Evolutionary Dynamic Optimization: A Survey of the State of the art

Article *in* Swarm and Evolutionary Computation · October 2012

DOI: 10.1016/j.swevo.2012.05.001

3 authors, including:

Trung Thanh Nguyen
Liverpool John Moores University
62 PUBLICATIONS **1,950** CITATIONS
SEE PROFILE

Juergen Branke
The University of Warwick
221 PUBLICATIONS **12,929** CITATIONS
SEE PROFILE

Evolutionary dynamic optimization: A survey of the state of the art

Trung Thanh Nguyen[a,*], Shengxiang Yang[b], Juergen Branke[c]

[a]*School of Engineering, Technology and Maritime Operations, Liverpool John Moores University, Liverpool L3 3AF, United Kingdom*
[b]*Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex, UB8 3PH, United Kingdom*
[c]*Warwick Business School, University of Warwick, Coventry CV4 7AL, United Kingdom*

Abstract

Optimization in dynamic environments is a challenging but important task since many real-world optimization problems are changing over time. Evolutionary computation and swarm intelligence are good tools to address optimization problems in dynamic environments due to their inspiration from natural self-organised systems and biological evolution, which have always been subject to changing environments. Evolutionary optimization in dynamic environments, or evolutionary dynamic optimization (EDO), has attracted a lot of research effort during the last twenty years, and has become one of the most active research areas in the field of evolutionary computation. In this paper we carry out an in-depth survey of the state-of-the-art of academic research in the field of EDO and other meta-heuristics in four areas: benchmark problems/generators, performance measures, algorithmic approaches, and theoretical studies. The purpose is to for the first time (i) provide detailed explanations of how current approaches work; (ii) review the strengths and weaknesses of each approach; (iii) discuss the current assumptions and coverage of existing EDO research; and (iv) identify current gaps, challenges and opportunities in EDO.

Keywords: Evolutionary computation, swarm intelligence, dynamic problem, dynamic optimization problem, evolutionary dynamic optimization

1. Introduction

Many real-world optimization problems are subject to changing conditions over time, so being able to optimize in a dynamic environment is important. Changes

*Corresponding author: Trung Thanh Nguyen, email address: T.T.Nguyen@ljmu.ac.uk, Tel: +44 151 231 2028 may affect the object function, the problem instance, and/or constraints, e.g., due to the arrival of new tasks, the breakdown of machines, the change of economic and financial conditions, and the variance of available resources, [1, 2]. Hence, the optimal solution(s) of the problem being considered may change over time.

In the literature of optimization in dynamic environments, researchers usually define optimization problems that change over time as *dynamic problems* or *time-dependent problems*. In this paper, our concern is focused on *dynamic optimization problems (DOPs)*, which are *a special class of dynamic problems that "are solved online by an optimization algorithm as time goes by"*.

Addressing DOPs is very challenging since it requires an optimization algorithm to not only locate an optimal solution(s) of a given problem but also track the changing optimal solution(s) over time when the problem changes. Evolutionary computation (EC) and swarm intelligence are good tools to address DOPs due to their inspiration from natural self-organised systems and biological evolution, which have always been subject to changing environments. The study of applying evolutionary algorithms (EAs) and similar techniques to solving DOPs is termed *evolutionary optimization in dynamic environments* or *evolutionary dynamic optimization* (EDO) in this paper [1].

It is noticeable that in many EDO studies, the terms "dynamic problems/time-dependent problems" and "DOPs" are not explicitly distinguished or are used interchangeably. In these studies, DOPs are either defined as a sequence of static problems linked up by some dynamic rules [3, 4, 5, 6, 7] or as a problem that has time-dependent parameters in its mathematical expression [8, 9, 10, 11], without explicitly mentioning whether the problems are solved online by an optimization algorithm or not. In definitions like those cited above, although the authors may assume that the problems are solved online by the algorithm as time goes by (as mentioned by the authors elsewhere or as shown by the way their algorithms solve the problems), this assumption was not captured explicitly in the definitions. However, it is necessary to distinguish a DOP from a general time-dependent problem because, no matter how the problem changes, from the perspective of an EA or an optimization algorithm in general, a time-dependent problem is only different from a static problem if it is solved in a dynamic way, i.e., the algorithm needs to take into account changes during the optimization process as time goes by [1, 12, 13]. Hence, only DOPs are relevant to EDO research.

To make it clearer and to distinguish DOPs from other types of time-dependent or dynamic problems, in this paper we propose the following definition for DOPs:

Definition 1 (Dynamic optimisation problem). *Given a dynamic problem f_t,*

[1]although its main focus is on evolutionary optimization techniques, this paper will also cover swarm intelligence and other meta-heuristic techniques used to solve DOPs

*an optimisation algorithm G to solve f_t, and a given optimisation period $\left[t^{begin}, t^{end}\right]$, f_t is called a **dynamic optimisation problem** in the period $\left[t^{begin}, t^{end}\right]$ if during $\left[t^{begin}, t^{end}\right]$ the underlying fitness landscape that G uses to represent f_t changes **and** G has to react to this change by providing new optimal solutions.*[2]

A more detailed version of this definition for DOPs was provided in [15, Chapter 4] and [16].

Although a few EDO works appeared in the early days of EC [17, 18], the field is still relatively young since most of the studies on EDO have been made in the last 20 years. During the last 20 years, especially in recent years, EDO has attracted a lot of research effort and has become one of the most active research areas in the EC community in terms of the number of activities and publications. There have been regular annual special sessions/workshops dedicated to EDO in major conferences in the field such as the Congress on Evolutionary Computation, the Evo*, and the GECCO workshops; there are several special issues on EDO in specialist journals (e.g. IEEE Transactions on Evolutionary Computation and Soft Computing); and there are a number of monographs on the topic [19, 7, 13, 20].

A number of studies have been made in the past to review the literature in the field. Some first attempts were made by Branke [21, 19]. The topic, as a part of the broader area of uncertainty and dynamic environments, was briefly surveyed and classified in 2005 in [12]. Various aspects of EDO were also covered in many PhD theses and monographs [19, 7, 13, 22, 20, 15]. Most recently, Cruz *et al.* [11] have made a detailed review on DOP studies to (a) provide an "overview of related works on DOPs on the last decade" and (b) to present a new repository about the topic in an "organized" way. The review was done based on a systematic search on search engines using some DO-related terms to find relevant references. The found references then are grouped into different categories in terms of type of publications, type of dynamism, methods, performance measures, applications and publication year. These categorisations provide some interesting statistics of the current trend of current literature, and the proportion of studies following a particular approach within each category. Some discussions about future directions of the field, based on the overview, were also provided.

All the survey studies mentioned above are very useful in summarising, classifying and providing an up-to-date overview of existing work in EDO. However, we believe that to provide researchers with a complete review of how and why existing approaches work and what are the current challenges of the field, it is necessary to complement the above surveys with a more in-depth review which provides (a) deeper explanations of how current approaches work; (b) the strengths and weak-

[2]This definition also covers the robust-optimisation-over-time situation described in [14] where a sequence of $\langle S_1, ...S_k\rangle$ robust solutions is found provided that $k > 1$.

nesses of each approach; (c) the current assumptions and coverage of existing EDO research; and (d) an analysis of current gaps, the challenges and opportunities in EDO based on (a), (b) and (c).

The purpose of this paper is to provide such an in-depth review. It will focus on reviewing four different aspects of EDO research: benchmark problems/generators, performance measures, algorithmic approaches (EAs, swarm intelligence and other meta-heuristic methods), and theoretical developments. Some future research issues and directions regarding EDO will also be presented.

The rest of this paper is organized as follows. The next section reviews the benchmark problems and benchmark problem generators that have been used for EDO in the literature. Section 3 describes the performance measures that are commonly used by researchers in the domain. Section 4 reviews different approaches that have been developed by researchers to address DOPs. The strengths and weaknesses of different approaches are also discussed in Section 4. The theoretical development regarding EDO is presented in Section 5. Finally, Section 6 summarizes the paper and presents some discussions on the future research issues and directions regarding evolutionary optimization in dynamic environments.

2. Benchmark problems

2.1. Properties of a good benchmark problem

The use of benchmark problems is crucial in the process of developing, evaluating, and comparing EDO algorithms. According to [19, 13, 23, 22], a good benchmark problem is one that has the following characteristics:

1. Flexibility: Configurable under different dynamic settings (change severity, frequency, periodicity) and different scales (number of optima, dimensions, domain ranges etc).
2. Simplicity and efficiency: Simple to implement, analyse, or evaluate, and computationally efficient.

In addition, because the ultimate goal of any optimisation algorithm is to be applicable to real-world situations, a good benchmark problem needs to satisfy the following important property:

3. Allow conjectures to real-world problems or resemble real-world problems to some extent [19, 20].

2.2. Reviewing existing general-purpose benchmark generators/problems

In this section, we will review the commonly used general-purpose dynamic optimisation benchmark generators/problems in the literature based on the above criteria. The purpose is to identify the common characteristics of benchmark problems to (a) investigate how academic problems reflect the properties of real-world problems and (b) facilitate researchers in choosing the right test problems. It should be noted that in this section we focus only on simple, aritificial general-purpose benchmark generators/problems. There are a large number of problem-specific dynamic combinatorial benchmark problems, which are created from static combinatorial problems such as the travelling salesmen problems, the scheduling problems and the knapsack problems by adding time-dependent elements to the problem parameters. For a comprehensive list of such problems, readers are refer to [11].

When reviewing existing benchmark generators/problems, we can either categorise problems based on the ways they are generated, or based on the characteristics of the generated problems. In this section we choose the second way of categorisation because (i) it better suits the purpose of identifying the common characteristics of benchmark problems and (ii) it helps users in choosing the suitable benchmark for their applications. In the end, what users look for in selecting a benchmark problem is not how they are generated but what types of dynamics they represent and what characteristics they have.

The characteristics of each general-purpose benchmark generator/problem are identified and the problems are classified into different groups based on the following different criteria:

1. Time-linkage: Whether the future behaviour of the problem depends on the current and/or the previous solutions found by the algorithm or not.
2. Predictability: Whether the generated changes follow a regular pattern (e.g. optima moving in fixed step sizes, landscape rotating in fixed angles, cyclic/periodical changes, and predictable change intervals), and hence are predictable, or not.
3. Visibility: Whether the changes are visible to the optimisation algorithm and if so whether changes can be detected by using just a few detectors (special locations in the search space where the objective or constraint functions are re-evaluated to detect changes)
4. Constrained problem: Whether the problem is constrained or not, and if yes, whether the constraints change over time.
5. Number of objectives: Whether the problem has a single objective or multiple objectives.
6. Type of changes: Detailed explanation of how changes occur.
7. Changes are cyclic/periodical/recurrent or not?
8. Factors that change: Objective functions, domain of variables, number of variables, constraints, or other parameters.

Tables 1 and 2 provides the detailed information of each artificial benchmark problem in the continuous and combinatorial domains, respectively, and their characteristics.

From tables 1 and 2, we can see that the common characteristics of academic benchmark problems are as follows.

- *All of the reviewed general-purpose benchmark generators/problems are non time-linkage problems.* There are a couple of general-purpose benchmark problems with the time-linkage property [24, 16], but they are proposed as a proof of principle rather than a complete set of benchmark problems.

- *Most of the reviewed benchmark generators/problems are unconstrained or domain constrained*, except the two most recent studies [25, 26]

- *In the default settings of most of the review benchmark generators/problems, changes are detectable by using just a few detectors.* Exceptions are some problem instances in [27, 28] where only one or some peaks move, and in [6, 25, 26] where the presences of the visibility mask or constraints make only some parts of the landscapes change. Due to their highly configurable property some benchmark generators can be configured to create scenarios where changes are more difficult to detect.

