New measures for comparing optimization algorithms on dynamic optimization problems

Javidan Kazemi Kordestani, Alireza Rezvanian & Mohammad Reza Meybodi

Natural Computing
An International Journal

ISSN 1567-7818

Nat Comput
DOI 10.1007/s11047-016-9596-8

Springer

Nat Comput
DOI 10.1007/s11047-016-9596-8

New measures for comparing optimization algorithms on dynamic optimization problems

Javidan Kazemi Kordestani[1] · **Alireza Rezvanian**[2,3] · **Mohammad Reza Meybodi**[2]

Abstract Dynamic optimization problems have emerged as an important field of research during the last two decades, since many real-world optimization problems are changing over time. These problems need fast and accurate algorithms, not only to locate the optimum in a limited amount of time but also track its trajectories as close as possible. Although lots of research efforts have been given in developing dynamic benchmark generator/problems and proposing algorithms to solve these problems, the role of numerical performance measurements have been barely considered in the literature. Several performance criteria have been already proposed to evaluate the performance of algorithms. However, because they only take confined aspects of the algorithms into consideration, they do not provide enough information about the effectiveness of each algorithm. In this paper, at first we review the existing performance measures and then we present a set of two measures as a framework for comparing algorithms in dynamic environments, named *fitness adaptation speed* and *alpha–accuracy*. A comparative study is then conducted among different state-of-the-art algorithms on moving peaks benchmark via proposed metrics, along with several other performance measures, to demonstrate the relative advantages of the introduced measures. We hope that the collected knowledge in this paper opens a door toward a more comprehensive comparison among algorithms for dynamic optimization problems.

Keywords Performance measures · Dynamic optimization problems · Swarm intelligence · Fitness adaptation speed · Alpha-accuracy measure

✉ Alireza Rezvanian
a.rezvanian@aut.ac.ir

Javidan Kazemi Kordestani
Javidan.kazemi@gmail.com

Mohammad Reza Meybodi
mmeybodi@aut.ac.ir

[1] Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran

[2] Soft Computing Laboratory, Computer Engineering and Information Technology, Department, Amirkabir University of Technology (Tehran Polytechnic), 424 Hafez Ave., Tehran, Iran

[3] Department of Computer Engineering and Information Technology, Hamedan University of Technology, Hamedan, Iran

1 Introduction

Almost all aspects of science and engineering include the optimization of a set of complex problems in which the objectives of the optimization, some restrictions or other elements of the problems may vary over time. In these cases, the optimum solution(s) to the problems are subject to change as well. Unlike the optimization in static environments (Hasanzadeh et al. 2013; Kordestani et al. 2014a; Hasanzadeh et al. 2016), the goal of optimization in dynamic problems is no longer just locating the optimal solution(s), but tracking the trajectories of optimum(s) over time with a high level of accuracy (Kordestani et al. 2014b, 2016; Ranginkaman et al. 2014; Sharifi et al. 2015; Alizadeh et al. 2013).

So far, extensive research has been conducted on various aspects of dynamic optimization problems (DOPs). A group of attempts has been focused on modeling different dynamic situations. Different benchmark generators/problems have been devised in the literature to approximate real-world dynamic problems. For example, Branke

Published online: 03 January 2017

(Branke 2002) developed moving peaks benchmark (MPB) which is a real-valued dynamic maximization problem with a D-dimensional landscape consisting of N peaks, where the height, the width and the position of each peak is changed slightly every the time a change occurs in the environment.

Numerous techniques have been developed for solving DOPs, among them *evolutionary optimization techniques* are the most widely used methods to address DOPs. These methods should conquer several challenges in order to obtain promising results in dynamic environments. The most serious challenge when solving DOPs is the problem of *diversity loss* in population-based evolutionary optimization techniques, which arises due to convergence of the population and may prevent individuals from successfully tracking the moving optima (Blackwell 2005).

Different strategies have been presented to deal with diversity loss problem which can be categorized into four groups: (a) increasing diversity (Woldesenbet and Yen 2009; Richter and Dietel 2010), (b) maintaining diversity throughout the run (Yu et al. 2009; Rezvanian et al. 2010), (c) memory-based approaches (Yang 2007; Yang and Yao 2008) and (d) multi-population schemes (Blackwell and Branke 2006; Yang and Li 2010).

Multi-population schemes have attracted a great deal of attention due to their effectiveness. Several multi-population approaches with different perspectives have been proposed in the field to deal with DOPs challenges, which include methods based on atom analogy (Blackwell and Branke 2006; Blackwell et al. 2008), algorithms based on a parent population and some child populations (Li and Yang 2008; Kamosi et al. 2010a, b), space-partitioning techniques (Hashemi and Meybodi 2009a, b; Noroozi et al. 2011; Kianfar and Meybodi 2012; Noroozi et al. 2012; Sharifi et al. 2012), clustering models (Yang and Li 2010; Li et al. 2012; Li and Yang 2012; Nickabadi et al. 2012), collaborative approaches (Lung and Dumitrescu 2007, 2010), adaptive algorithms (Branke 2002; Rezazadeh et al. 2011), and hybrid methods (Nabizadeh et al. 2012a, b; Kordestani et al. 2014b).

Another group of studies have examined the issue of change in DOPs include change detection and response to change (Hu and Eberhart 2002), change prediction (Simões and Costa 2008, 2009), and eliminating the need for change detection (Li et al. 2012).

In addition, several studies have been done on deep reviewing of academic research (Cruz et al. 2010; Nguyen et al. 2012) and analyzing different methods (del Amo et al. 2012) for solving DOPs.

Assessing the quality of different optimization techniques, comparing different methods and ranking various algorithms (or variations of the same algorithm) would be useful for selecting an appropriate algorithm to a particular problem. This requires a set of suitable measures and metrics to evaluate the quality of different algorithms and select a technique that meets the objectives intended.

Several criteria have been already proposed in the literature in order to measure the optimality of the optimization algorithms for DOPs. However, most of them are unable to provide a meaningful interpretation of the superiority of one method over the others. Besides, they are not applicable in many real-world problems where the optimality is not the main concern. Therefore, other aspects should be taken account in many real-world optimizations problems.

In this paper, a set of two measures is presented, one for evaluating the speed and the other for evaluating the accuracy of the optimization algorithms, which is used for comparing the performance of different algorithms in dynamic environments. We will then compare a number of state-of-the-art algorithms on MPB using the proposed measures, as well as other metrics, to show the relative advantages of our proposed set of performance measures.

The rest of this paper is organized as follows: Sect. 2 reviews some of the existing performance measures that are commonly used in the literature. Proposed measures are described in Sect. 3. Section 4 is devoted to experimental studies and discussions. Finally, Sect. 5 concludes the paper and presents several future directions for further research regarding performance measurements in dynamic environments.

2 Related work

In this section, we provide an overview on some of the existing performance measures for dynamic environments based on the survey by Nguyen et al. (2012). The authors divided the existing performance measures into two major categories: (a) optimality-based measures, and (b) behavior-based measures. The following provides a detailed description of each class of performance measures.

2.1 Optimality-based performance measures

Optimality-based performance measures are those metrics that evaluate the ability of different algorithms in terms of finding the optimum or near optimum solutions. Several optimality-based measures will be examined in the sequel.

2.1.1 Best-of-generation

The most commonly used measure in the literature to compare different algorithms is the *Best-of-generation*, in which the best fitness value at each generation is averaged over several runs and plotted against the generation

numbers (Yang 2008; Cheng and Yang 2010). This measure provides a visual view of the performance of different algorithms. The advantage of this measure is that it can provide a tangible picture of how the tested algorithm has performed. However, it is difficult to determine whether the final outcomes of two algorithms are statically different.

2.1.2 Average best-of-generation

Another widely used optimality-based performance measure is the *average Best-of-generation* which is calculated as follows (Yang and Yao 2008; Yang et al. 2010):

$$\bar{F}_{BOG} = \frac{1}{G} \times \sum_{i=1}^{G} \left(\frac{1}{N} \times \sum_{j=1}^{N} F_{BOG_{ij}} \right) \tag{1}$$

where G is the total number of generations in a run, N is the total number of runs and $F_{BOG_{ij}}$ is the fitness value of the best individual of generation i of run j.

2.1.3 Mean fitness error

This measure (Richter and Yang 2009; Wang et al. 2010) is expressed as the mean difference between the best fitness value of the population and the optimum value of the search space averaged across the total number of runs, which is then averaged over all generations. Mean fitness error is formulated as below (Wang et al. 2010):

$$\bar{E}_{BG} = \frac{1}{G} \times \sum_{i=1}^{G} \left(\frac{1}{N} \times \sum_{j=1}^{N} \left(F_{ij}^{*} - F_{BG_{ij}} \right) \right) \tag{2}$$

where G is the number of generations, N is the total number of runs, F_{ij}^{*} and $F_{BG_{ij}}$ are the fitness of global optimum and the best particle in the population of generation i of run j, respectively.

2.1.4 Best-error-before-change

This measure is described as a difference between the value of the current best individual in the population and the optimum value of the environment "just before the moment of change" (Trojanowski and Michalewicz 1999; Li et al. 2008). This measure is calculated as follows:

$$\bar{E}_{\tau} = \frac{1}{C} \times \sum_{i=1}^{C} e_{\tau,i} \tag{3}$$

Where C is the total number of changes and $e_{\tau,i}$ is the fitness error of the best individual right before the ith change. The smaller the obtained value, the better the optimization algorithm. Although this measure is useful in situations where we are interested in the average quality of the best solution right before the change, it does not allow studying the performance of an algorithm along the whole search process. Moreover, the measure requires the information about the global optimum value of the environment at each period, which may be unavailable in real-world applications.

