, is one of the most widely used approaches to solve UCDOPs. However, there has been no in-depth analysis on the possible challenges that this approach may encounter. In order to present a deeper insight into how to design efficient multi-population based algorithms for UCDOPs, this paper comprehensively analyzes and summarizes several challenging issues, which should be considered when designing such algorithms. They are when to react to changes, how many populations are needed, and how to determine the search area of each population. Besides the major challenges, this paper also discusses several other considerations, which are communications between populations, overlapping search, the size of each individual population, the choice of local search operator, the way to create populations, and change detection issues, respectively. The difficulties in evaluating multi-population methods using behavior-based measures are also discussed in this paper.

The challenging issues discussed in this paper suggest that future multi-population based EAs, which aim at locating and tracking multiple peaks, should be able to:

1. adaptively figure out the proper moment to react to changes;
2. adaptively adjust the number of populations to adapt to changes;
3. adaptively determine the search area for each population;
4. adaptively cluster populations;
5. maintain population diversity without change detection;
6. handle overlapping search among populations.

In this paper, in-depth analyses and experimental findings have also been provided to help to achieve some of the objectives listed above. These analyses and findings are summarized as follows:

1. For certain algorithms, e.g., AMSO, SAMO, and DynPopDE, one of the appropriate moment to start to react to changes could be the time point where populations enter into converging status.
2. For algorithms with restarting scheme, e.g., mPSO and mDE, restarting all populations when changes occur may slow down the search process.
3. The number of populations to be increased or decreased may be related to the historical changes of the number of survived populations.
4. Memory schemes are helpful for accelerating the search when changes occur.
5. Clustering based approaches are helpful to guide populations searching in different areas.
6. Clustering based approaches are helpful to determine the search area of each population.
7. Communication between populations might not always be helpful for certain algorithms like CPSO.
8. For some PSO algorithms, e.g., CPSOR, the *gbest* local search operator may be a better choice.

9. In certain cases, overlapping control is essential for solving problems with many peaks. However, it might not be useful for solving problems with only a few peaks.
10. For certain swarm-based algorithms like CPSO and AMSO, the size of a single population should be small.

In summary, a fully adaptive and effective algorithm should be able to learn useful information about the problem from historical data and to use the learned knowledge to guide the future search, and finally to adapt populations to dynamic environments without artificial intervention.

Optimization based performance measures are important and they have been widely studied and used in the literature of dynamic optimization. However, behavior-based measures for multi-population methods have not been widely studied, especially the measures for the number of populations and the search area. Modeling benchmark problems from real-world dataset is also a challenging task. There are obvious gaps between the common academic problem benchmarks and real-world problems in this research area. More detailed discussions on this issue can be seen in [49]. However, these issues should be addressed in the future.

This paper proposes several suggestions on the design of the multi-population based algorithms, which aim to track multiple optima. However, there is no discussion on the difficulties in designing multi-population methods, which are not motivated by the divide-and-conquer idea, such as, CESO and ESCA. Future works on this topic should be addressed. All the studies in this paper only focus on problems in the continuous space. Relevant studies on combinatorial problems should be addressed.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant 61203306, the Engineering and Physical Sciences Research Council (EPSRC) of U.K. under Grant EP/K001310/1, an EU-funded project named “Intelligent Transportation for Dynamic Environment (InTraDE)”, and a Seed-corn funding grant by the Chartered Institute of Logistics and Transport.