- *In most cases the factors that change are the objective functions.* Exceptions are the problems in [25, 26] where the constraints also change and one instance in [29] where the dimension also changes. Dimensional changes have also been taken into account in recent combinatorial optimisation research, for example [30].

- *Many generators/problems have unpredictable changes in their default settings*, but due to their flexibility some of the generators/problems can be configured to allow predictable changes, at least in the frequency and periodicity of changes

- *A majority of benchmark generators / problems have periodical/ recurrent changes*

- *Most generators/problems are single-objective* except the problems in [31], [32] and [33]. Recently there are some new dynamic multi-objective problems e.g. [34], but most of them are based on the first two of the papers mentioned above.

The common characteristics of academic benchmark problems above reflect the current main assumptions of the EDO community about the characteristics of DOPs. In Subsection 6.2 we will discuss whether these assumptions fully reflect the properties of real-world dynamic optimisation problems.

Table 1: Common general-purpose benchmark generators/problems in the continuous domain

	General notes	Time-linkage	Changes are predictable?	Changes are detectable by using just a few detectors?	Single/ Multi Obj?	Type of changes	Changes are cyclic/ periodical/ recurrent?	Factors that change				Others notes
								Objective functions	Domain of variables	Number of variables	Constr. functions	
Switching function [27]	The benchmark consists of two landscapes A and B. Changes can occur in three ways: (1) linear translation of peaks in A ; (2) global optimum randomly moves while the rest of landscape A is fixed; (3) switching landscapes between A and B.	No	Mostly no (for changes where peaks are linearly translated, peak movements might be predictable; the re-occurance of the switching landscape can also be predictable)	Yes & No (There are three types of changes. The first and the last can be detected by using a few detectors, while the second type of change is not)	Single-objective	Three types of changes: (1) linear translation of peaks; (2) global optimum randomly moves while the landscape is fixed; (3) switching landscapes.	Yes (scenario (3)), in both fast (2 generations) and slow (20 generations) modes	N/I (no detail of the objective function is given)	No	No	No	Linear translation of all peaks; random movement of global optimum and switching landscape
Moving Peaks [21]	The search landscape consists of a number of randomly generated peaks, each has its width, height and location changed after each change step. The benchmark is highly configurable (dimension, number of peaks and the dynamics of each peak are all configurable)	No	Mostly no in the default settings but some factors can be predictable if specifically configured (e.g. where the parameter lambda=1, the peaks move in the same direction and hence movement direction is predictable)	Yes in the default settings but can be configurable (the benchmark generator can be modifiable to allow changes in only a part of the landscape to make changes more difficult to detect)	Single-objective	Changes in heights, widths and locations of peaks. The widths and heights of peaks are changed by adding a Gaussian variable. The location of peaks are moved by a fixed step and the direction of peaks are based on a combination of the previous direction and a direction parameter.	Configurable	Yes	No	No	No	Each of the peaks has its own time-dependent parameters height, width and location and hence each peak can change differently. E.g. [14] configured the benchmark to make different peaks change with different frequencies and severities.
Oscillating Peaks [21]	The search landscape oscillates among L fixed landscapes.	No	Mostly no (it might be possible to predict the period of oscillation)	Yes	Single-objective	Landscape switching	Yes (due to the oscillation of landscapes)	No	No	No	No	Landscape switching
DF1 [35, 36]	The search landscape consists of a number of peaks (randomly generated or pre-determined), each has its width, height and location changed after each time step. The behaviours of changes are controlled by a logistic function. The benchmark is highly configurable (dimension, number of peaks and the dynamics of each peak are all configurable)	No	No in the five tested instances provided in [36] but some factors can be predictable if specifically configured (e.g. where the motion of peaks is set to be linear, peaks' movement directions can be predictable)	Yes in four tested instances, no in one test instance and configurable (the benchmark generator can be modifiable to allow changes in only a part of the landscape to make changes more difficult)	Single-objective	Changes in heights, widths and locations of peaks. The behaviours of changes are controlled by a logistic function. Depending on the parameter of the logistic function, changes in step sizes can be fixed, bifurcation or chaotic.	No in the five tested instances provided in [36] but can be configurable	Yes	No	No	No	
Gaussian peak [37]	The search landscape consists of a number of peaks (randomly generated), each has its location changed after each time step. Two levels of severity: abrupt and gradual, were tested	No	No (all peaks move randomly)	Yes	Single-objective	Changes in location of peaks. Peaks move in random directions and the step sizes are uniformly distributed over an interval controlled by the level of severity.	No	Yes	No	No	No	

Table 1 Continuous benchmark generators/problems (cont.)

								Factors that change				
	General notes	Time-linkage	Changes are predictable?	Changes are detectable by using just a few detectors?	Single/ Multi Obj?	Type of changes	Changes are cyclic/ periodical/ recurrent?	Objective functions	Domain of variables	Number of variables	Constr. functions	Others notes
Disjoint landscape [28]	The main principle of this benchmark generator is to divide the search space into a number of disjoint sub-spaces, each with a separate unimodal function. The main search space hence is a composition of the local optima from the disjoint sub-spaces.	No	Mostly no but configurable (it is possible to configure the benchmark to make it predictable. In addition, in the tested example peaks' values are artificially move in a circle, making it to some extent possible to predict the movement)	Dependable on the number of peaks that change at each time step (in the tested example, only the values of some peaks change)	Single-objective	Changes in values of peaks	Yes	Yes	No	No	No	
Dynamic rotation [38]	The principle of this benchmark generator is to combine the original search space with a "visibility mask", which allows only certain parts of the search space to have the original fitness values. Other regions, which are hidden by the mask, have constant, pre-defined fitness values. The dynamics are created by rotating the original search space, the mask, or both.	No	Mostly no (because all changes in this generator are created by rotation, to some extent we can consider the rotation movement predictable)	Partly (in case there is no visibility mask, it is possible to detect changes using just one detector. In case there is a visibility mask, the level of difficulty in detecting changes depends on the way the visibility mask is defined)	Single-objective	Rotations of the underlying search space and the visibility masks. The rotation is controlled by an orthogonal matrix.	Yes (due to the rotation)	Yes	No	No	No	Changes in visibility masks
MOO-based dynamic problem generator [31]	The principle of this benchmark generator is to use the aggregating objectives approach to create a n-objective dynamic function from n+1 static single-objective functions through a dynamic weight. The dynamic weight governs how the dynamic problem changes. The benchmark is highly configurable	No	Yes and Configurable (it is possible to configure the benchmark generator to create predictable changes. For example, in the tested instance the optimum movement is configured to be linear, and hence could be predictable)	Yes	Both single and multiple-objectives are configurable	The global optimum (or the Pareto front in the multiple-objective case) can be configured to move linearly, non-linearly or to follow specific moving rules. The height of the peak also changes accordingly.	Not in the tested instance but configurable	Yes	No	No	No	The dynamic parameter of the main objective function is the aggregate weight, which controls how the problem changes

Table 1 Continuous benchmark generators/problems (cont.)

	General notes	Time-linkage	Changes are predictable?	Changes are detectable by using just a few detectors?	Single/ Multi Obj?	Type of changes	Changes are cyclic/ periodical/ recurrent?	Factors that change				Others notes
								Objective functions	Domain of variables	Number of variables	Constr. functions	
FDA dynamic multi-objective benchmark set [32] (extended to ZJZ in [34]) and two HE problems [33]	The principle of this benchmark generator is to combine the static multiple-objective functions with the time-dependent parameters: F(t) to control the dynamics of the density of Pareto solutions, H(t) to control the dynamic of the shape of the Pareto front, and G(t) to control the dynamic shape of the Pareto optimal set.	No	Configurable (it is possible to configure the benchmark generator to create predictable changes. It might also be possible to predict the period of oscillation)	Yes	Mulitple-objective	The density of Pareto solutions, the shape of the Pareto front and the shape of the Pareto set change over time	Yes	Yes	No	No	No	Changes in the objective functions are controlled by three time-dependent parameters: F(t), G(t) and H(t). FDA1 was extended in [34] to add nonlinear linkages between variables and to make PF dynamic. In HE problems [33] the Pareto front is discontinuous.
Dynamic test functions [39]	This benchmark set follows a landscape-oriented approach where the dynamic test problems are specifically designed to represent different changes in landscape structure, in optima's positions, in optima's values etc. Changes can be linear or periodical.	No	Partly (the changes in some problems in the benchmark set follow predictable rules like moving linearly or occurring periodically)	Yes for most tested instances (exceptions are in problems like the OPoL where only the position of the global optimum changes)	Single-objective	Different types of changes are generated in different functions, e.g. changes in landscape structure, in optima's positions, in optima's values etc. Changes can be linearly or periodically.	Yes	Yes	No	No	No	
CDOPG (XOR-extension for continuous domain) [40]	The principle in this generator is to use an orthogonal transformation matrix to periodically rotate a static landscape to create dynamic instances (in a similar way to the XOR benchmark in the combinatorial domain). The properties of the fitness landscape is preserved after each change	No	No (but the periodicity of rotations can be predictable)	Yes	Single-objective	The fitness landscape is rotated. The magnitude of change is defined by the rotation angle.	Yes (due to the rotation)	Yes	No	No	No	Changes (rotations) are made on the decision variables. Specifically, before being evaluated each individual vector is moved (rotated) to a different position in the fitness landscape using an orthogonal matrix.

Table 1 Continuous benchmark generators/problems (cont.)

								Factors that change				
	General notes	Time-linkage	Changes are predictable?	Changes are detectable by using just a few detectors?	Single/ Multi Obj?	Type of changes	Changes are cyclic/ periodical/ recurrent?	Objective functions	Domain of variables	Number of variables	Constr. functions	Others notes
CEC09 GDBG [29]	This set of benchmark generators is a combination of existing ideas about landscape shifting [41], landscape rotation [42, 38, 41, 40], and using dynamic rules to control change steps [36]	No	No (but the periodicity of rotations can be predictable)	Yes	Single-objective	The fitness landscape is rotated and shifted. The magnitude of change is defined by the rotation angle.	Yes (due to the rotation)	Yes	No	Yes (for each proposed problem, there is one instance with changing dimension)	No	The landscape is rotated and the heights/widths of peaks are also changed. Rotation are made on decision variables. The magnitude of changes (angle of rotation) is determined by dynamic rules (small/ large/ chaotic/ random/ recurrent/ noisy).
G24 dynamic constrained benchmark set [25]	The principle of this benchmark generator is to make existing benchmark problems dynamic by replacing their static parameters with time-dependent parameters. The benchmark supports dynamics in the constraint functions and the problems are organised in pairs, of which each pair has two almost identical problem, one with a special property and one with not.	No	Yes (changes follow predictable rules like linear movements and periodical movements)	No (there are situations when only a part of the landscape changes due to dynamic constraints. In such case it might not be easy to detect changes using a few detectors)	Single-objective	Combinations of changes in objective functions, changes in constraints and changes in both. Changes are linear and cyclic.	Yes	Yes	No	No	Yes	Changes (linear and cyclic) are made on the parameters of the objective functions and constraint functions
A dynamic constrained benchmark problem [26]	The principle of this benchmark generator is to combine existing ”field of cones on a zero plane” with dynamic norm-based constraints (with square/diamond/sphere-like shapes)	No	No in the proposed settings	No (there are situations when only a part of the landscape changes due to dynamic constraints. The author also proposed an unconstrained version [43] where the level of detectability is adjustable)	Single-objective	Locations of peaks and constraints are changed following a a Gaussian variable with fixed mean and variance	Partly (changes are recurrent because they are generated using a Gaussian variable with fixed mean and variance)	Yes	No	No	Yes	