2.1.5 Modified offline error

One of the most commonly used performance criteria in DOPs is modified offline error (Branke 2002) which is calculated as:

$$E_{MO} = \frac{1}{T} \times \sum_{t=1}^{T} e_t^{*} \tag{4}$$

where T is the total number of function evaluations so far and e_t^{*} is the minimum error gained by the optimization algorithm since the last change. Closer value of this measure to zero shows a better optimization process. Despite the fact that offline error is a generally accepted performance measure in dynamic environments, the interpretation of this measure is not always meaningful. For example, comparison of different algorithms in unknown dynamic environments, where the number of peaks are varying during the optimization, is only meaningful when the algorithms are solving a problem with an equal number of peaks.

2.1.6 Distance to the optimum

This measure is calculated as the distance between the best individual of the population and the current optimum in the environment (Ursem 2000; Li and Dam 2003).

$$DTO = Dist\left(best_p^t - opt_p^t \right) \tag{5}$$

where DTO represents distance to the optimum, $Dist$ is distance function, $best_p^t$ and opt_p^t are the position of the best individual of the population and the position of the optimum solution in the search space at time t, respectively. Smaller value of this measure means that the position of the best individual in the population is nearer to the optimum position of the search space. It is worth noticing that this measure is only valid when the peaks in the landscape are symmetric. Besides, the required information about the exact position of the global optimum in the search space may not be available in real-world situations.

2.1.7 Accuracy

Accuracy (Weicker 2002) is a measure in the range of [0,1], with 0 for the least accurate and 1 for the most accurate optimization. It is measured as:

$$accuracy^{t}_{F,EA} = \frac{F\left(best^{t}_{EA}\right) - Min^{t}_{F}}{Max^{t}_{F} - Min^{t}_{F}} \quad (6)$$

where $F\left(best^{t}_{EA}\right)$ is the fitness value of the best individual in the population of evolutionary algorithm at time t, Max^{t}_{F} and Min^{t}_{F} are the maximum and the minimum fitness values of the search space at time t, respectively. However, the computation of this measure depends on the absolute maximum and minimum values in the search space, which are not necessarily available in real-world situations. Moreover, if the whole search space is a plateau in any generation t, then the accuracy measure is not defined.

2.2 Behavior-based performance measures

Behavior-based performance measures evaluate a certain aspect of the algorithms rather than the optimality. In the following, we review some of the existing behavior-based measures in the literature.

2.2.1 Stability

This measure (Weicker 2002), determines the impact of environmental changes on the optimization accuracy.

$$stab^{t}_{F,EA} = Max\left\{0, accuracy^{t-1}_{F,EA} - accuracy^{t}_{F,EA}\right\} \quad (7)$$

The value of stability is in the range of [0, 1], with 0 for the most stable optimization and 1 for the least stable optimization. It should be noted that the most stable algorithm is not necessarily the best performing one in terms of producing high quality solutions.

2.2.2 Reactivity

How fast an algorithm can adapt to changes in the environment, is determined by the *reactivity* measure (Weicker 2002). The ε-reactivity for an algorithm at time t is defined as:

$$react^{(t)}_{F,EA} = min \times \left\{ t' - t | t < t' \leq maxgen, t' \in N, \frac{accuracy^{(t')}_{F,EA}}{accuracy^{(t')}_{F,EA}} \geq 1 - \varepsilon \right\} \times \cup\{maxgen - t\} \quad (8)$$

which *maxgen* refers to the number of generations in evolutionary algorithms, or other types of generation-based algorithms. Algorithms with lower reactivity values are able to react better and faster against changes in the environment. However, the measure is undefined in periods where accuracy is equal to zero.

2.2.3 Fitness degradation

Fitness degradation ($\beta_{degradation}$) measures the degradation of algorithms in dynamic environments using linear regression over the consecutive accuracy values achieved at the end of each period (Nguyen et al. 2012; del Amo et al. 2012). This measure is calculated as:

$$y = \beta_{degradation}\vec{x} + \varepsilon \quad (9)$$

$$x_i = \frac{1}{N}\sum_{j=1}^{N} f\left(period_{best\,ji}\right) \quad (10)$$

where y is an approximation to the overall accuracy, $\vec{x}$ is a vector of size equal to the environmental periods, and $\beta_{degradation}$ is the slope of the regression line. Each x_i is the accuracy of the best solution found in period i averaged over all independent runs N.

The positive value of $\beta_{degradation}$ indicates that the algorithm can produce high quality solutions after changes in the environment. On the contrary, a negative value implies the loss of solution quality, where a smaller value implies a deeper degradation.

2.2.4 Area below curve

A more general performance measure for DOPs is area below curve (ABC) (Alba and Sarasola 2010). This criterion calculates an attribute of a population as the area below curve. If $p_A(x)$ be the function which determines a certain attribute value achieved by algorithm A in each generation, then the normalized area below $p_A(x)$ is calculated as:

$$abc^{A}_{p} = \frac{1}{N}\int_{1}^{N} p_A(x)dx \quad (11)$$

where N is the total number of generations, and $\int_{1}^{N} p_A(x)dx$ is the definite integral of $p_A(x)$ over the interval[1] $[1, N]$.

2.2.5 Dissimilarity factor

Dissimilarity factor (Ayvaz et al. 2012) evaluates the ability of an optimization algorithm to recover after a change. This measure is based on the similarity between optimal values at each generation and the best values of a given algorithm at each generation, in terms of Euclidian distance and cross correlation. Dissimilarity factor of an algorithm is calculated as:

$$DF_{EA} = W_e \times e^{S}_{EA} + W_c \times c^{s}_{EA} \quad (12)$$

In above equation W_e and W_c are the coefficients of Euclidean distance and cross correlation, respectively. e^{s}_{EA}

and c^s_{EA} are scaled versions of Euclidean distance and normalized cross correlation terms, respectively. Dissimilarity value of an algorithm is in the range of [0,1], where 0 is for the best and 1 is for the worst. A summary of the reviewed performance measures for dynamic environments is given in Table 1.

Other types of performance criteria have been also proposed for measuring other aspects of optimization algorithms in dynamic environments such as diversity (Branke 2002; Yang and Yao 2008), and robustness (Handa et al. 2007).

Although all mentioned measures are very profitable in studying and comparing different aspects of optimization algorithms in dynamic environments, they do not provide a general tool for selecting an appropriate algorithm for a specific problem. Besides, they only consider one aspect of the algorithms into account. Therefore, we believe that it is necessary to provide a collection of measures to draw a more comprehensive conclusion on effectiveness of an algorithm in a practical application.

To address above discussion, we provide a set of two measures to evaluate two different aspects of the optimization algorithms in dynamic environments, simultaneously. First measure is fitness adaptation speed (FAS) which is a behavior-based measure and determines the speed of search process. Second measure is alpha-accuracy which is an optimality-based measure and can evaluate the accuracy of the optimization. The collection of these two measures is used as a framework for comparing the efficiency of different methods in DOPs.

3 Proposed measures

One crucial requirement in real-world optimization problems is to establish a balance between the accuracy and speed of the optimization. To meet this requirement, in this section we attempt to introduce two measures to assess the speed and accuracy of algorithms in dynamic environments. A complete description of our proposed measures is given in the rest of this section.

3.1 Fitness adaptation speed

This section presents the FAS measure for evaluating the speed of algorithms in dynamic environments. First, let us provide some definitions which are used in our work.

Definition 1 (*speed*) Consider an object moving along a straight line, the speed of the object is the magnitude or absolute value of its velocity. Suppose the position of the object is a function of time t, $x(t)$, in some time range. The average velocity of the object between any two time step, t_1 and t_2, is calculated as follows:

Table 1 Summary of some of the existing performance measures for dynamic environments

Name	References	Measure
Mean best-of-generation	Yang and Yao (2008), Cheng and Yang (2010), Yang et al. (2010)	The ability of the algorithm to find a better solution at each generation
Mean fitness error	Richter and Yang (2009), Wang et al. (2010)	The mean difference between the best fitness value of the population and the optimum value of the search space averaged across total number of runs
Best-error-before-change	Trojanowski and Michalewicz (1999), Li et al. (2008)	The average quality of the final solution that the algorithm found, just before the change
Modified offline error	Branke (2002)	The average fitness error of the best position found by the algorithm at every point in time
Distance to the optimum	Ursem (2000, Li and Dam (2003)	The ability of the algorithm to find nearer positions in the search space to the optimum
Accuracy	Weicker (2002), Alba et al. (2010), Sarasola et al. (2013)	The goodness of the best solution in the current population with respect to the best and worst known values in the search space
Stability	Weicker (2002), Alba et al. (2010), Sarasola et al. (2013)	The impact of changes in the environment on optimization accuracy
Reactivity	Weicker (2002), Alba et al. (2010), Sarasola et al. (2013)	The ability of the algorithm to react quickly to changes
Fitness degradation	Alba et al. (2010), Sarasola et al. (2013)	The degradation of algorithms in dynamic environments
Area below curve	Alba and Sarasola (2010)	An attribute of a population as the area below curve
Dissimilarity factor	Ayvaz et al. (2012)	The ability of an optimization algorithm to recover after a change

$$\bar{V} = \frac{\Delta x}{\Delta t} = \frac{x(t_2) - x(t_1)}{t_2 - t_1} \tag{13}$$

where $\bar{V}$ is the speed of the object, Δx is the distance travelled between $x(t_1)$ and $x(t_2)$, and Δt is the elapsed time for the object to travel from $x(t_1)$ to $x(t_2)$. Considering the position function $x(t)$, the average velocity between any two points x_1 and x_2 can be obtained by calculating the slope of the straight line which connects x_1 to x_2. Here we adopt the definition of speed from the 2-dimensional (2D) space of position and time and modify it to use in 2D space of fitness value and function evaluation.