[1] Aragón, V. S., Esquivel, S. C., Coello Coello, C. A., September 2011. A t-cell algorithm for solving dynamic optimization problems. Inf. Sci. 181, 3614–3637.
[2] Bird, S., Li, X., 2006. Adaptively choosing niching parameters in a pso. In: 2006 Genetic and Evol. Comput. Conf., pp. 3–10.
[3] Bird, S., Li, X., 2007. Using regression to improve local convergence. In: 2007 IEEE Congr. on Evol. Comput., pp. 592–599.
[4] Blackwell, T., 2007. Particle swarm optimization in dynamic environments. In: Evolutionary Computation in Dynamic and Uncertain Environments. Studies in Computational Intelligence. Spinger, Ch. 2, pp. 29–49.
[5] Blackwell, T., Bentley, P., 2002. Don’t push me! collision-avoiding swarms. In: 2002 IEEE Congr. on Evol. Comput., Vol. 2. pp. 1691–1696.
[6] Blackwell, T. M., Branke, J., 2004. Multi-swarm optimization in dynamic environments. In: Applications of Evolutionary Computation. Vol. 3005. Springer Berlin Heidelberg, pp. 489–500.
[7] Blackwell, T. M., Branke, J., 2006. Multiswarms, exclusion, and anti-convergence in dynamic environments. IEEE Trans. on Evol. Comput. 10 (4), 459–472.
[8] Branke, J., 1999. Memory enhanced evolutionary algorithms for changing optimization problems. In: 1999 IEEE Congr. on Evol. Comput., Vol. 3. pp. 1875–1882.
[9] Branke, J., Kaußler, T., Schmidth, C., Schmeck, H., 2000. A multi-population approach to dynamic optimization problem. In: 4th International Conference on Adaptive Computing in Design and Manufacturing. pp. 299–308.
[10] Bui, L. T., Abbass, H. A., Branke, J., 2005. Multiobjective optimization for dynamic environments. In: 2005 Congr. on Evol. Comput., Vol. 3. pp. 2349–2356.
[11] Bui, L. T., Michalewicz, Z., Parkinson, E., Abello, M., 2012. Adaptation in dynamic environments: A case study in mission planning. IEEE Trans. on Evol. Comput. 16 (2), 190–209.
[12] Chau, K., 2007. Application of a pso-based neural network in analysis of outcomes of construction claims. Automation in Construction 16 (5), 642–646.
[13] Chen, H., Li, M., Chen, X., dec. 2010. A predator-prey cellular genetic algorithm for dynamic optimization problems. In: 2nd Int. Conf. on Information Engineering and Computer Science (ICIECS), pp. 1–6.
[14] Chen, L., Ding, L., Du, X., march 2011. Genetic algorithm with particle filter for dynamic optimization problems. In: 3rd Int. Conf. on Computer Research and Development (ICCRD), Vol. 1. pp. 452–457.
[15] Chen, M.-Y., 2011. Bankruptcy prediction in firms with statistical and intelligent techniques and a comparison of evolutionary computation approaches. Comput. & Math. with Appl. 62 (12), 4514 – 4524.
[16] Chen, M.-Y., 2013. A hybrid anfis model for business failure prediction utilizing particle swarm optimization and subtractive clustering. Inf. Sci. 220 (0), 180–195.
[17] Chen, M.-Y., Chen, D.-R., Fan, M.-H., Huang, T.-Y., 2013. International transmission of stock market movements: an adaptive neuro-fuzzy inference system for analysis of taiex forecasting. Neural Comput. and Appl., 23 (1), 369–378.
[18] Cobb, H. G., Grefenstette, J. J., 1993. Genetic algorithms for tracking changing environments. In: 5th Int. Conf. on Genetic Algorithms, pp. 523–530.