Table 2: Common general-purpose benchmark generators/problems in the combinatorial domain

								Factors that change				
	General notes	Time-linkage	Changes are predictable?	Changes are detectable by using just a few detectors?	Single/ Multi Obj?	Type of changes	Changes are cyclic/ periodical/ recurrent?	Objective functions	Domain of variables	Number of variables	Constr functions	Others notes
Dynamic Match Fitness [44]	This benchmark generator is based on the static bit-matching function (find a solution that matches a given string). The dynamics elements are introduced by changing the match-string.	No	Not in the default settings but configurable to be cyclic and hence its periodicity can be predictable	Dependable on particular changes (number of bits changed and the location of changing bits)	Single-objective	Changes are introduced by changing a number of bits in the match-string	No (not considered in the tested instances but configurable)	Yes	No	No	No	
XOR [45, 46]	This benchmark generator can be combined with any static binary-coded problems to generate dynamic problems. Dynamic problems are generated by XOR-ing each individual with a special binary mask, which determines the magnitude of changes (in term of Hamming distances). Dynamic landscapes generated by the XOR operator have a special property that the landscape structure (and hence the distances among individuals and their fitness values) is preserved after each change.	No	Not in the default settings but configurable to be cyclic and hence its periodicity can be predictable	Dependable on particular changes and on the underlying landscape	Single-objective	Changes are introduced by changing a number of bits in the binary mask, which will later be used to transform the position of individuals in the population. The severity level of changes is represented by the Hamming distance between the old and new binary mask.	Yes (the original version does not support cyclic changes, but an extended version was proposed in [47, 48]	Yes	No	No	No	Changes are made on the vector of decision variables. Specifically, before being evaluated each individual vector is moved to a different position in the fitness landscape using the XOR operator and the binary mask. In other words, instead of moving the optimum, using the XOR operator the search population is moved after each change.
Dynamic DTF [23]	This benchmark generator is based on the static Unitation and Trap functions. The static functions are made dynamic by making theirs static parameters time-dependent and by changing the scales of function values. The benchmark generator is highly configurable	No	No in the default settings but configurable to be cyclic and hence its periodicity can be predictable	No (there is no guarantee that using a few detectors can detect changes because due to the nature of the dynamic trap function, only a part of the search landscape changes)	Single-objective	Changes in optima height, size of basin and both optima height and basin size. Other advanced dynamic environments can also be constructed	Not considered in the tested instances but configurable	Yes	No	No	No	

3. Performance measures

Properly measuring the performance of algorithms is vital in EDO. In this section we will (i) review existing studies to identify the most common criteria used to evaluate EDO algorithms, (ii) analyse the strengths and weaknesses of each measure, and (iii) discuss the possibility to reduce the disadvantages (if there are any) of current performance measures. Performance measures in EDO can be classified into two main groups: optimality-based and behaviour-based. There is also a sub group of measures for dynamic multi-objective optimisation. The subsections below will discuss each groups of measures in details.

3.1. Optimality-based performance measures

Optimality-based performance measures are measures that evaluate the ability of algorithms in finding the solutions with the best objective/fitness values (fitness-based measures) or finding the solutions that are closest to the global optimum (distance-based measures). This type of measures is by far the most common in EDO. The measures can be categorised into groups as follow:

Best-of-generation. This measure is calculated as the best value at each generation, averaged over several runs. It is usually used in two ways: First, the best value in each generation is plotted against time to create a performance curve. This measure has been used since the early research in [49, 50, 8, 51, 37] and is still one of the most commonly used measures in the literature. The advantage of such performance curves is that they can show a more holistic picture of how the tested algorithm has performed. However, because the performance curve is not scalar, it is difficult to compare the final outcome of different algorithms and to see whether the difference between two algorithms is statistically significant [36].

To improve the above disadvantage, a variation of the measure is proposed where the best-of-generation values are averaged over all generations [46]. The measure is described below:

$$\overline{F}_{BOG} = \frac{1}{G} \times \sum\nolimits_{i=1}^{i=G} \left(\frac{1}{N} \times \sum\nolimits_{j=1}^{j=N} F_{BOG_{ij}} \right) \tag{2}$$

where $\overline{F}_{BOG}$ is the mean best-of-generation fitness, G is the number of generations, N is the total number of runs, and $F_{BOG_{ij}}$ is the best-of-generation fitness of generation i of run j of an algorithm on a particular problem. An identical measure has independently been proposed by Morrison [36] under the name *collective mean fitness* (F_C).

$\overline{F}_{BOG}$ is one of the most commonly used measures. The advantage of this measure, as mentioned above, is to enable algorithm designers to quantitatively compare the performance of algorithms. The disadvantage of the measure and its variants is that they are not normalised, hence can be biased by the difference of the fitness landscapes at different periods of change. For example, if at a certain period of change the overall fitness values of the landscape is particularly higher than those at other periods of changes, or if an algorithm is able to get particular high fitness value at a certain period of change, the final $\overline{F}_{BOG}$ or F_C might be biased toward the high fitness values in this particular period and hence might not correctly reflect the overall performance of the algorithm. Similarly, if $\overline{F}_{BOG}$ is used averagely to evaluate the performance of algorithms in solving a group of problems, it is also biased toward problems with larger fitness values.

Modified offline error and offline performance. Proposed in [19] and [52], the *modified offline error* is measured as the average over, at every evaluations, the error of the best solution found since the last change of the environment. This measure is always greater than or equal to zero and would be zero for a perfect performance.

$$E_{MO} = \frac{1}{n} \sum\nolimits_{j=1}^{n} e_{MO}(j) \tag{3}$$

where n is the number of generations so far, and $e_{MO}(j)$ is the best error since the last change gained by the algorithm at the generation j.

A similar measure, the *modified offline performance*, is also proposed in the same reference to evaluate algorithm performance in case the exact values of the global optima are not known

$$P_{MO} = \frac{1}{n} \sum\nolimits_{j=1}^{n} F_{MO}(j) \tag{4}$$

where n is the number of generations so far, and $F_{MO}(j)$ is the best performance since the last change gained by the algorithm at the generation j.

E_{MO} is one of the most commonly used measures in EDO. With this type of measures, the faster the algorithm to find a good solution, the higher the score. The E_{MO} is closely related to $\overline{F}_{BOG}$. The only major difference between the two measures is that E_{MO} looks at each evaluation while $\overline{F}_{BOG}$ looks at only the best per generation. Similar to the $\overline{F}_{BOG}$, the offline error/performance are also useful in evaluating the overall performance of an algorithm and to compare the final outcomes of different algorithms. These measures however have some disadvantages. First, they require that the time a change occurs is known. Second, similar to $\overline{F}_{BOG}$, these measures are also not normalised and hence can be biased under certain circumstances.

In [15][sect. 5.3.2], the offline error/performance was modified to measure the performance of algorithms in dynamic constrained environments. Specifically, when calculating Eq. 3 for dynamic constrained problems, the authors only consider the best errors/fitness values of *feasible* solutions at each generation. If in any generation there is no feasible solution, the measure will take the worst possible value that a feasible solution can have for that particular generation.

Best-error-before-change. Proposed in [28] [3], this measure is calculated as the average of the smallest errors (the difference between the optimum value and the value of the best individual) achieved at the end of each change period (right before the moment of change).

$$E_B = \frac{1}{m}\sum\nolimits_{i=1}^{m} e_B\left(i\right) \tag{5}$$

where $e_B\left(i\right)$ is the best error just before the ith change happens; m is the number of changes.

This measure is useful in situations where we are interested in the final solution that the algorithm achieved before the change. The measure also makes it possible to compare the final outcome of different algorithms. However, the measure also has three important disadvantages. First, it does not say anything about how the algorithms have done to achieve the current performance. As a result, the measure is not suitable if what users are interested in is the overall performance or behaviours of the algorithms. Second, similar to the best-of-generation measure, this measure is also not normalised and hence can be biased toward periods where the errors are relatively very large. Third, the measure requires that the global optimum value at each change is known.

This measures is adapted as the basis for one of the complementary performance measures in the CEC'09 competition on dynamic optimisation [29].

Optimisation accuracy. The *optimisation accuracy* measure (also known as the *relative error*) was initially proposed in [53] and was adopted in [54] for the dynamic case:

$$accuracy_{F,EA}^{(t)} = \frac{F(best_{EA}^{(t)}) - Min_F^{(t)}}{Max_F^{(t)} - Min_F^{(t)}} \tag{6}$$

where $best_{EA}^{(t)}$ is the best solution in the population at time t, $Max_F^{(t)} \in \mathbb{M}$ is the best fitness value of the search space and $Min_F^{(t)} \in \mathbb{M}$ is the worst fitness value of the search space. The range of the accuracy measure ranges from 0 to 1, with a value of 1 and 0 represents the best and worst possible values, respectively.

[3]named *Accuracy* by the authors

The optimisation accuracy have the same advantages as the $\overline{F}_{BOG}$ and E_{MO} in providing quantitative value and in evaluating the overall performance of algorithms. The measure has an advantage over $\overline{F}_{BOG}$ and E_{MO}: it is independent to fitness rescalings and hence become less biased to those change periods where the difference in fitness becomes particularly large. The measure, however, has a disadvantage: it requires information about the absolute best and worst fitness values in the search space, which might not always be available in practical situations. In addition, as pointed by the author himself [54], the optimisation accuracy measure is only well-defined if the complete search space is not a plateau at any generation t, because otherwise the denominator of Eq. 6 at t would be equal to zero.

Normalised scores. When trying to compare algorithms across a number of different change periods, or a number of problem instances, or even different problem domains, there is the challenge of combining quality measures. One possibility is to use rank-based (non-parametric) statistical tests for comparison. Another option is to normalize the values.

[15] proposes such a normalization even across the different change periods of a dynamic problem. The idea is that, given a group of n tested algorithms and m test instances (which could be m different test problems or m change periods of a problem), for each instance j the performance of each algorithm is normalised to the range $(0, 1)$ so that the best algorithm in this instance j will have the score of 1 and the worst algorithm will get the score of 0. The final overall score of each algorithm will be calculated as the average of the normalised scores from each individual instance. According to this calculation, if an algorithm is able to perform best in all tested instances, it will get an overall score of 1. Similarly, if an algorithm performs worst in all tested instances, it will get an overall score of 0.

A formal description of the *normalised score* of the ith algorithm is given in Equation 7:

$$S_{norm}(i) = \frac{1}{m}\sum\nolimits_{j=1}^{m} \frac{|e_{\max}(j) - e(i,j)|}{|e_{\max}(j) - e_{\min}(j)|},\ \forall i = 1:n. \tag{7}$$

where $e(i,j)$ is the modified offline error of algorithm i in test instance j; and $e_{\max}(j)$ and $e_{\min}(j)$ are the largest and smallest errors among all algorithms in solving instance j. In case the offline errors of the algorithms are not known (because global optima are not know), we can replace them by the offline performance to get exactly the same score. The normalised score S_{norm} can also be calculated based on the best-of-generation values.

The normalised score has two major advantages. First, it looks at relative rather than absolute performance. Second, it does not need the knowledge of the global optima or the absolute best and worst fitness values of a problem.