Definition 2 (*fitness mobility*) The numbers of fitness evaluations in the interval between two successive environmental changes, in which the best value of the algorithm reaches to its highest level and then remains steady.

Definition 3 (*fitness stability*) The number of fitness evaluations before the next change in the environment, in which the best value obtained by the algorithm remains unchanged.

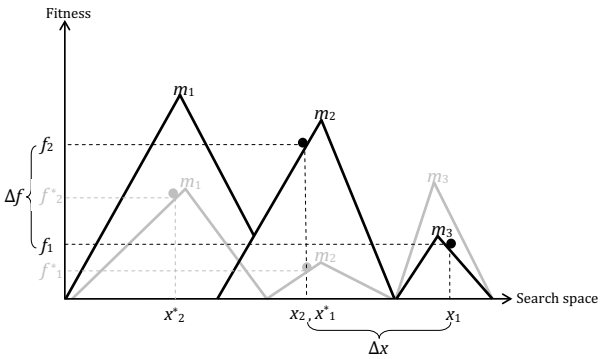

Fig. 1 Two different states of a typical 1-dimensional dynamic environment with three peaks (m_1, m_2 and m_3). Δx is the distance travelled between x_1 and x_2. Δf is the improvement in quality of the best solution in the first state of the environment ($f_2 - f_1$)

The concepts of *fitness mobility* and *fitness stability* are illustrated in Figs. 1 and 2. Figure 1 shows two different states of a typical dynamic environment with three peaks and change frequency 5000. In the first state (Black peaks), the optimization algorithm spends a number of function evaluations to get the best individual from point x_1 to point x_2 and improves the quality of the best found solution of Δf. Then a change occurs and the environment goes to the second state (Gray peaks). In second state, again the optimization algorithm spends a number of evaluations to elevate the quality of the best solution. Figure 2 illustrates the evolution of the fitness value of the best individual of the population in the first state of the environment.

Regarding the above definitions, the FAS of the algorithm between two points x_1 and x_2 is equal to the slope of the straight line which connects (f_1, 0) and (f_2, 3000) in the *fitness mobility* period (Fig. 2). Therefore, we can formulate the FAS as:

$$FAS = \tan\theta = \frac{\Delta f}{fitnessmobility} \tag{14}$$

It is clear that between two different algorithms that reach to the same best fitness in a period, the algorithm with the smaller *fitness mobility* or bigger *fitness stability* is faster. The overall average FAS of the algorithm is then calculated as follows:

$$\overline{FAS} = \frac{1}{G} \times \sum_{i=1}^{G} \left(\frac{1}{N} \times \sum_{j=1}^{N} \frac{\Delta f_{ij}}{(fitnessmobility)_{ij}} \right) \tag{15}$$

where G is the total number of environmental changes, and N is the total number of runs. Δf_{ij} and $(fitnessmobility)_{ij}$ are

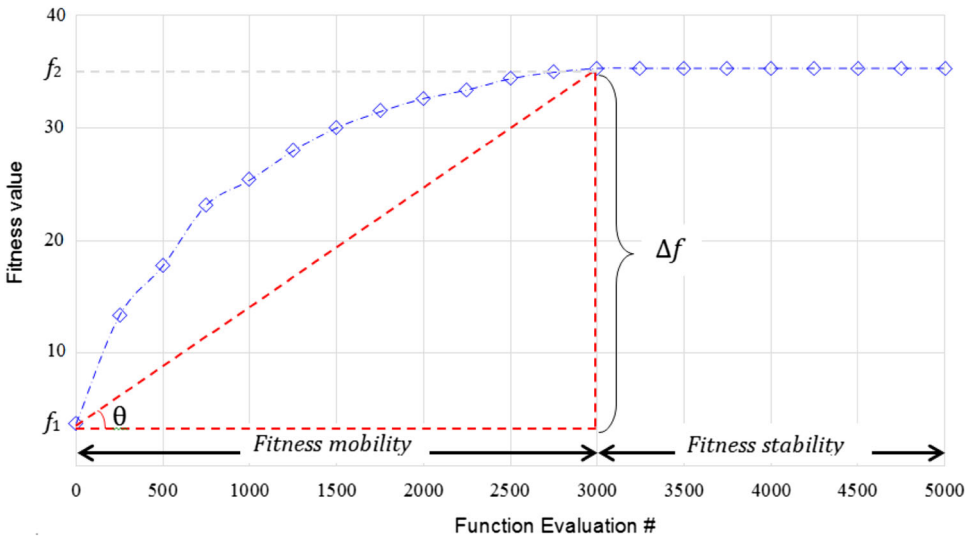

Fig. 2 Evolution of the fitness values versus the number of function evaluations for the best individual of the population in a period between two successive changes in the environment. θ is the angle between straight line connecting f_1 and f_2, and the line parallel to the function evaluation axis

the values of Δf and *fitness mobility* in the ith change period and jth run, respectively.

It should be noted that there may be several situations, e.g. rapidly or severely changing environment, where the optimization algorithm may not have enough time to reach the fitness stability stage before the next change occurs in the environment. In these situations, the FAS is calculated as follows:

$$\overline{FAS} = \frac{1}{G} \times \sum_{i=1}^{G} \left(\frac{1}{N} \times \sum_{j=1}^{N} \frac{f_{max}^{ij} - f_{init}^{ij}}{\psi_{ij}} \right) \tag{16}$$

In above equation, again G is the total number of environmental changes and N is the total number of runs. f_{max}^{ij} is the maximum fitness value achieved by the optimization algorithm in the ith change period and jth run. f_{init}^{ij} is the initial fitness value of the optimization algorithm in the ith change period and jth run. It should be mention that in Fig. 2, $f_1 = f_{init}^{ij}$. ψ_{ij} is the number of function evaluations needed for an optimization algorithm to improve the fitness value from f_{init}^{ij} to f_{max}^{ij} in the ith change period and jth run.

This measure determines how fast an algorithm can reach to its maximum performance in each state of the environment. The minimum FAS is obtained when the best value of the population remains unchanged during a period, and the maximum FAS is obtained when the best value of the population moves from *global minimum in the search space* (F_{min}) to *global maximum in the search space* (F_{max}) with spending just one function evaluation (seems unlikely in practice). Hence, the range of FAS in theory will be $\left[0, \frac{F_{max} - F_{min}}{1}\right]$.

3.2 Alpha-accuracy

This section presents the alpha-accuracy measure for assessing the accuracy of the optimization algorithms in dynamic environments. Alpha-accuracy aims to evaluate the degree of closeness of the optimization performance to a certain level during the whole optimization process, which is presented as a value in the range of [0,1] with 0 for the least and 1 for the most accurate algorithm. Moreover, this measure is able to determine the ability of different algorithms in gaining from information obtained in the former stages of the search process. The alpha-accuracy defines an acceptable error threshold at any point of optimization process. Therefore, we first need to model the acceptable error threshold during the optimization. For this purpose, we define a time-dependant function $\alpha(t)$. The value of α at each time is calculated according to the following linearly decreasing scheme:

$$\alpha(t) = \frac{(f \times \xi) - t}{(f \times \xi)} \times \left(\alpha_{initial} - \alpha_{final}\right) + \alpha_{final} \tag{17}$$

where f is the change frequency of the environment, ξ is the total number of environmental changes, t is the current evaluation number, $\alpha_{initial}$ and α_{final} are the value of α at the beginning and ending of the optimization process, respectively. Alpha-accuracy for a single run is then calculated according to Procedure 1.

Procedure 1. General procedure for calculating alpha-accuracy

1. **Input:** recorded fitness error of the best individual
2. **Output:** alpha-accuracy
3. **Let** $\{\xi_1, \xi_2, \ldots, \xi_m\}$ denotes m different states of a typical dynamic environment *DOP*
4. **Let** $\{\alpha_1, \alpha_2, \ldots, \alpha_m\}$ indicates the success or failure of the optimization algorithm corresponding to m different states of *DOP* where $\alpha_i = 0$ if the optimization algorithm fails to reach to the expected accuracy in i^{th} state of the environment, otherwise $\alpha_i = 1$
5. **begin**
6. **for each** state $\xi_i \mid i \epsilon \{1,2,\ldots,m\}$ of dynamic environment *DOP* **do**
7. **repeat**
8. Calculate $\alpha(t)$ according to Eq. (17)
9. **if** fitness error of the best individual <= $\alpha(t)$ **then**
10. $\alpha_i = 1$
11. go to the next state of the environment (i=i+1)
12. **else**
13. $\alpha_i = 0$
14. **end-if**
15. t=t+1
16. **until** next change in the environment
17. **end-for**
18. alpha-accuracy = $\frac{1}{m} \times \sum_{i=1}^{m} \alpha_i$
19. **End**

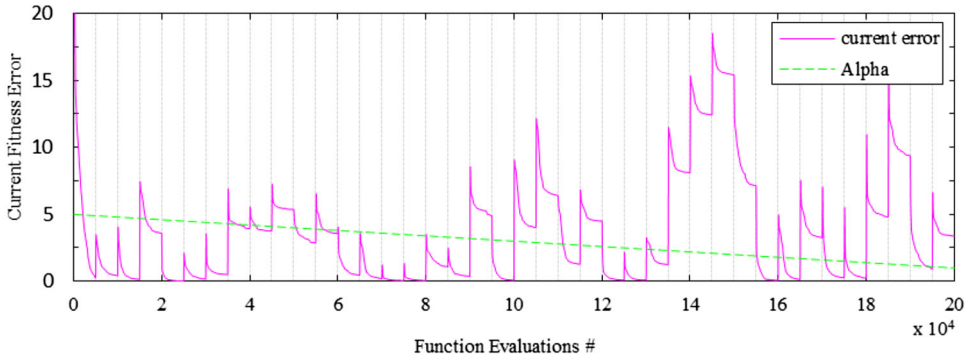

Fig. 3 Current fitness error along with alpha-accuracy threshold in an environment with 40 periods. The *vertical dotted lines* in the figure indicate the start of a new period

The above procedure calculates the number of environments where the optimization algorithm is able to reach to an acceptable level of error. The alpha-accuracy is further averaged over the total number of runs. To facilitate the understanding of alpha-accuracy, Fig. 3 shows an example of how to calculate this measure. Figure 3 depicts the evolution of the current fitness error of an optimization algorithm along with the value of alpha threshold in an environment which changes every 5000 fitness evaluations. Regarding Fig. 3, the alpha-accuracy is equal to the portion of the periods that the current fitness error of the algorithm is below the threshold alpha. Therefore, the alpha-accuracy is equal to $\frac{27}{40} = 0.675$. In Fig. 3, the values of $\alpha_{initial}$ and α_{final} are 5 and 1, respectively. The initial and final value of alpha depends on the application where the optimization algorithm is applied to.