[19] Daneshyari, M., Yen, G., june 2011. Dynamic optimization using cultural based pso. In: 2011 IEEE Congress on Evolutionary Computation. pp. 509–516.
[20] del Amo, I., Pelta, D., Gonzáez, lez, J., july 2010. Using heuristic rules to enhance a multiswarm pso for dynamic environments. In: 2010 IEEE Congr. on Evol. Conput., pp. 1–8.
[21] Du, W., Li, B., 2008. Multi-strategy ensemble particle swarm optimization for dynamic optimization. Inf. Sci. 178 (15), 3096–3109.
[22] du Plessis, M., Engelbrecht, A., april 2011. Self-adaptive competitive differential evolution for dynamic environments. In: 2011 IEEE Symp. on Differential Evolution (SDE), pp. 1–8.
[23] Fernandez-Marquez, J., Arcos, J., july 2010. Adapting particle swarm optimization in dynamic and noisy environments. In: 2010 IEEE Congr. on Evol. Comput., pp. 1–8.
[24] Grefenstette, J. J., 1992. Genetic algorithms for changing environments. In: 2nd Int. Conf. on Parallel Problem Solving From Nature. pp. 137–144.
[25] Halder, U., Das, S., Maity, D., 2013. A cluster-based differential evolution algorithm with external archive for optimization in dynamic environments. IEEE Trans. on Cybernetics 43 (3), 881–897.
[26] Hashemi, A., Meybodi, M., oct. 2009. A multi-role cellular pso for dynamic environments. In: 14th Int. CSI Computer Conf. (CSICC 2009), pp. 412–417.
[27] Huang, Z.-K., Chau, K.-W., 2008. A new image thresholding method based on gaussian mixture model. Applied Math. and Comput. 205 (2), 899–907.
[28] Jiang, Y., Huang, W., Chen, L., jan. 2009. Applying multi-swarm accelerating particle swarm optimization to dynamic continuous functions. In: 2nd Int. Workshop on Knowledge Discovery and Data Mining (WKDD 2009), pp. 710–713.
[29] Jin, Y., Branke, J., 2005. Evolutionary optimization in uncertain environments: a survey. IEEE Trans. on Evol. Comput. 9 (3), 303–317.
[30] Kamosi, M., Hashemi, A. B., Meybodi, M. R., 2010. A hibernating multi-swarm optimization algorithm for dynamic environments. In: World Congress on Nature and Biologically Inspired Computing, NaBIC2010. pp. 363–369.
[31] Kennedy, J., 2000. Stereotyping: Improving particle swarm performance with cluster analysis. In: 2000 IEEE Congr. on Evol. Comput., pp. 1507–1512.
[32] Kennedy, J., Mendes, R., 2002. Population structure and particle swarm performance. In: 2002 IEEE Congr. on Evol. Comput., pp. 1671–1676.
[33] Khouadjia, M., Sarasola, B., Alba, E., Jourdan, L., Talbi, E., may 2011. Multi-environmental cooperative parallel metaheuristics for solving dynamic optimization problems. In: Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on. pp. 395–403.
[34] Li, C., Yang, S., 2008. Fast multi-swarm optimization for dynamic optimization problems. In: 4th Int. Conf. on Natural Comput. Vol. 7. pp. 624–628.
[35] Li, C., Yang, S., 2009. A clustering particle swarm optimizer for dynamic optimization. In: 2009 IEEE Congr. on Evol. Comput., pp. 439–446.
[36] Li, C., Yang, S., 2012. A general framework of multipopulation methods with clustering in undetectable dynamic environments. IEEE Trans. Evol. Comput. 16 (4), 556–577.
[37] Li, C., Yang, S., Pelta, D., 2011. Benchmark generator for cec'2012 competition on evolutionary computation for dynamic optimization problems. Tech. rep., the School of Computer Science, China University of Geosciences, Wuhan, China.
[38] Li, C., Yang, S., Yang, M., 2013. An adaptive multi-swarm optimizer for dynamic optimization problems, Evol. Comput., in press, 2014.
[39] Li, X., 2004. Adaptively choosing neighborhood bests using species in a particle swarm optimizer for multimodal function optimization. In: 2004 Genetic and Evol. Comput. Conf., pp. 105–116.
[40] Liang, Y., nov. 2009. An new efficient evolutionary approach for dynamic optimization problems. In: 2009 IEEE Int. Conf. on Intel. Comput. and Intel. Syst. Vol. 1. pp. 61–65.
[41] Liu, L., Ranjithan, S. R., 2010. An adaptive optimization technique for dynamic environments. Engineering Appl. of Artif. Intell. 23 (5), 772–779.
[42] Liu, L., Yang, S., Wang, D., 2010. Particle swarm optimization with composite particles in dynamic environments. IEEE Trans. on Systems, Man and Cybern. Pat B: Cybern. 40 (6), 1634–1648.
[43] Lung, R. I., Dumitrescu, D., 2007. A collaborative model for tracking optima in dynamic environments. In: 2007 IEEE Congr. on Evol. Comput., pp. 564–567.
[44] Lung, R. I., Dumitrescu, D., 2010. Evolutionary swarm cooperative optimization in dynamic environments. Natural Computing 9 (1), 83–94.
[45] Mendes, R., Mohais, A. S., 2005. Dynde: a differential evolution for dynamic optimization problems. In: 2005 IEEE Congr. on Evol. Comput., pp. 2808–2815.
[46] Morrison, R. W., De Jong, K. A., 2000. Triggered hyper mutation revisited. In: 2000 IEEE Congr. on Evol. Comput., pp. 1025–1032.
[47] Mukherjee, R., Patra, G. R., Kundu, R., Das, S., 2014. Cluster-based differential evolution with crowding archive for niching in dynamic environments. Inf. Sci. 267 (0), 58–82.
[48] Nguyen, T. T., 2011. Continuous dynamic optimisation using evolutionary algorithms. URL `http://etheses.bham.ac.uk/1296/`
[49] Nguyen, T. T., Yang, S., Branke, J., 2012. Evolutionary dynamic optimization: A survey of the state of the art. Swarm and Evol. Comput. 6 (0), 1–24.
[50] Parrott, D., Li, X., 2004. A particle swarm model for tracking multiple peaks in a dynamic environment using speciation. In: 2004 IEEE Congr. on Evol. Comput., pp. 98–103.
[51] Parrott, D., Li, X., 2006. Locating and tracking multiple dynamic optima by a particle swarm model using speciation. IEEE Trans. on Evol. Comput. 10 (4), 440–458.
[52] Rezazadeh, I., Meybodi, M. R., Naebi, A., 2011. Adaptive particle swarm optimization algorithm for dynamic environments. In: Proc. of the 2nd Int. Conf. on Advances in Swarm Intelligence - Vol. I. pp. 120–129.
[53] Rezazadeh, I., Meybodi, M. R., Naebi, A., 2011. Particle swarm optimization algorithm in dynamic environments: Adapting inertia weight and clustering particles. In: 5th UKSim European Symp. on Computer Modeling and Simulation (EMS), pp. 76–82.
[54] Richter, H., 2009. Detecting change in dynamic fitness landscapes. In: 2009 IEEE Congr. on Evol. Comput., pp. 1613–1620.