The normalised score, however, also has its own disadvantages: First, S_{norm} is only feasible in case an algorithm is compared to other peer algorithms because the scores are calculated based on the performance of peer algorithms. Second, S_{norm} only shows the relative performance of an algorithm in comparison with other peer algorithms in the corresponding experiment. It cannot be used solely as an absolute score to compare algorithm performance from different experiments. For this purpose, we need to gather the offline errors/offline performance/best-of-generation of the algorithms first, then calculate the normalised score S_{norm} for these values. For example, assume that we have calculated S^A_{norm} for all algorithms in group A, and S^B_{norm} for all algorithms in group B in a separated experiment. If we need to compare the performance of algorithms in group A with algorithms in group B, we cannot compare the S^A_{norm} against S^B_{norm} directly. Instead, we need to gather the $E_{MO}/P_{MO}/F_{BOG}$ of all algorithms from the two groups first, then based on these errors we calculate the normalised scores S^{AB}_{norm} of all algorithms in the two groups.

Non-fitness distance-based measures. Although most of the optimality-based measures are fitness-based, some performance measures do rely on the distances from the current solutions to the global optimum to evaluate algorithm performance. In [55], a performance measure, which is calculated as the minimum distance from the individuals in the population to the global optimum, was proposed. In [56], another distance-based measure was introduced. This measure is calculated as the distance from the mass centre of the population to the global optimum.

The advantage of distance-based measures is that they are independent to fitness rescalings and hence are less affected by possible biases caused by the difference in fitness of the landscapes in different change periods. The disadvantages of these measures are that they require knowledge about the exact position of the global optimum, which is not always available in practical situation. In addition, compared to some other measures this type of measures might not always correctly approximate the exact adaptation characteristics of the algorithm under evaluated, as shown in an analysis in [54].

3.2. Behaviour-based performance measures

Behaviour-based performance measures are those that evaluate whether EDO algorithms exhibit certain behaviours that are believed to be useful in dynamic environments. Example of such behaviours are maintaining high diversity through out the run; quickly recovering from a drop in performance when a change happens, and limiting the fitness drops when changes happen. These measures are usually used complementarily with optimality-based measures to study the behaviour of algorithms. They can be categorised into the following groups:

Diversity. Diversity-based measures, as their name imply, are used to evaluate the ability of algorithms in maintaining diversity to deal with environmental dynamics. There are many diversity-based measures, e.g. *entropy* [57], *Hamming distance* [58, 59, 60], *moment-of-inertia* [61], *peak cover* [19], and *maximum spread* [62] of which Hamming distance-based measures are the most common.

Hamming distance-based measures for diversity have been widely used in static evolutionary optimisation and one of the first EDO research to use this measure for dynamic environments is the study of [58] where the *all possible pair-wise Hamming distance* among all individuals of the population was used as the diversity measure. In [59] the measure was modified so that only the Hamming distances among the best individuals are taken into account.

A different and interesting diversity measure is the *moment-of-inertia* [61], which is inspired from the fact that the moment of inertia of a physical, rotating object can be used to measure how far the mass of the object is distributed from the centroid. Morrison and De Jong [61] applied this idea to measuring the diversity of an EA population. Given a population of P individuals in N-dimensional space, the coordinates $C = (c_1, ..., c_N)$ of the centroid of the population can be computed as follows:

$$c_i = \frac{\sum_{j=1}^{P} x_{ij}}{P}$$

where x_{ij} is the ith coordinate of the jth individual and c_i is the ith coordinate of the centroid.

Given the computed centroid above, the moment-of-inertia of the population is calculated as follows:

$$I = \sum_{i=1}^{N} \sum_{j=1}^{P} (x_{ij} - c_i)^2$$

In [61], the authors proved that the moment-of-inertia measure is equal to the pair-wise Hamming distance measure in the binary space. The moment-of-inertia, however, has an advantage over the Hamming distance measure: it is more computationally efficient. The complexity of computing the moment-of-inertia is only linear with the population size P while the complexity of the pair-wise diversity computation is quadratic.

Another interesting, but less common diversity measure is the *peak cover* [19], which counts the number of peaks covered by the algorithms over all peaks. This measure requires full information about the peaks in the landscape and hence is only suitable in academic environments.

In dynamic constrained environments, a diversity-related measure was also proposed [15][Sect 5.3.2], which counts the percentage of solutions that are infeasible among the solutions selected in each generation. The average score of this measure (over all tested generations) is then compared with the percentage of infeasible areas over the total search area of the landscape. If the considered algorithm is able to treat infeasible diversified individuals and feasible diversified individuals on an equal basis (and hence to maintain diversity effectively), the two percentage values should be equal.

Drops in performance after changes. Some EDO studies also develop measures to evaluate the ability of algorithms in restricting the drop of fitness when a change occurs. Of which, the most representative measures are the measures *stability* [54], *satisficability* and *robustness* [59].

The measure *stability* is evaluated by calculating the difference in the fitness-based *accuracy* measure (see Eq. 6) of the considered algorithm between each two time steps

$$stab_{F,EA}^{(t)} = max\{0, accuracy_{F,EA}^{(t-1)} - accuracy_{F,EA}^{(t)}\} \tag{8}$$

where $accuracy_{F,EA}^{(t)}$ has already been defined in Eq. 6.

The *robustness* measure is similar to the measure *stability* in that it also determines how much the fitness of the next generation of the EA can drop, given the current generation's fitness. The measure is calculated as the ratio of the fitness values of the best solutions (or the average fitness of the population) between each two consecutive generations.

The *satisficability* measure focuses on a slightly different aspect. It determines how well the system is in maintaining a certain level of fitness and not dropping below a pre-set threshold. The measure is calculated by counting how many times the algorithm is able to exceed a given threshold in fitness value.

Convergence speed after changes. Convergence speed after changes, or the ability of the algorithm to recover quickly after a change, is also an aspect that attracts the attention of various studies in EDO. In fact many of the optimality-based measures, such as the offline error/performance, best-of-generation, relative-ratio-of-best-value discussed previously can be used to indirectly evaluate the convergence speed. In addition, in [54], the author also proposed a measure dedicated to evaluating the ability of an adaptive algorithm to react quickly to changes. The measure is named *reactivity* and is defined as follows:

$$\begin{aligned} react_{F,A,\epsilon}^{(t)} = \; & min\left\{ t' - t | t < t' \leq maxgen, t' \in \mathbb{N}, \frac{accuracy_{F,A}^{(t')}}{accuracy_{F,A}^{(t)}} \geq (1-\epsilon) \right\} \\ & \cup \{maxgen - t\} \end{aligned} \tag{9}$$

where *maxgen* is the number of generations. The *reactivity* measure has a disadvantage: it is only meaningful if there is actually a drop in performance when a change occurs. Otherwise, nothing can be said about how well the algorithm reacts to changes. In situations like the dynamic constrained benchmark problems in [25] where the total fitness level of the search space may increase after a change, the measure *reactivity* cannot be used. In addition, the measure is undefined in any period where $accuracy_{F,A}^{(t)}$ is zero.

To provide more insights on the convergence behaviour of algorithms, recently a new measure, the *absolute recovery rate* (ARR) was proposed [63].

The ARR measure is used to analyse *how quick it is for an algorithm to start converging on the global optimum before the next change occurs*:

$$ARR = \frac{1}{m}\sum\nolimits_{i=1}^{m} \frac{\sum_{j=1}^{p(i)}\left[f_{best}\left(i,j\right) - f_{best}\left(i,1\right)\right]}{p\left(i\right)\left[f^{*}\left(i\right) - f_{best}\left(i,1\right)\right]} \tag{10}$$

where $f_{best}\left(i,j\right)$ is the fitness value of the best solution since the last change found by the tested algorithm until the jth generation of the change period i , m is the number of changes and $p\left(i\right), i = 1 : m$ is the number of generations at each change period i and $f^{*}\left(i\right)$ is the global optimal value of the landscape at the ith change. The ARR score would be equal to 1 in the best case when the algorithm is able to recover and converge on the global optimum immediately after a change, and would be equal to zero in case the algorithm is unable to recover from the change at all. Note that in order to use the measure ARR we need to know the global optimum value at each change period.

Fitness degradation over time. A recent experimental observation [64] showed that in DOPs the performance of an algorithm might degrade over time due to the fact that the algorithm fails to follow the optima after some changes have occurred. To measure this degradation, in [64] a measure named $\beta-degradation$ was proposed. The measure is calculated by firstly using linear regression (over the accuracy values achieved at each change period) to create a regression line, then evaluate the measure as the slope of the regression line. A positive $\beta-degradation$ value might indicate that the algorithm is able to keep track with the moving optima. The measure however does not indicate whether the degradation in performance is really caused by the long-term impact of DOP, or simply by an increase in the difficulty level of the problem after a change. In addition, a positive $\beta-degradation$ value might also not always be an indication that the algorithm is able to keep track with the moving optima. In problems where the total fitness level increases, like in the dynamic constrained benchmark problems in [25, 63] mentioned above, a positive $\beta-degradation$ can be achieved even when the algorithm stays at the same place.

3.3. Performance measures for dynamic multi-objective optimisation

In the case of dynamic multiobjective problems, researchers measured performance similar to the single-objective community, except that in a first step the multiple objectives are reduced to a single set performance criterion which has been commonly used in static multiobjective optimisation (e.g., hypervolume, maximum spread, inverse generational distance). Then this measure is calculated after each certain period, mostly just before the change occurs, so that eventually the values can be aggregated over time, e.g. along the idea of offline performance [65, 62, 66, 67].

Similar to the accuracy in single-objective optimisation (Eq. 6), there was also an *accuracy* measure for multi-objective optimisation [68], which was defined as the ratio between the current hypervolume and the maximum hypervolume achieved so far (maximisation case), or the ratio between the minimum hypervolume so far and the current hypervolume (minimisation case). Based on this accuracy for MOO, a stability and a reactivity measure values can be calculated the same way as in the single-objective case using the equations 8 and 9.

3.4. Discussion

There are some open questions about performance measures in EDO. First, it is not clear if optimality is the only goal of real-world DOPs and if existing performance measures really reflect what practitioners would expect from optimisation algorithms. So far, only a few studies, e.g., [59, 14, 15], tried to justify the meaning of the measures by suggesting some possible real-world examples where the measures are applicable. It would be interesting to find the answer for the question of what are the main goals of real-world DOPs, how existing performance measures reflect these goals and from that investigate if it is possible to make the performance measures more specific (if needed) to suit practical requirements. In [15, chapter 3], a first attempt has been be made to find out more about the main optimisation goals of real-world DOPs and the link between existing performance measures and the goals of real-world applications.

Second, most optimality-based measures are based on absolute fitness values, while relative performance values may also be interesting when comparing different algorithms. The *accuracy* measure [54] is among the few studies that tried to normalise fitness values at each change period using a window of the maximum and minimum possible values. This requires full knowledge of the maximum and minimum possible values at each change period, which might not be available in practical situations, while the normalised score proposed in [15] does not require this problem-specific knowledge.

Third, although the behaviour-based measures are usually used complementary with the optimality-based measures, it is not clear if the earlier really correlate with the latter. Recent studies[64] have shown that the behaviour-based measure *stability* does not directly relate to the quality of solutions and the results of the behaviour-based measure *reactivity* are “usually insignificant” [69, 64]. It would be interesting to systematically study the relationship between behaviour-based measures and optimality-based measures, and more importantly the relationship between the quality of solutions and the assumptions of the community about the expected behaviours of dynamic optimization algorithms.

4. Optimisation approaches

4.1. The goals of dynamic evolutionary algorithms

In stationary optimisation usually the goal is to find the global optimum as quickly as possible. When the considered problem is time-varying, the goal becomes to track the changing optimum. The general assumption is that the problem after a change is somehow related to the problem before the change, and thus an optimisation algorithm needs to learn from its previous search experience as much as possible to hopefully advance the search more effectively.

The following sections will briefly review typical approaches in EDO that have been proposed to satisfy the goals above. We will discuss the strengths and weaknesses of the approaches and their suitability for different types of problems.