4 Experimental study

In this section we first describe the features of the MPB. Then, we briefly describe a number of state-of-the-art algorithms used in this study. Finally, obtained results by different methods are presented and compared with each other via several metrics. The experiments are divided into two parts, where the first one compares different algorithms and the second one compares different variations of the same algorithm. Our purpose is essentially to demonstrate the advantages of our proposed set of measures.

4.1 Experimental setup

4.1.1 Dynamic test function

One of the most widely used synthetic dynamic optimization test suites in the literature is the MPB proposed by Branke (1999), which is highly regarded due to its configurability. MPB is a real-valued dynamic environment with a D-dimensional landscape consisting of N peaks, where the height, the width and the position of each peak is changed slightly every the time a change occurs in the environment. Different landscapes can be defined by specifying the shape of the peaks. A common peak shape is conical which is defined as follows:

$$f(\vec{x},t) = \max_{i=1,\ldots,N} H_t(i) - W_t(i)\sqrt{\sum_{j=1}^{D}(x_t(j) - X_t(i,j))^2} \tag{18}$$

where $H_t(i)$ and $W_t(i)$ are the height and the width of peak i at time t, respectively. The coordinates of each dimension $j \in [1, D]$ related to the location of peak i at time t, are expressed by $X_t(i,j)$, and D is the problem dimensionality. A typical change of a single peak can be modeled as follows:

$$H_{t+1}(i) = H_t(i) + height_{severity}.\sigma_h \tag{19}$$

$$W_{t+1}(i) = W_t(i) + width_{severity}.\sigma_w \tag{20}$$

$$\vec{X}_{t+1}(i) = \vec{X}_t(i) + \vec{v}_{t+1}(i) \tag{21}$$

$$\vec{v}_{t+1}(i) = \frac{s}{|\vec{r} + \vec{v}_t(i)|}((1-\lambda)\vec{r} + \lambda\vec{v}_t(i)) \tag{21}$$

where σ_h and σ_w are two random Gaussian numbers with zero mean and standard deviation one. Moreover, the shift vector $\vec{v}_t(i)$ is a linear combination of a random vector $\vec{r}$, which is created by drawing random numbers in $[-0.5, 0.5]$ for each dimension, and the current shift vector $\vec{v}_t(i)$, and normalized to the length s. Parameter $\lambda \in [0.0, 1.0]$ specifies the correlation of each peak's changes to the previous one. This parameter determine the trajectory of changes, where $\lambda = 0$ means that the peaks are shifted in completely random directions and $\lambda = 1$ means that the peaks always follow the same direction, until they hit the boundaries where they bounce off (Table 2).

4.1.2 Selected algorithms to solve MPB

In this subsection we introduce algorithms used in this study. These algorithms include RPSO (Hu and Eberhart 2002), mQSO (Blackwell and Branke 2006), AmQSO

Table 2 Default parameter settings for the MPB

Parameter	Default value
Number of peaks (m)	10
Height severity	7
Width severity	1
Peak function	Cone
Number of dimensions (d)	5
Height range	$\in$ [30, 70]
Width range	$\in$ [1, 12]
Standard height	50
Standard width	0
Search space range	$[0, 100]^d$
Frequency of change (f)	5000
Shift severity (s)	1
Correlation coefficient	0
Change step size	Constant
Basis function	FALSE

(Blackwell and Branke 2004), and HmSO (Kamosi et al. 2010b), which we will briefly describe in the following.

The first algorithm is re-randomization PSO (RPSO) proposed by Hu and Eberhart (Hu and Eberhart 2002). In RPSO, upon detecting a change in the environment, the entire or a part of the swarm is randomly relocated in the search space. Their results suggest that the re-randomization of a small portion of the swarm (i.e. 10% in their study) can improve the ability of the canonical PSO in tracking the shifting optimum, when the step size of the environmental change is small.

Blackwell and Branke (2006) introduced a multi-swarm algorithm based on the atomic metaphor, named mQSO. The mQSO starts with a predefined number of swarms, typically equal to the number of peaks in the landscape, exploring in the search space to locate the optima. In mQSO, each swarm is composed of a number of *neutral* particles and a number of *quantum* particles. Neutral particles update their velocity and position according to the principles of pure PSO. On the other hand, quantum particles do not move according to the PSO dynamics but change their positions around the center of the best particle of the swarm according to a random uniform distribution with radius r_{cloud}. Consequently, they never converge, but provide the swarm with a sustainable level of diversity needed for tracking the shifting optimum. They also introduced two operators to establish two forms of interactions between swarms. The first operator was named *exclusion*, which prevents populations from settling on the same peak by reinitializing the worst performing swarm in the search space. The second operator was referred to as *anti-convergence* which is triggered whenever all swarms converged, and removes the worst swarm from its peak and reinitialize it in the search space. In another work, Blackwell (Blackwell and Branke 2004) introduced a self-adaptive version of mQSO, called AmQSO, which dynamically adjusts the number of swarms either by spawning new swarms into the search space, or by destroying redundant swarms.

Kamosi et al. (2010b) proposed a multi-swarm algorithm for dynamic environments, which consists of a parent swarm to explore the search space and a varying number of child swarms to exploit promising areas found by the parent swarm. In this method, called HmSO, the optimization process starts with a parent swarm exploring the entire search space for locating promising areas of the search space. Whenever the fitness value of the best particle in the parent swarm improves, a child swarm will be created as follows: (a) a number of particles located within the radius r from the best particle of the parent swarm are separated from the parent swarm and form a new child swarm (b) a number of particles are randomly initialized in a hyper-sphere with radius $r/3$ centered at the best particle of the child swarm. Afterwards, a number of randomly initialized particles are added to the parent swarm in order to compensate for the migration of particles from parent swarm to the child swarm. After creating child swarms, each of them exploits their respective sub-regions in the search space. Upon detecting a collision between two child swarms, the worse child swarm is removed. In case that a particle of the parent swarm finds a better position in the search territory of a child swarm, it replaces the local best particle of the respective child swarm. Afterwards, the particle of the parent swarm is reinitialized in the search space. Moreover, inspired by hibernation phenomenon in animals, unproductive search in child swarms is stopped by "sleeping" converged child swarms. This hibernation mechanism makes more function evaluations available for the parent swarm to explore the search space, and for the other child swarms to exploit their respective areas in the landscape.

Yang and Li (2010) employed a single linkage hierarchical clustering method in CPSO to create an adaptive number of sub-swarms, and assign each of them to different promising sub-regions of the search space. Each created sub-swarm then uses the PSO with a modified gbest model to exploit the respective region. In order to avoid different clusters from crowding in the same area, they have applied a redundancy control check. In this regard, when two sub-swarms are located on a single peak, first they are merged together and then the worst performing individuals of the sub-swarm are removed until its size is equal to a pre-defined threshold.

The parameters setting for each tested algorithm is given in Table 3.

Table 3 Parameter settings for different PSO-based algorithms used in this study

Algorithms	Setting
RPSO	Swarm size = 100, $v_{clamp} = [-20, 20]$, the rate of re-randomization = 50%, $c_1 = c_2 = 1.49$, $w = 0.72984$
mQSO	mQSO with configuration 10 ($5 + 5^q$) was adopted, $r_{excl} = r_{conv} = 31.5$, $c_1 = c_2 = 2.05$, $\chi = 0.72984$, $r_q = 0.5 \times s$
AmQSO	The number of neutral particles in each swarm = 5, number of quantum particles in each swarm = 1, $n_{excess} = 1$, $c_1 = c_2 = 2.05$, $\chi = 0.72984$, $r_q = 0.5 \times s$
HmSO	The number of particles in the parent swarm = 5, number of particles in each child swarm = 10, $c_1 = c_2 = 1.49$, $\chi = 0.72984$, $v_{init} = [-10, 10]$, $\xi = 5$, $r_{init} = 10$, $r_{collision} = 30$, $r_{conv} = 1$, $r_{rls} = 0.5 \times s$
CPSO	initial swarm size of the cradle swarm (M) = {100, 70, 50}, *max_subsize* (N) = {2, 3, 4}, $R_{overlap} = 0.7$, $w_{max} = 0.6$, $w_{min} = 0.3$, $c_1 = c_2 = 1.7$

4.1.3 Performance measures

In order to validate the usefulness of our proposed set of performance measures, we conduct a comparative study along previously mentioned methods on MPB via the *Offline performance* (Branke 2002), *modified offline error* (Branke 2002), *accuracy* (Weicker 2002), *best-error-before-change* (Li et al. 2008) and our proposed set of measures (*alpha-accuracy* and *FAS*).