[55] Schoeman, I. L., Engelbrecht, A. P., 2009. A novel particle swarm niching technique based on extensive vector operations. Natural Computing 9 (3), 683–701.
[56] Shi, X., Qian, F., 2010. Gradient-based immune algorithm for optimization of dynamic environments. In: Proc. 6th Int. Conf. on Natural Computation (ICNC), Vol. 1. pp. 327–330.
[57] Simoes, A., Costa, E., 2008. Evolutionary algorithms for dynamic environments:prediction using linear regression and markov chains. In: Parallel Problem Solving from Nature. pp. 306–315.
[58] Taormina, R., wing Chau, K., Sethi, R., 2012. Artificial neural network simulation of hourly groundwater levels in a coastal aquifer system of the venice lagoon. Engineering Appl. of Artif. Intell. 25 (8), 1670–1676.
[59] du Plessis, M. C., Engelbrecht, A. P., 2012. Differential evolution for dynamic environments with unknown numbers of optima. Journal of Global Optimization, 1–27.
[60] Thomsen, R., 2004. Multimodal optimization using crowding-based differential evolution. In: 2004 IEEE Congr. on Evol. Comput., Vol. 2. pp. 1382–1389.
[61] Wang, H., Wang, N., Wang, D., 2008. Multi-swarm optimization algorithm for dynamic optimization problems using forking. In: Control and Decision Conference (CDC 2008), pp. 2415–2419.
[62] Yang, S., 2006. Associative memory scheme for genetic algorithms in dynamic environments. In: EvoWorkshops 2006: Applications of Evolutionary Computing. Vol. 3907. pp. 788–799.
[63] Yang, S., 2008. Genetic algorithms with memory- and elitism-based immigrants in dynamic environment. Evol. Comput. 16 (3), 385–416.
[64] Yang, S., Li, C., 2010. A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. IEEE Trans. on Evol. Comput. 14 (6), 959–974.
[65] Yang, S., Richter, H., 2009. Hyper-learning for population-based incremental learning in dynamic environments. In: Proc. of the 2009 IEEE Congr. on Evol. Comput., pp. 682–689.
[66] Yang, S., Tinos, R., 2008. Hyper-selection in dynamic environments. In: Proc. of the 2008 IEEE Congr. on Evol. Comput., pp. 3185–3192.
[67] Yang, S., Yao, X., 2005. Experimental study on population-based incremental learning algorithms for dynamic optimization problems. Soft Computing 9 (11), 815–834.
[68] Yang, S., Yao, X., 2008. Population-based incremental learning with associative memory for dynamic environments. IEEE Trans. on Evol. Comput. 12 (5), 542–561.
[69] Yazdani, D., Nasiri, B., Sepas-Moghaddam, A., Meybodi, M., Akbarzadeh-Totonchi, M., 2014. MNAFSA: A novel approach for optimization in dynamic environments with global changes. Swarm and Evol. Comput. (0), in press.
[70] Yazdani, D., Nasiri, B., Sepas-Moghaddam, A., Meybodi, M. R., 2013. A novel multi-swarm algorithm for optimization in dynamic environments based on particle swarm optimization. Applied Soft Computing 13 (4), 2144–2158.
[71] Zechman, E., Ranjithan, S., 2009. Evolutionary computation-based methods for characterizing contaminant sources in a water distribution system. Journal of Water Resources Planning and Management 135 (5), 334–343.
[72] Zhang, J., Chau, K.-W., feb 2009. Multilayer ensemble pruning via novel multi-sub-swarm particle swarm optimization. Journal of Universal Computer Science 15 (4), 840–858.
[73] Zheng, X., Liu, H., 2009. A different topology multi-swarm pso in dynamic environment. In: IEEE Int. Symp. on IT in Medicine Education (ITIME '09), Vol. 1. pp. 790–795.
[74] Zheng, X., Liu, H., 2011. A cooperative dual-swarm pso for dynamic optimization problems. In: 7th Int. Conf. on Natural Computation (ICNC), Vol. 2. pp. 1131–1135.
[75] Zhu, T., Luo, W., Li, Z., 2011. An adaptive strategy for updating the memory in evolutionary algorithms for dynamic optimization. In: 2011 IEEE Symp. on Comput. Intell. in Dynamic and Uncertain Environments (CIDUE), pp. 8–15.