Many of the approaches explicitly react to a change. If the occurrance of a change is not explicit, it has to be detected. Thus we start with a section on change detection.

4.2. Detecting a change

Many EDO approaches take explicit action to respond to a change in the environment. This either assumes that changes in the environment are made known to the algorithm, or that the algorithm has to detect the change. This section therefore discusses change detection mechanisms, categorized into (a) detecting change by re-evaluating dedicated detectors, and (b) detecting change based on algorithm behaviour.

4.2.1. Detecting change by re-evaluating solutions

Detecting changes by re-evaluating solutions is by far the most common change-detection approach. The algorithm regularly re-evaluates some specific solutions (detectors) to detect changes in their function values and/or feasibility. Detectors can be a part of the population, such as the current best solutions [70] [71] [72] [15], a memory-based sub-population [21] [73], or a feasible sub-population [15] [74].

Detectors can also be maintained separately from the search population. In this case they can be just a fixed point [75], one or a set of random solutions

[76, 77, 43], a regular grid of solutions / set of specifically distributed of solutions [13], or a list of found peaks [78, 79].

Because using detectors involves additional function evaluations, it might be necessary to identify an optimal number of detectors to maximise algorithm performance. Most existing methods just use one or a small number of detectors to avoid being affected by additional evaluation cost. However, in situations where only some parts of the search space change, e.g. in [25, 43, 63] and in a list of real-world problems cited in [15], using only a small number of detectors might not guarantee that changes are detected [43]. A number of recent attempts has been made to overcome this drawback. In [13], [43],[15], different methods were considered to study the optimal number of detectors depending on the size and complexity of the solved problem. A theoretical analysis in [13] showed that problem dimensionality is a prominent factor in the success of change detection. This finding was later confirmed in the experiments in [43].

The clear advantage of re-evaluting dedicated detectors is that it allows "robust 100% detection" if a high enough number of detectors is used [43]. Ricther [43] also showed that the more difficult change detection is, the more favourable is the approach of re-evaluating dedicated detectors, which was called "sensors" by the author.

Re-evaluating dedicated detectors also have some disadvantages. First, there is the additional cost due to that detectors have to be re-evaluated at every generation. Second, this approach might not be accurate when used in problems with noisy fitness function because noises may mislead the algorithm to thinking that a change has occured [80].

4.2.2. Detecting changes based on algorithm behaviour

In [49] and many following studies that use the same idea, changes are detected based on monitoring the drop in value of the average of best found solutions over a number of generations. In a swarm-based study [80] where the swarm was divided into a tree-based hierarchy of sub-swarms, environmental change was detected based on observation of changes in the hierarchy itself. In [13], the possibility of detecting changes based on diversity, and the relationship between the diversity of fitness values and the success rate of change detection, were studied. In [43], changes were detected based on statistical hypothesis tests to find the difference between distribution of the populations from two consecutive generations. This technique has been commonly used in environmental change detection in the real-world applications of biomedicine, data mining and image processing, as can be seen in the references cited in [43].

Methods that detect changes based on algorithm behaviours have the advantage of not requiring any additional function evaluations. However, because no dedicated detector is used, there is no guarantee that changes are detected [43]. In addition, this approach may cause false positives and hence cause the algorithm to react unnecessarily when no change occurs. Evidence of false positives was found in [80], [43] and [63]. For example, in [63, subsection IV-C5] it was shown that the change detection method based on monitoring the drop in value of best found solutions can give false positive indications in non-elitism genetic algorithms on constrained dynamic problems. Another possible disadvantage is that some change detection methods following this approach might be algorithm-specific, such as the method of monitoring swarm hierarchy in [80].

4.3. Introducing diversity when changes occur

4.3.1. Overview

In stationary optimisation, the convergence of an evolutionary algorithm is required so that the algorithm can focus on finding the best solution in the promising area that it has already found. In dynamic optimisation, however, convergence may be detrimental. This is because if the dynamic landscape changes in one area and there is no member of the algorithm in this area, the algorithm will not be able to react to the change effectively and hence might fail to track the moving global optimum.

Intuitively one simple solution for this drawback is to increase the diversity of an EA after a change has been detected. This solution is described in the pseudo-code of Algorithm 1.

Algorithm 1 Introducing diversity after detecting a change

```
1. Initialise:: Initialise the population
2. For each generation
   (a) Evaluate: Evaluate each member of the population
   (b) Check for changes: Detect changes by monitoring possible signs of
       changes, e.g. a reduction in the best fitness values, or re-evaluation
       of old solutions
   (c) Increase diversity: If change occurs, increase population's diversity by
       changing the mutations (sizes or rates) or relocating individuals
   (d) Reproduce: Reproduce a new population using the adjusted muta-
       tion/learning/adaptation rate
   (e) Return to step 2a
```

Pioneer studies following this solution are hyper-mutation [49] and variable local search (VLS) [81, 82]. They differ mostly in step 2c (Algorithm 1) where different strategies are used to introduce diversity to the population. In his research, Cob [49] proposed an adaptive mutation operator called hyper-mutation whose mutation rate is a multiplication of the normal mutation rate and a hyper-mutation factor. The hyper-mutation is invoked only after a change is detected. In the VLS algorithm, the mutation size is controlled by a variable local search range. This range is determined by the formula $(2^{BITS} - 1)$ where BITS is a value adjustable during the search [83] or adapted using a learning strategy borrowed from the feature partitioning algorithm by Vavak *et al.* [81].

In [15], hyper-mutation was used in an EA to solve dynamic constraint problems. Detectors are placed near the boundary of feasible regions and when the feasibility of these detectors changes, the EA increases its mutation rate to raise the diversity level to track the moving feasible regions. The mutation rate is decreased again once the moving feasible region has been tracked successfully.

Riekert and Malan [84] proposed adaptive Genetic Programming which not only increased mutation, but also reduces elitism and increases crossover probability after a change.

The idea of introducing diversity after a change has also been used in dynamic multi-objective optimisation (DMO). For example, in a multi-population algorithm for DMO [62], when a change is detected, random individuals and some competitor individuals from other sub-populations are introduced to each sub-population to increase diversity.

The approach of introducing diversity after changes is also used in Particle Swarm Optimisation (PSO). Hu and Eberhart [70] introduced a simple mechanism in which a part of the swarm or the whole swarm will be re-diversified using randomization after a change is detected. Janson and Middendorf [80] followed a more sophisticated mechanism where additional to partial re-diversification, after each change the swarm is divided into several sub-swarms for a certain number of generations. The purpose of this is to prevent the swarm from converging to the old position of the global optimum too quickly. Daneshyari and Yen [85] proposed a cultural-based PSO where after a change, the swarms are re-diversified using a framework of knowledge inspired from the belief space in Cultural Algorithms.

Recently, Woldesenbet and Yen [10] proposed a new adaptive method named "relocation variable". In this method, after a change individuals are relocated (mutated) to a position within a specific radius, which is estimated based on the history of the performance of the individual. The more sensitive the individual is to changes, the larger the radius.

The diversity-introducing approach is still commonly used in many recent EDO algorithms, e.g., [78, 86, 43, 26, 87, 88].

4.3.2. Strengths and weaknesses

Because methods following this approach do not need to waste their efforts on maintaining diversity all the time, they have a clear advantage of fully focusing on the search process and only react to changes once they are detected. In addition, methods like hypermutation appear to be good in solving problems with highly frequent changes where changes are small and medium. This is because invoking mutations or distributing individuals around an optimum resembles a type of “local search”, which is useful to observe the nearby places of this optimum. Thus, if the optimum moves to nearby places, it might be tracked [83, 81].

However, this approach has some drawbacks that might make it not so suitable for certain type of problems. They are listed bellow:

- *Dependence on whether changes are known / easy to detect or not*: To increase diversity after a change assumes that the occurrence of a change is know or can be easily detected.
- *Difficulty in identifying the correct mutation size (in case of Hyper-mutation and VLS) or the number of sub-swarms (in case of Hierarchy PSO)*: too small steps will resemble local search while too large steps will result in random search [12].
- *Little information retained from previous search:* Basically, once the algorithm has converged, the only information carried over from one stage of the problem to the next stage after a change is the location of the previous optimum. Intuitively, a lot more information could be relevant.

4.4. Maintaining diversity during the search

4.4.1. Overview

Another approach is to maintain population diversity throughout the search process to avoid the possibility that the whole population converges into one place, hence unable to either track the moving optimum or detect a new competing peak (see Algorithm 2).

Algorithm 2 Maintaining diversity

```
1. Initialise:: Initialise the population
2. For each generation
    (a) Evaluate:: Evaluate each member of the population
    (b) Maintain diversity: Add a number of new, diversified individuals to the
        current population, or select more diversified individuals, or explicitly
        relocate individuals to keep them away from one another.
    (c) Reproduce: Reproduce a new population
    (d) Return to step 2a
```

Methods following this approach do not detect changes explicitly. Instead they rely on their diversity to adaptively cope with the changes. Typical examples of this approach are Random Immigrants [50], fitness sharing [89], Thermo-Dynamical GA[90], Sentinel Placement [13], Population-Based Incremental Learning [91], several Particle Swarm Optimisation (PSO) variants [92, 93, 94, 95], and dynamic Evolutionary Multiobjective optimisation [96, 97, 98].

In the Random Immigrants method, in every generation a number of generated random individuals are added to the population to maintain diversity. Experimental results show that the method is more effective in handling dynamics than the regular EA [50]. It is reported that the high diversity level brought by random immigrants also helps in handling constraints. In [15], it was shown that when combined with the constraint-handling repair method, random-immigrant significantly improve the performance of the tested EA.

Morrison [13] followed a slightly different mechanism in which instead of generating random individuals, his Sentinel Placement method initialises a number of sentinels which are specifically distributed throughout the search space. Experiments show that this method might get better results than Random Immigrants and Hyper-mutation in problems with large and chaotic changes [13].

Two other approaches - Parallel PBIL (PPBIL2) and Dual PBIL (DPBIL) were proposed by Yang and Yao [91]. These methods are based on the Population-based Incremental Learning (PBIL) algorithm, which is a simple combination of population-based EA and incremental learning. PBIL has an adjustable probability vector which is used to generate individuals. After each generation the probability vector is updated based on the best found solutions. It ensures that the vector will gradually "learn" the appropriate value to generate high quality individuals. In PPBIL2, Yang and Yao [91] improved PBIL for DOPs by maintaining two parallel probability vectors: a vector similar to the original one in PBIL and a random initialised probability dedicated to maintain diversity during the search. The two vectors are sampled and updated independently so that their sample sizes might be adjusted based on their relative performance. To improve PBIL2 in dealing with large changes, Yang and Yao [91] proposed the DPBIL where two probability vectors are *dual* with each other, i.e., given the first vector P_1, the second vector P_2 is determined by $P_2[i] = 1 - P_1[i]$ $(i = 1, ..., n)$, where n is the number of variables. During the search only P_1 needs to learn from the best generated solution because P2 will change with P_1 automatically. PBIL and dual PBIL were also combined with random-immigrant in [48] with better results than the original algorithms.

Diversity can also be maintained by rewarding individuals that are genetically different to their parents [99]. In this approach, in addition to a regular GA population, the algorithm maintains an additional population where individuals are selected based on their Hamming distance to their parents (to promote diversity) and another population where individuals are selected based on their fitness improvement compared to their parents (to promote exploitation). By observing its own performance in stagnation and population diversity, the algorithm adaptively adjusts the size of the three populations to react to the dynamic environments.