4.1.4 Experimental settings

For each experiment of an algorithm on a specific DOP, 100 independent runs are executed with different random seeds and the results are reported in terms of average and standard deviation. For each run of an algorithm, 100 environmental changes are considered as the termination condition which results in $f \times 100$ function evaluations.

4.2 Comparison among different algorithms

The experimental results of four PSO variants on different dynamic environments are presented in Table 4. In Table 4, the results of the best performing algorithm, which are significantly superior to the others with a t test at 5% significant level, are printed in boldface. In order to counteract the effect of type I error, Holm–Bonferroni correction has been applied to control the family-wise error rate. In situations where the outcomes of the t-test reveal that the results of the best performing algorithms are not statically significant, all of them are printed in boldface.

4.2.1 Offline performance

Figure 4 shows the average offline fitness of different methods across 20,000 fitness evaluations on MPB with *default* dynamics. Considering Fig. 4, it can be observed that RPSO is clearly the worst performing method. Among the other contestant algorithms, AmQSO and HmSO are able to grow faster. However, after 50,000 evaluations, mQSO, AmQSO and HmSO shows a very similar behavior. From Fig. 4, it is difficult to draw conclusions about the effectiveness of the compared algorithms. In particular, curves can be hardly identified and the Figure does not indicate the complete picture of the algorithms performance. Therefore, numerical results are preferable. The reported offline performances in

Table 4 Numerical comparison of RPSO, mQSO, AmQSO, and HmSO with respect to different measures on various dynamic environments generated by MPB

Measure*	Dynamic	Algorithms			
		RPSO	mQSO	AmQSO	HmSO
Offline performance	*default*	51.0486 ± 0.41	64.5152 ± 0.09	64.6442 ± 0.09	**64.9711 ± 0.08**
	$s = 3$	50.4293 ± 0.36	63.3802 ± 0.09	**63.4283 ± 0.10**	**63.4043 ± 0.09**
	$s = 5$	49.4959 ± 0.36	62.3351 ± 0.10	**62.4816 ± 0.11**	62.1171 ± 0.09
	$f = 500$	40.6582 ± 0.43	57.3230 ± 0.21	59.3151 ± 0.16	**59.5909 ± 0.11**
	$f = 2500$	49.4702 ± 0.38	63.4349 ± 0.11	63.6357 ± 0.10	**63.9573 ± 0.08**
Modified offline error	*default*	15.20 ± 0.33	1.77 ± 0.05	1.64 ± 0.05	**1.42 ± 0.03**
	$s = 3$	16.19 ± 0.34	2.90 ± 0.05	2.86 ± 0.06	**2.80 ± 0.05**
	$s = 5$	16.35 ± 0.33	3.95 ± 0.06	**3.80 ± 0.07**	4.08 ± 0.05

Table 4 continued

Measure*	Dynamic	Algorithms			
		RPSO	mQSO	AmQSO	HmSO
	$f = 500$	24.88 ± 0.41	8.96 ± 0.19	6.97 ± 0.13	**6.77 ± 0.10**
	$f = 2500$	16.63 ± 0.35	2.85 ± 0.07	2.65 ± 0.05	**2.29 ± 0.04**
Accuracy	*default*	0.56 ± 0.01	0.94 ± 0.003	0.95 ± 0.003	**0.98 ± 0.001**
	$s = 3$	0.58 ± 0.01	0.92 ± 0.002	0.93 ± 0.003	**0.97 ± 0.001**
	$s = 5$	0.60 ± 0.01	0.91 ± 0.002	0.92 ± 0.003	**0.97 ± 0.001**
	$f = 500$	0.31 ± 0.01	0.65 ± 0.01	0.73 ± 0.01	**0.86 ± 0.003**
	$f = 2500$	0.53 ± 0.01	0.89 ± 0.04	0.91 ± 0.03	**0.97 ± 0.001**
Best-error-before-change	*default*	13.23 ± 0.31	0.84 ± 0.05	0.73 ± 0.05	**0.47 ± 0.03**
	$s = 3$	12.27 ± 0.31	1.05 ± 0.04	0.91 ± 0.05	**0.66 ± 0.04**
	$s = 5$	12.50 ± 0.33	1.35 ± 0.04	1.06 ± 0.05	**0.94 ± 0.05**
	$f = 500$	21.93 ± 0.37	4.91 ± 0.15	**3.84 ± 0.12**	4.18 ± 0.09
	$f = 2500$	15.38 ± 0.37	1.46 ± 0.06	1.25 ± 0.05	**0.81 ± 0.04**
Alpha-accuracy(20,10)†	*default*	0.4751 ± 0.0133	0.9832 ± 0.0022	0.9838 ± 0.0019	**0.9884 ± 0.0007**
	$s = 3$	0.4813 ± 0.0124	0.9832 ± 0.0020	0.9819 ± 0.0022	**0.9898 ± 0.0007**
	$s = 5$	0.4918 ± 0.0135	0.9822 ± 0.0019	0.9816 ± 0.0022	**0.9888 ± 0.0006**
	$f = 500$	0.1696 ± 0.0090	0.8705 ± 0.0080	0.9149 ± 0.0069	**0.9634 ± 0.0032**
	$f = 2500$	0.4046 ± 0.0126	0.9684 ± 0.0030	0.9735 ± 0.0026	**0.9828 ± 0.0011**
Alpha-accuracy(10,5)	*default*	0.3082 ± 0.0101	0.9445 ± 0.0045	0.9481 ± 0.0042	**0.9781 ± 0.0016**
	$s = 3$	0.3131 ± 0.0108	0.9446 ± 0.0041	0.9456 ± 0.0046	**0.9722 ± 0.0024**
	$s = 5$	0.3263 ± 0.0101	0.9345 ± 0.0040	0.9407 ± 0.0049	**0.9606 ± 0.0100**
	$f = 500$	0.0483 ± 0.0034	0.7596 ± 0.0098	0.7873 ± 0.0087	**0.8452 ± 0.0111**
	$f = 2500$	0.2355 ± 0.0103	0.9160 ± 0.0050	0.9254 ± 0.0047	**0.9574 ± 0.0101**
Alpha-accuracy(5,2.5)	*default*	0.2221 ± 0.0092	0.9021 ± 0.0058	0.9155 ± 0.0060	**0.9477 ± 0.0041**
	$s = 3$	0.2424 ± 0.0094	0.8944 ± 0.0055	0.9040 ± 0.0066	**0.9441 ± 0.0042**
	$s = 5$	0.2583 ± 0.0084	0.8641 ± 0.0056	0.8992 ± 0.0071	**0.9216 ± 0.0051**
	$f = 500$	0.0043 ± 7.68E−04	0.5710 ± 0.0102	0.5578 ± 0.0099	**0.6081 ± 0.0096**
	$f = 2500$	0.1392 ± 0.0072	0.8599 ± 0.0066	0.8776 ± 0.0062	**0.9175 ± 0.0050**
Alpha-accuracy(2.5,1.25)	*default*	0.1822 ± 0.0071	0.8684 ± 0.0072	0.8900 ± 0.0077	**0.9166 ± 0.0056**
	$s = 3$	0.1748 ± 0.0075	0.8347 ± 0.0073	0.8644 ± 0.0080	**0.9126 ± 0.0055**
	$s = 5$	0.1862 ± 0.0074	0.7513 ± 0.0070	0.8282 ± 0.0085	**0.8826 ± 0.0069**
	$f = 500$	0.0003 ± 0.0001	0.2929 ± 0.0100	0.2919 ± 0.0090	**0.3376 ± 0.0100**
	$f = 2500$	0.0690 ± 0.0042	0.8021 ± 0.0082	0.8234 ± 0.0077	**0.8962 ± 0.0052**
FAS	*default*	0.0073 ± 1.40E−03	0.0100 ± 1.65E−04	0.0115 ± 2.10E−04	**0.0185 ± 2.67E−04**
	$s = 3$	0.0044 ± 2.90E−04	0.0126 ± 1.69E−04	0.0134 ± 1.75E−04	**0.0188 ± 3.97E−04**
	$s = 5$	0.0049 ± 2.46E−04	0.0150 ± 1.75E−04	0.0151 ± 2.24E−04	**0.0185 ± 2.48E−04**
	$f = 500$	**1.1160 ± 3.54E−02**	0.1853 ± 4.10E−03	0.1681 ± 3.40E−03	0.1816 ± 6.40E−03
	$f = 2500$	**0.0830 ± 8.80E−03**	0.0237 ± 4.95E−04	0.0290 ± 5.93E−04	0.0389 ± 7.08E−04

Results are the mean ± standard error over 100 runs

The best results obtained for each DOP is highlighted in boldface

* Lower values for offline performance, modified offline error and best-error-before-change indicate better optimization process. Higher values for accuracy, alpha-accuracy and FAS mean better optimization process

† Alpha-accuracy (a, b) means that the value of *alpha* is reduced from a to b during the optimization process

Table 4 indicate that HmSO and AmQSO show a better performance on the tested dynamic environments. Among these two algorithms, HmSO performs better. Regarding Table 4, in most cases there is a slightly difference between mQSO, AmQSO, and HmSO.