The approach of maintaining diversity is also used in PSO to solve dynamic continuous problems. In their charged PSOs [93, 94, 95], Blackwell *et al.* applied a *repulsion* mechanism, which is inspired from the atom field, to prevent particles/swarms to get too closed to each other. In this mechanism, each swarm is comprised of a nucleus and a cloud of charged particles which are responsible to maintain diversity. There is a repulsion among these particles to keep particles from approaching near to each other. In [85], both the particle selection and replacement mechanisms are modified so that the most diversified particles (in term of Hamming distance) are selected and the particles that have similar positions are replaced. In the Compound PSO [100], the degree of particles deviating from their original directions becomes larger when the velocities becomes smaller, and distance information was incorporated as one of the criteria to choose a particle for the update mechanism.

Bui *et al.* [96] proposed using multiple objectives to maintain diversity. The dynamic problem is represented as a two-objective problem. The first objective is the original objective, and the second is a special objective created to maintain diversity. Other examples of using multiple objectives to maintain diversity can be found in [97] and [98], where they proposed six different types of objectives, including retaining more old solutions; retaining more random solutions; reversing the first objective; keeping a distance from the closest neighbour; keeping a distance from all individuals; and keeping a distance from the best individual.

The diversity-maintaining strategy is still the main strategy in many recent approaches, for example, see [95, 94, 101, 102, 15, 103, 92, 91, 48, 67, 104].

4.4.2. Strengths and weaknesses

Methods following this approach can bring the following advantages:

- *May be good for solving problems with severe changes*: Thanks to its good diversity, in certain situations the approach is good to solve problems with large changes (for example in [25, 15] it has been shown that random-immigrant help significantly improve the performance in dynamic constrained problems where changes are severe due to the presence of disconnected feasible regions)

- *May be good for solving problem with rare changes* (as shown in e.g. [89, 91]). This is because for rare changes an algorithm with high diversity may have enough time to converge.

- *May be effective in solving problems with competing peaks* (as reported in [105])

However, methods that maintain diversity throughout the search also have some disadvantages as follows:

- *Slow*: Continuously focusing on diversity may slows down, or even distract the optimisation process [12].

- *Not effective when the changes are small*: Most methods following this approach maintain their diversity by adding some stochastic element throughout the search. Obviously it will make the algorithm less effective in dealing with small changes where the optima just take a slight move away from their previous places [27].

4.5. Memory Approaches

When changes in dynamic problems are periodical or recurrent, i.e. the optima may return to the regions near their previous locations, it might be useful to re-use previously found solutions to save computational time and to bias the search process. To re-use old solutions in this manner, many researchers decided to add some sort of memory components to their EAs. The memory can also play the role as a reserved place for storing old solutions in order to maintain diversity when needed. The memory can be integrated *implicitly* as a redundant representation in the EAs, or it could be maintained *explicitly* as a separate memory component.

4.5.1. Implicit memory

Redundant coding using diploid genomes are the most common implicit memory used in EAs for solving dynamic problems, e.g., [18, 106, 107, 108, 109]. A diploid EA is usually an algorithm whose chromosomes contain two alleles at each locus. Although most normal EAs for stationary are haploid, it is believed that diploid, and other multiploid approaches, are suitable for solving non-stationary problems [106]. A pseudo code for multiploid approaches for dynamic environments is described in Algorithm 3, where the following three components need to be incorporated: (i) represent the redundant code; (ii)readjust the dominance of alleles; and (iii) check for changes.

One typical way to represent the dominance of alleles is to use a table [107, 110] or a mask [111] mapping between genotypes and phenotypes. The dominance then can be changed adaptively among alleles depending on the detection of changes in the landscape.

Algorithm 3 Multiploid EA for dynamic optimisation

```
1. Initialise:: Initialise the population and the multiploid representation
2. For each generation
   (a) Evaluate: Evaluate each member of the population
   (b) For each individual:
       i. Detect changes
       ii. Adjust the dominance level of each allele : If there is any change,
           adjust the dominance to accommodate the current change
       iii. Select the dominant alleles according to their dominance level
   (c) Reproduce: Reproduce a new population using the adjusted mutations
   (d) Return to step 2a
```

Algorithm 4 Using explicit memory

```
1. Initialise:
   (a) Initialise the population
   (b) Initialise the explicit memory
2. For each generation
   (a) Evaluate each member of the population
   (b) Update the memory
   (c) Reproduce a new population
   (d) Use information from the memory to update the new population
   (e) Return to step 2a
```

4.5.2. Explicit memory

Methods that maintain the memory explicitly are described by the pseudo code in Algorithm 4:

Methods following this approach need to accomplish four tasks:

1. *Decide the content of the explicit memory*: The content of the memory can be either:
   (a) *Direct memory:* In most cases the direct memories are the previous good solutions/local optima [112, 21, 113, 114, 115, 116, 48, 117, 118, 119, 85, 120, 88]. In [117] for certain circumstances the most diversified solutions (in term of standard deviation of fitness) are also selected for the memory. In [85] a set of previous positions and the corresponding fitness values of each individual may also be stored in the memory.
   (b) *Associative memory:* Various type of information can be included in the associative memory, for example the information about the environment at the considered time [121, 122]; the list of environmental states and state transition probabilities [123]; the most common allelle in the population for each locus [88]; the probability vector that created the best solutions [48]; the distribution statistics information of the population at the considered time [116]; the probability of the occurrence of good solutions in each area of the landscape [124, 87]; or the probability of likely feasible regions [26]. An interesting example of associative memory was shown in Artificial Immune Systems (AIS) [125], [126]. In these methods, changes in a dynamic environment are usually viewed as antigens and the "building blocks" (gene segments) from successful individuals in the past are considered as antibodies. The gene segments are stored as memory in a gene library so that they can be recalled whenever a change occurs. To identify which gene segments (antibodies) should match with a particular antigen (change in the environment), each individual in the gene library is associated with the average fitness of the population at the moment it was stored [125].
2. *Decide how to update the memory:* Generally the best found elements (direct or associative) of the current generation will be used to update the memory. These newly found elements will replace some existing elements in the memory, which can be one or some of the followings:
   (a) The oldest member in the memory [121, 127, 28, 10]
   (b) The one with the least contributions to the diversity of the population [21, 121, 127, 115, 48, 128]. One common way to evaluate this criterion is to examine the similarity of elements in the memory, for example evaluating the minimum distance among all pairs of memory elements [21, 127]. In this case the less fit one of a pair will be replaced.

    (c) The one with least contribution to fitness [121]

3. *Decide when to update the memory*: Ideally if we know exactly when a change happens, then the most suitable time to update the memory is right after the time the change happens. However, in general it might not always be possible to know exactly when a change happens. As a result the memory may also be updated after each generation or after a certain number of generations. Doing so might also favour diversity, for example see [1, 129, 117].
4. *Decide how to use the memory*: Usually the best elements in the memory (i.e. the ones that show the best results when re-evaluated) will be used to replace the worst individuals in the population. Replacement can take place after each generation or after a certain number of generations, or it can be done after each change if the change can be recognised.

4.5.3. Strengths and weaknesses

Here are the advantages of using memory-based approaches:

1. *Effective for solving problems with cyclic environments.* Thanks to their ability to recall old solutions from the memory, memory-based approaches are especially suitable for solving problems with cyclic changes. For example, Yang [60] showed that the memory-based versions of GA and random-immigrant significantly outperform the original algorithms in cyclic dynamic environments.
2. *May be good in slowing down convergence and favour diversity* [1, 129].

Memory approaches, however, also have some disadvantages that may require them to be integrated with some other methods for the best results:

1. *Might be useful only when optima reappear at their previous locations or if the environment returns to its previous states.* In the experiments, Lewis *et al.* [106] showed that redundant coding does not ensure enough diversity to adapt to random changes. Branke [21] also mentions that some explicit memory approaches might no longer be effective if the oscillation does not bring the global optimum to the exact previous location but a slightly different one [1].
2. *Might not be good enough to maintain diversity for the population*, and in fact, memory may not be very useful unless combined with some diversity mechanism as pointed out by Branke [21]. Recently, several studies have tried to improve this disadvantage by combining memory-based approaches with diversity schemes, e.g., [127, 60].
3. *Redundant coding approaches might not be good for cases where the number of oscillating states is large.* There are two reasons for this:

   (a) First, the redundant code might become too large, hence reduce the performance of the algorithm.
   (b) Second, in practice it might not always be possible to know the number of oscillating states before hand. Without this information, it is impossible to design an appropriate representation for the redundant code.

4. *The information stored in the memory might become redundant (and obsolete) when the environment changes.* This redundancy may affect the performance of the algorithm. For example, Branke [19] empirically showed that memories are of no use if there is no recurrence in the environments.

4.6. Prediction approaches

In certain cases, changes in dynamic environments may exhibit some patterns that are predictable. In this case, it might be sensible to try to learn these types of patterns from the previous search experience and based on these patterns try to predict changes in the future. Some studies have been made following this idea to exploit the predictability of dynamic environments. Obviously, memory approaches, which are proposed to deal with periodical changes, can also be considered a special type of prediction approaches. However, generally methods following the prediction approach are able to use their memory to cope with more various types of changes than only cyclic/recurrent changes. A pseudo code describing prediction approaches is shown in Algorithm 5.

Algorithm 5 Prediction approach to solve dynamic problems

```
1. Initialise phase:
   (a) Initialise the population
   (b) Initialise the learning model and training set
2. Search for optimum solutions and detect changes
3. If a change is detected
   (a) Use the current environment state as the input for the learning model
   (b) Use the learning model to estimate the type of this current change
       and/or how the next change should be
   (c) Generate new /recall old individuals that best match with the estima-
       tion
   (d) Search for the new optimum using the new population
   (e) Update the training set based on the search results
4. Return to step 2
```

A common prediction approach is to predict the movement of the moving optima. Hatzakis and Wallace [65] combined a forecasting technique (autoregressive) with an EA. This forecasting technique is used to predict the location of the next optimal solution after a change is detected. The forecasting model (time series model) is created using a sequence of optimum positions found in the past. Experimental results show that if this algorithm can predict the movements of optima correctly, it can work well with very fast changes. A similar approach was proposed in [130] where the movement of optima was predicted using Kalman filters. The predicted information (the next location of the optimum) is incorporated into an EA in three ways: First, the mutation operator is modified by introducing some bias so that individuals' exploration is directed toward the predicted region. Second, the fitness function is modified so that individuals close to the estimated future position are rewarded. Third, some "gift" individuals are generated at the predicted position, and introduced into the population to guide the search. Experiments on a visual tracking benchmark problem show that the proposed method does improve the tracking of the optimum, both in terms of distance to the real position and smoothness of the tracking.

Another approach is to predict the locations that individuals should be re-initialised to when a change occurs. In [34] this approach is used to solve two dynamic multi-objective optimisation benchmark problems in two ways: First, the solutions in the Pareto set from the previous change periods were used as a time series to predict the next re-initialisation locations. Second, to improve the chance of the initial population to cover the new Pareto set, the predicted re-initialisation population is perturbed with a Gaussian noise whose variance is estimated based on historical data. Compared with random-initialisation, the approach was able to achieve better results on the two tested problems. Another approach to estimate the areas to re-initialise individuals after a change occurs is the relocation variable method [10] described in Subsection 4.3. This method to some extents can also be considered a prediction method.

Another interesting approach is to predict the time when the next change will occur and which possible environments will appear in the next change [123, 131]. In these works, the authors used two prediction modules to predict two different factors. The first module, which uses either a linear regression [123] or a non-linear regression [131], is used to estimate the generation when the next change will occur. The second module, which uses Markov chain, monitors the transitions of previous environments and based on this data provides estimations of which environment will appear in the next change. Experimental results show that an EA with the proposed predictor is able to perform better than a regular EA in cyclic/periodic environments.

Relating to prediction approaches, recently there are also some studies [24, 9, 132, 16] on time-linkage problems, i.e. problems where the current solutions made by the algorithms can influence the future dynamics. In such problems, it was suggested that the only way to solve the problems effectively is to predict future changes and take into account the possible future outcomes when solving the problems online. Another related study is the anticipation approach [133] in solving dynamic scheduling problems where in addition to finding good solutions, the solver also tries to move the system "into a flexible state" where adaptation to changes can be done more easily. Specifically, because it is observed that in the tested dynamic job-shop scheduling problem, the flexibility of the system can be increased by avoiding early machine idle times, the authors proposed a scheduling approach where in addition to the main optimality objective, solutions with early idle time are penalised. The experimental results show that such an anticipation approach significantly improved the performance of the system.

4.6.1. Strengths and weaknesses

Methods following the prediction approach may become very effective if their predictions are correct. In this case, the algorithms can detect/track/find the global optima quickly, as shown in [65], [126] and [125].

However, prediction-based algorithms also have their own disadvantages, mostly due to training errors. These errors might be resulted from:

1. *Wrong training data*: If the algorithm has not performed successfully in the previous change periods, the history data collected by the algorithm might not be helpful for the prediction or might even provide the wrong training data.
2. *Lack of training data*: As in the case of any learning/predicting/forecasting model, the algorithms may need a large enough set of training data to produce good results. It also means that the prediction can only be started after sufficient training data has been collected, e.g., [123, 131, 24, 9]. In the case of dynamic optimisation where there is a need of finding/tracking the optima as quick as possible, this might be a disadvantage.
3. *The nature of the dynamic problems*:
    - If changes in the dynamic environment are easily predictable (e.g., linear, periodical or deterministic), the result is expected to be good, as can be seen in [65, 130].
    - However, if the changes are stochastic, or history data is misleading, prediction approaches might not get satisfiable results. For example, Nguyen and Yao [16] illustrated a situation where history data are actually inappropriate for the prediction and might even mislead the predictor to get worse results.

4.7. Self-adaptive methods

Another approach is to make use of the self-adaptive mechanisms of EAs and other meta-heuristics to cope with changes. To some extent this approach closely relates to the prediction approach, because deep down self adaptation is the outcome of a process involving learning and predicting based on history data.

One example is the GA with Genetic Mutation Rate [37], which allows the algorithm to evolve its own mutation strategy parameters during the search process based on the fitness of the population. In this method, the mutation rate is encoded in genes and is influenced by the selection process. The algorithm was tested in both gradual and abrupt dynamic landscapes. The results show that the algorithm has better performance than a standard GA. However, it is still not better than hyper-mutation (see section 4.3 and [49]) - a method that increases its mutation rate after each change.

A similar method was proposed by Ursem in his Multinational Genetic Algorithm (MGA) [134]. Five different parameters (probability for mutation, probability for crossover, selection ratio, mutation variance and distance) are encoded in the genomes of his MGA for adaptation. The adaptation mechanism works well in simple cases where the velocity of moving peaks is constant. However, in cases where the velocity is not constant, the adaptation seems to be not fast enough. These two results show the difficulty of applying adaptive parameter tuning to complex dynamic optimisation.

Some researchers also expressed their interests in using the self-adaptive mechanism of such EAs as Evolution Strategy (ES) or EP (Evolutionary Programming) in dynamic optimisation. Angeline [135] examined self-adaptive EP (saEP) and showed that the strategy is not effective for all types of tested problems. Bäck [8] showed that the log-normal self-adaptation in ES may perform better than saEP. Experiments pointed out that algorithm implementation and parameter settings have much less influence on ES in dynamic environments than in stationary environments [56] and that ES might be unreliable in rapidly changing environment [55]. Weicker [7] also argued that it is possible that the Gaussian mutation in the standard ES self-adaptation might not be appropriate for dynamic optimisation.

There are some mathematical analyses on the performance of self-adaptive ES in dynamic environments. Arnold and Beyer [136] pointed out that the cumulative mutation step-size adaptation of ES can work well on a variant of the sphere model with random dynamics of the target. The strategy can realise optimal mutation step-size for the model. However, in the sphere modal with linear dynamics, another research of Arnold and Beyer [137] revealed that the mutation step-size realised by ES is not the optimal one (but the adaptation still ensures that the target can be tracked).

4.8. Multi-population approaches

4.8.1. Overview

Another approach, which to some extent can be seen as a combination of diversity maintaining/introducing, memory and adaptation, is to maintain multiple sub-populations concurrently. Each sub-population may handle a separate area of the search space. Each of them may also take responsibility for a separate task. For example, some sub-populations may focus on searching for the global optimum while some others may concentrate on tracking any possible changes. These two types of populations then may communicate with each other to bias the search. A pseudo code of a typical multi-population approach is shown below as Algorithm 6.

Algorithm 6 Multi-population approach

```
1. Initialise::
    (a) Initialise the set$P_{search}$ of sub populations finding the global optima
    (b) Initialise the set $P_{track}$ of sub populations tracking changes in the landscape
2. For each generation:
    (a) Search for optima: Sub-populations in $P_{search}$ find the global optima
    (b) Track changes: Sub-populations in $P_{track}$ track any changes
    (c) Maintain diversity: Re-allocate/split/merge the sub-populations so
        that they are not overlapped and can cover a larger area of the search space
    (d) Adjust: Re-adjust each sub-population in $P_{search}$ based on the experience
        from sub-populations in $P_{track}$
    (e) Reproduce each sub-population
    (f) Return to step 2a
```

As can be seen, methods following the approach of using multiple populations usually need to accomplish two goals: First, they may need to assign different types of tasks to different sub-populations, for example P_{search} to search and P_{track} to track, so that the search can be done effectively. Second, they need to divide the sub-populations appropriately and make sure that the sub-populations are not overlapped to have the best diversity and also to avoid the situation where many sub-populations find the same peak.

For the first goal, assigning different tasks to the sub-populations, different methods have different approaches. One approach was proposed by Oppacher and Wineberg [58] in their Shifting Balance GA (SBGA). In SBGA, there are a number of small populations in P_{search} searching for new solutions and there is only one large population in P_{track} to track changing peaks.

Another method, the Self-Organizing Scouts (SOS) [138], follows a different direction which uses the main large population to search for optima (P_{search}) and dedicates several small populations to track any change of each optimum that the algorithm has found so far (P_{track}). Whenever the main population finds a new peak, it will create a new sub-population to track changes in this peak. This approach was adopted in different types of EAs and meta-heuristics, e.g., GA [101], DE [139, 140], and PSO [95, 141]. Relating to using one large population to search and a smaller population to track changes, an algorithm named RepairGA for solving dynamic constrained problems was proposed in [25]. In this method, a large sub-population is dedicated to searching and one smaller sub-population is dedicated to tracking the moving feasible regions. The difference between RepairGA and previous approaches is that in RepairGA the two sub-populations are allowed to overlap in the search space because their main purpose is not to maintain diversity. What distinguishes the two sub-populations in this work is that the main population accepts both infeasible and feasible solutions while the sub-population contains only feasible solutions.

Another approach, the Multinational GA (MGA), introduced by Ursem [134], integrates both the functions of P_{search} and P_{track} into each sub-population. It means that each population can both search for new solutions and track changes. Whenever a sub-population detects a new optimum, it will split into two sub-populations to make sure that each sub-population only tracks one optimum at a time. This approach has been used not only in EAs but also in artificial immune algorithms, for example [102]. The approach is also used by PSO-based algorithms for dynamic optimisation. One example is the Speciation PSO [71] where each sub-population, or species, is a hyper-sphere defined by the best fit individual and a specific radius. Another recent PSO example that also has multi swarms with equal roles is the Clustering PSO in [142, 143].

Relating to the goal of assigning the tasks to sub-populations, it should be noted that in dynamic optimisation multiple populations are used not only for the purpose of exploring different parts of the search space, but also for the purpose of co-evolution [62] [25] or maintaining diversity and balancing exploitation/exploration [99].

For the second goal, *dividing the sub-populations and making sure that the sub-populations are not overlapping*, there are also different approaches. The most common approach is clustering: choosing some solutions in the population as the centres of the future clusters, then defining each sub-population as a hyper-cube or sphere with a given size. All individuals within the range of a hyper-cube/sphere will belong to the corresponding sub-population of that hyper-cube/sphere. SOS [138] is one the earliest methods that adopt this approach. It keeps the sub-populations from being overlapped by using an idea borrowed from the Forking

Genetic Algorithm (FGA) [144] to divide up the space. Whenever the main population in P_{search} finds a new optimum, it creates a new population in P_{track} and assign this new population to the optimum. To separate the sub-populations, SOS [138] confines each sub-population to a hyper-cube determined by a centre (the most fit individual in the population) and a pre-defined range. If an individual of one sub-population ventures to the area monitored by another sub-population, this individual will simply be discarded and re-initialised (this process is called *exclusion*). The same forking approach is also used in other EAs, e.g., DE [139, 140]. Similar approaches are also used in PSO. For example, in Multi-swarm PSO (mPSO) [94], swarms are also divided into sub-swarm in the same way as in SOS so that each swarm watches a different peak. In addition, mPSO also maintains a similar mechanism (named anti-convergence) to the P_{search} in SOS so that there is always one free swarm to continue exploring the search space. Another example is in Speciation PSO [71] where each species is a hyper-sphere whose centre is the best-fit individual in the species and each species can be used to track a peak.

For clustering approaches, it is not always necessary to choose the best solutions as the centres of the clusters. In recent approaches [142, 10], density-based clustering methods are also used to divide/separate the sub-populations and to allow the algorithms explore different parts of the search landscape. It was reported that these density-based clustering techniques do help to improve the performance, but at the expense of additional computational cost to calculate the pair-wise distance among particles. The clustering-based approach is still widely used in recent EDO studies, e.g., in [101] to optimise the dynamic network routing problems.

The second approach is to incorporate some mechanism of penalty/rewarding to keep the sub-populations apart, of which SBGA [58] is a typical example. SBGA maintains the separation of populations by selecting individuals in P_{search} for reproduction according to their distance from the core in P_{track} rather than according to their original fitness values. The further an individual is from the core, the more likely that it will be reproduced.

The third approach is to estimate the basins of attractions of peaks and use these basins as the separate regions for each sub-population. MGA [134] is the first work following this approach. The authors provided a mechanism called *hill-valley detection*: given two individuals in the search spaces, they calculate the fitness of several random samples on the line between these two individuals. If the fitness in a sample point is lower than that of the two individuals, then a valley is detected. If a sub-population contains more than one valley, it will be split.

4.8.2. Strengths and weaknesses

Methods with the multi-population approach have the following advantages:

1. *Can maintain enough diversity* for the algorithm to adaptively start a new

search whenever a new change appears. Examples can be seen in the experiments in [19] where the proposed multi-population algorithm (SOS) was able to cover most of the peaks if given enough time while the non-multi-population GA could not.

2. *Able to recall some information from the previous generations* thanks to one (or several) population(s) dedicated for retaining old solutions. This makes multi-population approaches usable in solving certain recurrent dynamic problems. For example, Ursem [134] and Branke [21] showed that the multi-population MGA and memory-based EA were able to recall good old solutions to deal with recurrent problems and hence outperformed normal EAs.
3. *Can search/ track the moves of multiple optima*, as analysed in many existing studies on multi-population, e.g., [134] and [19].
4. *Can be very effective for solving problems with competing peaks or multimodal problems.* A survey of Moser [145] showed that among 19 surveyed algorithms that are designed to solve the multimodal competing peaks benchmark Moving Peaks, a majory (15 out of 19) follow the multi-population approach.