4.2.2 *Modified offline error*

The results of offline error are slightly different from those of offline performance. A closer look at Table 4 shows that the difference between the performance of RPSO and other

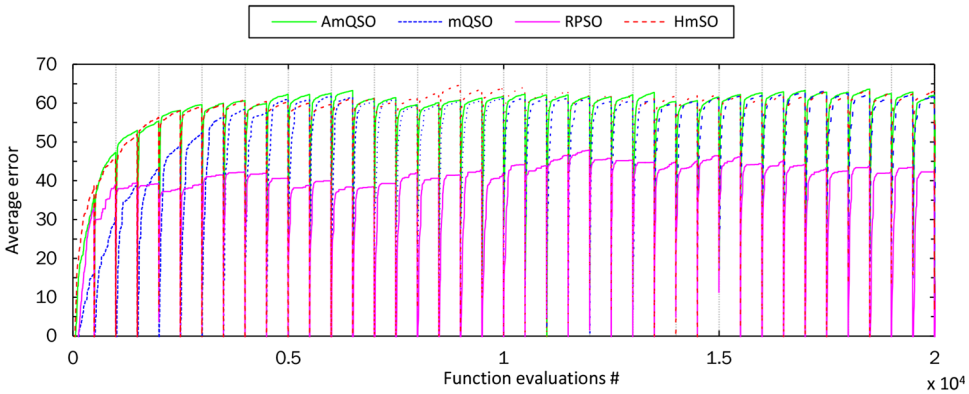

Fig. 4 Current fitness error along with alpha-accuracy threshold in an environment with 40 periods

contestants using offline error is more significant. For example, in dynamic environment with $f = 500$, RPSO is 7.26 time worse than HmSO. However, according to offline performance, this difference is only 1.29 times.

4.2.3 Accuracy

As mentioned in Sect. 2, the computation of this measure requires information about the Max_F^t and Min_F^t. The value of Max_F^t in MPB is the fitness of the highest peak in the landscape at time t. We also compute the value of Min_F^t for each period of the environment. It is worth noticing that computation of Min_F^t impose additional computation complexity to the system. The accuracy of the tested algorithms at the moment of change (t is the last time step of each period) is reported in Table 4.

From Table 4, we can observe that all tested algorithms, except RPSO, achieve good results. Regarding mQSO and AmQSO on DOP with *default* environmental dynamics, $s = 3$, and $s = 5$, it is difficult to determine the best performing algorithm among them. Also, the result of accuracy does not confirm the previous interpretations of the results obtained by offline performance and modified offline error. It should be noted that when measuring the performance of different algorithms on DOP with $s = 3$ using offline performance and modified offline error, AmQSO is the best performing algorithm. However, it is observed in Table 4 that HmSO is the best performing PSO variants in terms of accuracy.

4.2.4 Best-error-before-change

As can be seen in Table 4, there is a bigger difference between tested algorithms using best-error-before-change measure than the other measures. For example, the offline performance, modified offline error and accuracy for the dynamic environment with $s = 3$, the difference between mQSO, AmQSO, and HmSO is not very severe. However, when we consider best-error-before-change as a measure this difference becomes more sever. Although offline error is a generally accepted performance measure in the literature, it does not provide a meaningful interpretation about the superiority of one method over the other. Besides, it is difficult to understand how better an algorithm can perform in comparison with another method.

4.2.5 Alpha-accuracy and fitness adaptation speed

In this section, we elaborately interpret the results of alpha-accuracy and FAS. Our goal is to specifically emphasize on the relative advantages of our proposed measures.

The very first point which is obvious from Table 4 is that increasing the desired optimization accuracy level will result in decreasing the alpha-accuracy. As can be observed, when we change the range of alpha from alpha(20,10) to alpha(10,5) and alpha(5,2.5), the obtained results of all algorithms are degraded. Regarding Table 4, when we change the range of alpha from alpha(20,10) to alpha(10,5), the results of HmSO degrades just 1.05%. For RPSO, mQSO and AmQSO this degradation in the alpha-accuracy is 35.13, 3.94 and 3.63%, respectively. Therefore, by controlling the value of the alpha we can obtain more information about the performance of different algorithms.

Regarding FAS in Table 4, HmSO is the fastest algorithm on 3 out of 5 dynamic environments, i.e. *default*, $s = 3$ and $s = 5$. On fast changing environments, i.e. $f = 500$ and $f = 2500$, RPSO has the fastest speed. However, this speed is at the cost of loosing accuracy.

Table 5 shows the outcome of the pairwise comparison between different PSO-based methods. From Table 5, the first conclusion is that RPSO is outperformed by all other contestants in terms of alpha-accuracy. From the obtained results, it is also clear that HmSO is the best performing method in terms of both alpha-accuracy and FAS. Result of our experiments reveal that the most accurate algorithm is not necessarily the fastest one (in terms of FAS), and vice versa.

Table 5 Results of the pairwise comparison between different PSO variants in terms of (alpha-accuracy(2.5,1.25), FAS)

	Dynamics	mQSO	AmQSO	HmSO
RPSO	*default*	($\triangledown$, ▲)	($\triangledown$, ▲)	($\triangledown$, $\triangledown$)
	$s = 3$	($\triangledown$, ▲)	($\triangledown$, ▲)	($\triangledown$, $\triangledown$)
	$s = 5$	($\triangledown$, ▲)	($\triangledown$, ▲)	($\triangledown$, $\triangledown$)
	$f = 500$	($\triangledown$, ▲)	($\triangledown$, ▲)	($\triangledown$, ▲)
	$f = 2500$	($\triangledown$, ▲)	($\triangledown$, ▲)	($\triangledown$, ▲)
HmSO	*default*	(▲, ▲)	(▲, ▲)	
	$s = 3$	(▲, ▲)	(▲, ▲)	
	$s = 5$	(▲, ▲)	(▲, ▲)	
	$f = 500$	(▲, $\triangledown$)	(▲, ▲)	
	$f = 2500$	(▲, ▲)	(▲, ▲)	
AmQSO	*default*	(▲, ▲)		
	$s = 3$	(▲, ▲)		
	$s = 5$	(▲, ▲)		
	$f = 500$	($\approx$, $\triangledown$)		
	$f = 2500$	(▲, ▲)		

"▲" and "$\triangledown$" 0.05 level of significance by *t*-test. ''▲'', ''$\triangledown$'' and ''$\approx$'' the performance of the corresponding algorithm in the row is better than, worse than, and similar to that of the algorithm in the column, respectively

Comparing AmQSO and mQSO on environment with $f = 500$, we can observe that the alpha-accuracy of both algorithms are not statically significant but mQSO is faster. Therefore, mQSO is more preferable for optimization of this environment because it can reach to the expected accuracy quicker. Hence, choosing an appropriate algorithm for an application depends on the problem's demands.

4.2.6 *A conceptual comparison among different performance measures*

Several conclusions can be drawn from the results provided in the previous section which can be summarized as follows:

(1) Offline performance, modified offline error, accuracy and best-error-before-change do not allow meaningful conclusions about the superiority of a method over the others. In some experiments, the interpretations of these measures appear to be in complete contradiction with each other. For example, comparing AmQSO and HmSO on DOP with $s = 500$, one can see that AmQSO outperforms HmSO using offline performance and modified offline error whereas HmSO outperforms AmQSO using accuracy and best-error-before-change.

(2) Unlike accuracy (Weicker 2002), the alpha-accuracy is flexible and can provide more detail about the performance of different methods. Regarding Table 4, it is observed that AmQSO is superior to mQSO on all tested DOPs using accuracy measure. However, considering alpha-accuracy, we can see that the interpretation of the results depends on expected performance threshold. It means that if the expected threshold is linearly decreased from 20 to 10, i.e. our performance measure is alpha-accuracy(20,10), on DOP with default environmental dynamics AmQSO and mQSO can almost provide an identical performance.

(3) In many real-world applications time is a critical factor. Therefore, the speed of optimization process is also very important. Here FAS can be used to investigate how fast the solutions are provided by an optimizer on DOPs. This measure is specifically useful when two methods have identical optimality. For example, comparing mQSO and AmQSO on DOP with $f = 500$ (Table 5), it is observed that both methods has identical performance using alpha-accuracy(2.5,1.25). However, the FAS of mQSO is better than that of AmQSO therefore mQSO is preferable.

(4) All measures in Table 4, i.e. offline performance, modified offline error, best-error-before-change and accuracy, examine different algorithms only from the optimality point of view. However, our proposed set of measures is interpreted together and evaluates the optimality and behaviour of different methods simultaneously.

4.3 Comparison among different variations of the same algorithm

The performance of different variations of CPSO were evaluated on various DOPs with $f = 5000$, $m = 10$, $d \in \{5, 10, 20, 30, 40, 50\}$, and $s \in \{1, 2, 3, 4, 5\}$. Table 6 shows the obtained results by CPSO(100,4), CPSO(70,3) and CPSO(50,2) in different environments.

Different conclusion can be drawn from Table 6. Here we highlight a number of important conclusions as follows:

- As can be observed In Table 6, CPSO(100,4) is the fastest yet least accurate algorithm.
- In most of DOPs, CPSO(70,3) is the most accurate method (with respect to alpha-accuracy).
- Some very interesting point about performance comparison among different variations of CPSO can be observed in Table 6. For instance, in the environment with $s = 1$ and $d = 20$, not only CPSO(70,3) is more accurate than CPSO(50,2), but it is approximately fifteen times faster.

Table 6 Fitness adaptation speed and alpha-accuracy of CPSO(100,4), CPSO(70,3) and CPSO(50,2) for varying number of dimensions and change severity in environments with $f = 5000$ and $m = 10$

Configuration		CPSO(100,4)				CPSO(70,3)				CPSO(50,2)			
s	d	FAS	alpha-accuracy (10,5)	alpha-accuracy (14,7)	alpha-accuracy (20,10)	FAS	alpha-accuracy (10,5)	alpha-accuracy (14,7)	alpha-accuracy (20,10)	FAS	alpha-accuracy (10,5)	alpha-accuracy (14,7)	alpha-accuracy (20,10)
1	5	**0.0017**	**0.**9334	**0.**9662	**0.**9892	0.0013	**0.9398**	**0.**9642	**0.**9868	0.0012	**0.**9276	**0.9668**	**0.9894**
	10	**0.0045**	**0.**6708	**0.**7830	**0.**8866	0.0019	**0.**7584	**0.**8500	**0.**9302	0.0027	**0.7758**	**0.8624**	**0.9358**
	20	0.0281	**0.**3692	**0.**4898	**0.**6278	**0.0289**	**0.5642**	**0.6960**	**0.8220**	0.0020	**0.**5538	**0.**6796	**0.**8090
	30	**0.0989**	**0.**1724	**0.**2606	**0.**3796	0.0157	**0.3872**	**0.5130**	**0.6662**	0.0142	**0.**3472	**0.**4784	**0.**6272
	40	**0.1110**	**0.**4140	**0.**6540	**0.**1094	0.0416	**0.2358**	**0.3544**	**0.5016**	0.0156	**0.**2248	**0.**3488	**0.**4980
	50	**0.0968**	**0.**0480	**0.**1000	**0.**1780	0.0581	**0.**1420	**0.**2242	**0.**3364	0.0174	**0.1464**	**0.2348**	**0.3548**
2	5	**0.0020**	**0.**9058	**0.**9530	**0.**9838	0.0015	**0.9360**	**0.9662**	**0.9888**	0.0014	**0.**9212	**0.**9654	**0.**9872
	10	**0.0055**	**0.**6004	**0.**7314	**0.**8552	0.0021	**0.7094**	**0.8222**	**0.9128**	0.0019	**0.**6560	**0.**7864	**0.**8894
	20	**0.0211**	**0.**2650	**0.**3920	**0.**5498	0.0036	**0.4290**	**0.5656**	**0.7340**	0.0024	**0.**3828	**0.**5264	**0.**6982
	30	**0.0761**	**0.**1342	**0.**2180	**0.**3272	0.0248	**0.2890**	**0.4280**	**0.5904**	0.0126	**0.**2064	**0.**3208	**0.**4850
	40	**0.0914**	**0.**2020	**0.**4060	**0.**7940	0.0439	**0.1594**	**0.2596**	**0.4234**	0.0071	**0.**1068	**0.**2100	**0.**3500
	50	**0.0853**	**0.**0040	**0.**0460	**0.**0940	0.0302	**0.**7460	**0.**1348	**0.**2446	0.0371	**0.7680**	**0.1476**	**0.2620**
3	5	**0.0024**	**0.**8758	**0.**9370	**0.**9738	0.0017	**0.9118**	**0.9574**	**0.9824**	0.0015	**0.**8896	**0.**9386	**0.**9770
	10	**0.0046**	**0.**5452	**0.**6824	**0.**8216	0.0026	**0.6352**	**0.7574**	**0.8706**	0.0021	**0.**5946	**0.**7348	**0.**8588
	20	**0.0201**	**0.**2418	**0.**3618	**0.**5170	0.0059	**0.3616**	**0.5124**	**0.6838**	0.0027	**0.**2936	**0.**4290	**0.**6042
	30	**0.0672**	**0.**7460	**0.**1438	**0.**2538	0.0089	**0.1996**	**0.3188**	**0.4780**	0.0037	**0.**1620	**0.**2778	**0.**4434
	40	**0.1399**	**0.**0880	**0.**2380	**0.**4660	0.0139	**0.1030**	**0.1800**	**0.2972**	0.0073	**0.**6980	**0.**1546	**0.**2740
	50	**0.0531**	**0.**0080	**0.**0340	**0.**0940	0.0299	**0.4940**	**0.1052**	**0.**1928	0.0258	**0.**4600	**0.**1014	**0.2032**
4	5	**0.0027**	**0.**8672	**0.**9268	**0.**9712	0.0019	**0.8918**	**0.9436**	**0.9784**	0.0017	**0.**8478	**0.**9142	**0.**9642
	10	**0.0036**	**0.**5108	**0.**6426	**0.**7870	0.0029	**0.5774**	**0.7196**	**0.8458**	0.0023	**0.**5672	**0.**7036	**0.**8384
	20	**0.0259**	**0.**1790	**0.**2916	**0.**4474	0.0033	**0.3202**	**0.4720**	**0.6368**	0.0030	**0.**2552	**0.**3880	**0.**5800
	30	**0.0640**	**0.**5560	**0.**1147	**0.**2187	0.0041	**0.1708**	**0.2870**	**0.4486**	0.0038	**0.**1138	**0.**2200	**0.**3694
	40	**0.0964**	**0.**0680	**0.**1880	**0.**3920	0.0361	**0.7120**	**0.1438**	**0.2622**	0.0060	**0.**5180	**0.**1168	**0.**2294
	50	**0.1559**	**0.**0040	**0.**0260	**0.**0680	0.0472	**0.3540**	**0.8520**	**0.1796**	0.0087	**0.**2160	**0.**6320	**0.**1592
5	5	**0.0028**	**0.**8618	**0.**9214	**0.**9656	0.0019	**0.8880**	**0.9412**	**0.9780**	0.0016	**0.**8488	**0.**9192	**0.**9702
	10	**0.0060**	**0.**4366	**0.**5734	**0.**7352	0.0029	**0.5910**	**0.7108**	**0.8302**	0.0025	**0.**4968	**0.**6538	**0.**8128
	20	**0.0318**	**0.**1340	**0.**2248	**0.**3766	0.0096	**0.2852**	**0.4256**	**0.5898**	0.0030	**0.**1746	**0.**3096	**0.**4890
	30	**0.0727**	**0.**4260	**0.**9460	**0.**1914	0.0073	**0.1556**	**0.2576**	**0.4118**	0.0043	**0.**7020	**0.**1508	**0.**2908
	40	**0.0633**	**0.**0760	**0.**1600	**0.**3840	0.0270	**0.5480**	**0.1284**	**0.2432**	0.0097	**0.**2560	**0.**7340	**0.**1700
	50	**0.1018**	**0.**0040	**0.**0200	**0.**0780	0.0723	**0.2140**	**0.6260**	**0.1402**	0.0082	**0.**0960	**0.**4000	**0.**1080

The best results obtained for each DOP is highlighted in boldface

- It is evident by the results that the alpha-accuracy of tested algorithms is more sensitive to the dimensionality of problem. It can be observed in Table 6 that the alpha-accuracy of algorithms is less affected by the severity of changes.

Experimental results show that the proposed set of measures can provide a good tool for judging about the performance of different methods and choosing a suitable optimizer for specific DOPs with respect to expected accuracy and speed of optimization.

5 Conclusions and future works

This paper presents a set of two measures for evaluating the speed and accuracy of optimization algorithms in dynamic environments. Inspired by the concept of velocity in physics, FAS was introduced to assess the speed of optimization algorithms in dynamic environments. The other measure is alpha–accuracy which determines the accuracy of dynamic optimization algorithms in dynamic environments. This measure specifies an acceptable threshold of error for optimization algorithms at each time, which also considers the ability of different algorithms in using obtained information from former stages of the optimization process. Unlike accuracy (Weicker 2002), the computation of alpha-accuracy does not need information about minimum fitness value of the search space, which is an advantage of the proposed alpha-accuracy. Moreover, the alpha-accuracy is more flexible and can be adjusted according to the application. Different state-of-the-art algorithms were then compared with the set of proposed measures on MPB. Experimental study confirmed that the collection of these two measures is a good criterion to choose a suitable algorithm for a specific problem with respect to trade-off between the speed and accuracy.

Future research may focus on using other types of functions for threshold alpha, e.g. logarithmic function. Moreover, the set of proposed measures can be further extended by including other measures, e.g. degradation, to take more aspects of the optimization algorithms into account.

Acknowledgement The authors are grateful to Dr. A.B. Hashemi for letting us use the source code of HmSO.

References

Alba E, Sarasola B (2010) ABC, a new performance tool for algorithms solving dynamic optimization problems. In: IEEE congress on evolutionary computation (CEC), pp 1–7

Alba E, Sarasola B, Di Chio C (2010) Measuring fitness degradation in dynamic optimization problems. In: Applications of evolutionary computation. Springer, Heidelberg, pp 572–581

Alizadeh M, Meybodi MR, Rezvanian A (2013) Solving moving peak problem using a fuzzy particle swarm optimization based memetic algorithm. CSI J Comput Sci Eng 11:10–21

Ayvaz D, Topcuoglu HR, Gurgen F (2012) Performance evaluation of evolutionary heuristics in dynamic environments. Int J Appl Intell 37:130–144. doi:10.1007/s10489-011-0317-9

Blackwell TM (2005) Particle swarms and population diversity. Soft Comput 9:793–802. doi:10.1007/s00500-004-0420-5

Blackwell T, Branke J (2004) Multi-swarm Optimization in Dynamic Environments. In: Raidl GR (ed) Applications of evolutionary computing, Lecture notes in computer science, vol 3005. Springer, Berlin, pp 489–500

Blackwell T, Branke J (2006) Multiswarms, exclusion, and anti-convergence in dynamic environments. IEEE Trans Evol Comput 10:459–472. doi:10.1109/TEVC.2005.857074

Blackwell T, Branke J, Li X (2008) Particle swarms for dynamic optimization problems. In: Blum C (ed) Swarm intelligence. Springer, Berlin, pp 193–217

Branke J (1999) Memory enhanced evolutionary algorithms for changing optimization problems. In: Proceedings of the 1999 congress on evolutionary computation. Washington, DC, USA, pp 1875–1882

Branke J (2002) Evolutionary optimization in dynamic environments. Kluwer, Norwell

Cheng H, Yang S (2010) Genetic algorithms with immigrants schemes for dynamic multicast problems in mobile ad hoc networks. Eng Appl Artif Intel 23:806–819

Cruz C, González JR, Pelta DA (2010) Optimization in dynamic environments: a survey on problems, methods and measures. Soft Comput 15:1427–1448

Del Amo IG, Pelta DA, González JR, Masegosa AD (2012) An algorithm comparison for dynamic optimization problems. Appl Soft Comput 12:3176–3192. doi:10.1016/j.asoc.2012.05.021

Handa H, Chapman L, Yao X (2007) Robust salting route optimization using evolutionary algorithms. In: Yang S (ed) Evolutionary computation in dynamic and uncertain environments. Springer, Berlin, pp 497–517

Hasanzadeh M, Meybodi MR, Ebadzadeh MM (2013) Adaptive cooperative particle swarm optimizer. Appl Intell 39:397–420

Hasanzadeh M, Sadeghi S, Rezvanian A, Meybodi MR (2016) Success rate group search optimiser. J Exp Theor Artif Intell 28:53–69. doi:10.1080/0952813X.2014.971467

Hashemi AB, Meybodi MR (2009a) A multi-role cellular PSO for dynamic environments. In: Proceedings of 14th international CSI computer conference. Tehran, Iran, pp 412–417

Hashemi A, Meybodi MR (2009b) Cellular PSO: A PSO for dynamic environments. In: Cai Z (ed) Advances in computation and intelligence. Springer, Berlin, pp 422–433

Hu X, Eberhart RC (2002) Adaptive particle swarm optimization: detection and response to dynamic systems. In: Proceedings of the 2002 congress on evolutionary computation, pp 1666–1670

Kamosi M, Hashemi AB, Meybodi MR (2010a) A new particle swarm optimization algorithm for dynamic environments. In: Panigrahi BK, Das S, Suganthan PN, Dash SS (eds) Swarm, evolutionary, and memetic computing: First International Conference on Swarm, Evolutionary, and Memetic Computing, SEMCCO 2010, Chennai, 16–18, 2010 December, Proceedings. Springer, Berlin, pp 129–138

Kamosi M, Hashemi AB, Meybodi MR (2010b) A hibernating multi-swarm optimization algorithm for dynamic environments. In: Second world congress on nature and biologically inspired computing (NaBIC), pp 363–369

Kianfar S, Meybodi MR (2012) Cellular ant colony algorithm. In: Proceedings of 17th annual CSI computer conference of Iran. Tehran, Iran, pp 45–50

Kordestani JK, Ahmadi A, Meybodi MR (2014a) An improved differential evolution algorithm using learning automata and population topologies. Appl Intell 41:1150–1169

Kordestani JK, Rezvanian A, Meybodi MR (2014b) CDEPSO: a bi-population hybrid approach for dynamic optimization problems. Appl Intell 40:682–694. doi:10.1007/s10489-013-0483-z

Kordestani JK, Rezvanian A, Meybodi MR (2016) An efficient oscillating inertia weight of particle swarm optimisation for tracking optima in dynamic environments. J Expe Theor Artif Intell 28:137–149. doi:10.1080/0952813X.2015.1020521

Li X, Dam KH (2003) Comparing particle swarms for tracking extrema in dynamic environments. In: The 2003 congress on evolutionary computation, 2003, (CEC'03), pp 1772–1779

Li C, Yang S (2008) Fast multi-swarm optimization for dynamic optimization problems. In: Fourth international conference on natural computation 2008, (ICNC'08), pp 624–628

Li C, Yang S (2012) A general framework of multipopulation methods with clustering in undetectable dynamic environments. IEEE Trans Evol Comput 16:556–577. doi:10.1109/TEVC.2011.2169966

Li C, Yang S, Nguyen TT et al (2008) Benchmark generator for CEC'2009 competition on dynamic optimization

Li C, Yang S, Yang M (2012) Maintaining diversity by clustering in dynamic environments. In: IEEE congress on evolutionary computation (CEC), pp 1–8

Lung RI, Dumitrescu D (2007) A collaborative model for tracking optima in dynamic environments. In: IEEE congress on evolutionary computation, pp 564–567

Lung RI, Dumitrescu D (2010) Evolutionary swarm cooperative optimization in dynamic environments. Nat Comput 9:83–94

Nabizadeh S, Rezvanian A, Meybodi MR (2012a) A multi-swarm cellular PSO based on clonal selection algorithm in dynamic environments. In: International conference on informatics, electronics and vision (ICIEV). Dhaka, Bangladesh, pp 482–486

Nabizadeh S, Rezvanian A, Meybodi MR (2012b) Tracking extrema in dynamic environment using multi-swarm cellular PSO with local search. Int J Electron Inform 1:29–37

Nguyen TT, Yang S, Branke J (2012) Evolutionary dynamic optimization: a survey of the state of the art. Swarm Evol Comput 6:1–24

Nickabadi A, Ebadzadeh M, Safabakhsh R (2012) A competitive clustering particle swarm optimizer for dynamic optimization problems. Swarm Intell 6:177–206. doi:10.1007/s11721-012-0069-0

Noroozi V, Hashemi A, Meybodi MR (2011) CellularDE: a cellular based differential evolution for dynamic optimization problems. In: Dobnikar A (ed) Adaptive and natural computing algorithms. Springer, Berlin, pp 340–349

Noroozi V, Hashemi AB, Meybodi MR (2012) Alpinist CellularDE: a cellular based optimization algorithm for dynamic environments. In: Proceedings of the 14th international conference on Genetic and evolutionary computation conference companion (GECCO 2012). ACM, pp 1519–1520

Ranginkaman AE, Kordestani JK, Rezvanian A, Meybodi MR (2014) A note on the paper "A multi-population harmony search algorithm with external archive for dynamic optimization problems" by Turky and Abdullah. Inf Sci 288:12–14

Rezazadeh I, Meybodi M, Naebi A (2011) Adaptive particle swarm optimization algorithm for dynamic environments. In: Tan Y (ed) Advances in swarm intelligence. Springer, Berlin, pp 120–129

Rezvanian A, Meybodi MR, Kim T (2010) Tracking extrema in dynamic environments using a learning automata-based immune algorithm. In: Grid and distributed computing, control and automation. Springer, Berlin, pp 216–225

Richter H, Dietel F (2010) Change detection in dynamic fitness landscapes with time-dependent constraints. In: Second world congress on nature and biologically inspired computing (NaBIC), pp 580–585

Richter H, Yang S (2009) Learning behavior in abstract memory schemes for dynamic optimization problems. Soft Comput 13:1163–1173

Sarasola B, Alba E, Alba E (2013) Quantitative performance measures for dynamic optimization problems. In: Metaheuristics for dynamic optimization. Springer, Berlin, pp 17–33

Sharifi A, Noroozi V, Bashiri M, et al (2012) Two phased cellular PSO: A new collaborative cellular algorithm for optimization in dynamic environments. In: IEEE congress on evolutionary computation (CEC), pp 1–8

Sharifi A, Kordestani JK, Mahdaviani M, Meybodi MR (2015) A novel hybrid adaptive collaborative approach based on particle swarm optimization and local search for dynamic optimization problems. Appl Soft Comput 32:432–448

Simões A, Costa E (2008) Evolutionary algorithms for dynamic environments: prediction using linear regression and Markov chains. In: Rudolph G (ed) Parallel problem solving from nature–PPSN X. Springer, Berlin, pp 306–315

Simões A, Costa E (2009) Improving prediction in evolutionary algorithms for dynamic environments. In: Proceedings of the 11th annual conference on genetic and evolutionary computation, pp 875–882

Trojanowski K, Michalewicz Z (1999) Searching for optima in non-stationary environments. In: Proceedings of the 1999 congress on evolutionary computation (CEC 99), pp 1–5

Ursem RK (2000) Multinational GAs: multimodal optimization techniques in dynamic environments. In: Proceedings of the genetic and evolutionary computation conference, pp 19–26

Wang H, Yang S, Ip WH, Wang D (2010) A particle swarm optimization based memetic algorithm for dynamic optimization problems. Nat Comput 9:703–725

Weicker K (2002) Performance measures for dynamic environments. In: Parallel problem solving from nature—PPSN VII. Springer, pp 64–73

Woldesenbet YG, Yen GG (2009) Dynamic evolutionary algorithm with variable relocation. IEEE Trans Evol Comput 13:500–513

Yang S (2007) Explicit memory schemes for evolutionary algorithms in dynamic environments. In: Yang S (ed) Evolutionary computation in dynamic and uncertain environments. Springer, Berlin, pp 3–28

Yang S (2008) Genetic algorithms with memory-and elitism-based immigrants in dynamic environments. Evol Comput 16:385–416. doi:10.1162/evco.2008.16.3.385

Yang S, Li C (2010) A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. IEEE Trans Evol Comput 14:959–974. doi:10.1109/TEVC.2010.2046667

Yang S, Yao X (2008) Population-based incremental learning with associative memory for dynamic environments. IEEE Trans Evol Comput 12:542–561. doi:10.1109/TEVC.2007.913070

Yang S, Cheng H, Wang F (2010) Genetic algorithms with immigrants and memory schemes for dynamic shortest path routing problems in mobile ad hoc networks. IEEE Trans Syst Man Cybern Part C Appl Rev 40:52–63

Yu X, Tang K, Chen T, Yao X (2009) Empirical analysis of evolutionary algorithms with immigrants schemes for dynamic optimization. Memet Comput 1:3–24