The multi-population approach also has some disadvantages. They are:

1. *Too many sub-populations may slow down the search.* For example, Blackwell and Branke [94] showed that for their multi-swarm PSO algorithm, if the number of sub-populations (swarms) is larger than the number of peaks, the performance of the algorithm decreases.
2. *Limited size of memory* or peaks that can be tracked.

4.9. Summary on the strengths and weaknesses of current EAs for DOPs

From the literature review above we can conclude that different EDO approaches seem to be best for different types of problems. This is the reason why many recent studies try to combine different approaches into one single algorithm to solve the problems better. Overall, multi-population approaches seem to be the most flexible approach to date. The survey also shows that most existing methods were tested and evaluated only on academic problems.

5. Theoretical development of EDO

Formally analysing EC for static problems has been a central theme in EC since the early days [146]. Besides static property analyses, researchers have also analysed EA's dynamic behaviour by Markov chain models, including convergence reliability [147], convergence rate [148], and expected time to reach an optimum (i.e., the *first hitting time*) [149, 150]. In contrast to the EA theory for static problems, the research on EDO so far has mainly been empirical. Theoretical analysis of EDO has just appeared in recent years with only a few results. This is because analysing EAs for DOPs is much more difficult than analysing EAs for static problems due to the extra dynamics introduced in DOPs. The theoretical studies on EDO so far are briefly reviewed as follows.

The early theoretical works on EDO mainly just extended the analysis of simple EAs, e.g., the (1+1) EA[4], for static optimisation to simple DOPs, e.g., the dynamic bit matching problem. This is natural and has served as a good starting point. As a first theoretical work on EDO, Stanhope and Daida [151] analyzed a (1+1) EA on the dynamic bit matching problem. They presented the transition probabilities of the (1+1) EA and showed that even small perturbations in the fitness function could have a significantly negative impact on the performance of the (1+1) EA. Based on the work by Stanhope and Daida [151], Branke and Wang [152] developed an analytical model for a (1, 2) evolution strategy (ES) and compared different strategies to handle an environmental change within a generation on the dynamic bit matching problem.

Droste [153] analysed the first hitting time of a (1+1) ES on the dynamic bit matching problem, where exactly one bit is changed with a given probability p after each function evaluation. It was shown that the expected first hitting time of the (1+1) ES is polynomial if and only if $p = O(logn/n)$. Arnold and Beyer [136] investigated the tracking behaviour of an $(\mu/\mu,\ \lambda)$ ES with self-adaptive mutation step-size on a single continuously moving peak. They derived a formula to predict the tracking distance of the population from the target. Jansen and Schellbach [154] presented a rigorous performance analysis of the $(1{+}\lambda)$ EA on a tracking problem in a two-dimensional lattice and showed that the expected first hitting time strictly increases with the offspring population size (i.e., λ) whereas the expected number of generations to reach the target decreases with λ. In [55], Weicker and Weicker analyzed the behaviour of ESs with several mutation variants on a simple rotating dynamic problem. In [6], Weicker presented a framework for classifying DOPs and used it to analyze the how the offspring population size and two special techniques for DOPs affect the tracking probability of a $(1, \lambda)$-ES. Weicker [155] also used Markov models to analyze the tracking behaviour of $(1, \lambda)$-ESs with different mutation operators for a discrete optimization problem with a single moving optimum.

More recently, in [156], Rohlfshagen *et al.* analyzed how the magnitude and frequency of change may affect the performance of the (1+1) EA on two specially

[4]In a (1+1) EA, there is only one solution maintained in the population. In each iteration, the unique solution acts as the parent to generate an offspring via mutation. If the fitness of the offspring is not worse than the parent, the offspring will replace the parent; otherwise, the parent will survive into the next generation.

designed pseudo-Boolean functions under the dynamic framework of the XOR DOP generator [115]. They demonstrated two counter-intuitive scenarios, i.e., the algorithm is efficient if the magnitude of change is large and inefficient when the magnitude of change is small, and the algorithm is efficient if the frequency of change is very high and inefficient if the frequency of change is sufficiently low. These results allow us to gain a better understanding of how the dynamics of a function may affect the runtime of an algorithm.

In addition to the above runtime analysis of EDO, as another line of research, some theoretical studies on EDO have been devoted to the analysis of dynamic fitness landscape. Branke *et al.* [157] analyzed the changes of the fitness landscape due to changes of the underlying problem instance and proposed a number of measures that are helpful to understand what aspects of the fitness landscape change and what information can be carried over from one stage of the problem to the next. They applied these measures to several instances of a multi-dimensional knapsack problem. In [158], Branke *et al.* further analyzed the role of representation on the fitness landscape based on the dynamic multi-dimensional knapsack problem.

In [4, 5], Rohlfshagen and Yao analyzed the properties of a dynamic subset sum problem and investigated the correlation between the dynamic parameters of the problem and the resulting movement of the global optimum. Interestingly, the authors showed empirically that the degree to which the global optimum moves in response to the underlying dynamics is correlated only in specific cases. Their observations obtained here and in [156] indicate that simply tracking an optimum is not sufficient in real-world problems.

Tinos and Yang [40] analyzed the XOR DOP generator based on a linear transformation matrix and showed that XOR does not alter the underlying function but rotates each search point prior to each fitness evaluation. In [159], Tinos and Yang further analyzed the properties of the XOR DOP generator based on the dynamical system approach of the GA in [160]. The authors showed that a DOP generated by the XOR generator can be described as a DOP with permutation, where the fitness landscape is changed according to a permutation matrix. Hence, the XOR DOP generator can be simplified by moving the initial population of a change cycle instead of rotating each individual prior to each fitness evaluation.

In [161, 162], Richter constructed spatio-temporal fitness landscapes based on Coupled Map Lattices (CML). The idea of using CML to construct dynamic fitness landscapes is interesting since CML facilitate efficient computing of the fitness landscape and can reveal a broad variety of complex spatio-temporal behavior [163]. In [124, 26], Richter further analyzed and quantified the properties of spatio-temporal fitness landscapes constructed from CML using topological and dynamical landscape measures such as modality, ruggedness, information content, epistasis, dynamic severity, and two types of dynamic complexity measures, Lyapunov exponents and bred vector dimension. Experiments were also carried out to study the relationship between these landscape measures and the performance criteria of an EA. These studies from Richter are interesting in terms of helping us to relate the landscape measures to the behavior of EDO algorithms.

6. Summary and future research directions

6.1. Summary

In this paper we have reviewed and categorised existing EDO studies from several perspectives, namely benchmark problems (Section 2), performance measures (Section 3), methodology (Section 4), and theory (Section 5).

The review showed us the strengths and weaknesses of different EDO methods. From the literature review, we can conclude that each EDO approach seems to be suitable only for certain types of DOPs, which conforms to the No Free Lunch theorem [164]. The fact that each approach is likely to be suitable to some particular classes of problems is also the reason why many recent studies try to combine different approaches into one single algorithm to solve the problems better.

The review showed us that there have been some recent works on the theory behind EDO. These theoretical studies are still quite basic. However, they have made very important first steps toward understanding EDO and will surely act as the basis for further theoretical studies on EDO.

The review also identified the common assumptions of the community about the characteristics of DOPs, which can be summarised as follows:

- *Optimisation goals*: *Optimality is the primary goal or the only goal in a majority of academic EDO studies*, as evidently shown by the large number of optimality-based measures reviewed in Section 3. Some studies do pay attention to developing other complementary measures (e.g. the behaviour-based measures in Subsection 3.2), but these complementary measures mainly focus on analysing the behaviours of the algorithms rather than checking if the algorithms satisfy users requirements.

- *The time-linkage property*: *Non time-linkage (the algorithm does not influence the future dynamics) is the main focus of current academic EDO research*, as evidently shown by the fact that all commonly used general-purpose benchmark problems are non-time-linkage.

- *Constraints*: *Unconstrained or bounded constrained problems are the main focus of academic research*, especially in the continuous domain, as shown by the majority of academic benchmark problems. There is a clear lack of studies on constrained and dynamic constrained problems.

- *Visibility and detectability of changes*: Most current EDO methods assume that changes either are known or can be easily detected using a few detectors.

- *Factors that change*: The major aspect that changes in academic problems is the objective function.

- *Predictability*: The predictability of changes has increasingly attracted the attention of the community. However, the number of studies in this topic is still relatively small compared to the unpredictable case

- *Periodicity*: The periodicity of changes is a given assumption in many mainstream approaches such as *memory* and *prediction.*

The literature review showed that not many of the assumptions above are backed up by evidence from real-world applications. This leads to the question of whether these assumptions still hold in real-world dynamic optimisation problems (DOPs) and whether the considered characteristics are representative in real-world applications. In the next subsection, we will discuss this question in detail.

6.2. The gaps between academic research and real-world problems

The lack of a clear link between EDO academic research and real-world scenarios has lead to some criticisms on how realistic current academic problems are. Ursem *et al.* [165] questioned the importance of current academic benchmarks by stating that “no research has been conducted to thoroughly evaluate how well they reflect characteristic dynamics of real-world problems”; Branke *et al.* [157] pointed out that “little has been done to characterize and understand the nature of a change in real-world problems”; Rohlfshagen and Yao [4] criticised that “a large amount of effort is directed at an academic problem that may only have little relevance in the real world”; and in [25, 16], it has been showed that there are some classes of real-world problems whose characteristics have not been captured by existing academic research yet.

Most recently, for the first time a detailed review [15, chap. 3] of a large set of recent “real”[5] real-world dynamic optimisation problems has been made to investigate the characteristics of real-world problems and how they relate to the characteristics of current academic benchmark problems.

The investigation in [15] pointed out certain gaps between academic EDO research and real-world DOPs. First, current studies in academic EDO do not cover

[5]Only references that actually use real-world data or solve problems in actual real-world situations were considered. Benchmark problems, even if designed to simulate real-world applications, were not considered unless there is evidence that the data used to create the benchmark were taken from real-world applications.

all types of common dynamic optimisation problems yet. As shown in Section 2, the most common type of DOPs that current academic research considers are unconstrained, non-time-linkage problems. However, the study in [15] showed that this type of problem occupies only a small part of the surveyed applications. The study also found that there are two types of problems that are very common in real-world situations but received very little attention from the community: dynamic constrained problems and time-linkage problems.

Second, although many of current EDO academic research works only focus on one major optimisation goal: optimality (to find the best fitness value, as shown in Subsection 3.1), the study in [15] showed that there might be many other common optimisation goals, for example (a) to provide a new (decent) solution quickly; (b) to make sure that the after-change solution is not very different from the before-change solution; (c) to make sure the new solutions are as close to a reference solution as possible; and (d) to make sure that the future solutions must be in certain bounds.

Third, although most current EDO artificial benchmark problems have only one changing factor: the objective function as shown in Section 2, the study in [15] showed that there are also other common types of changing factors: constraints, number of variables, domain ranges and switch-mode changes.

In summary, the review in [15] showed that besides the characteristics and assumptions commonly used in EDO academic research, real-world DOPs also have other important types of problems and problem characteristics that have not been studied extensively by the EDO community. In order to solve real-world DOPs more effectively, it is necessary to take these characteristics and problem types into account when designing new algorithms, performance measures and benchmark problems.

6.3. Future research directions

As reviewed in this paper, there have been quite a lot of studies devoted to EDO and fruitful results have been achieved over the last 20 years. However, the research domain of EDO is still relatively young. Much more effort is needed to fully develop and understand the domain of EDO. Some future research directions on EDO are highlighted and suggested as follows.

- *Benchmark problem*: The survey in this paper shows that most existing methods were tested and evaluated only on academic problems. The question then is to find out (i) what are the common characteristics of existing academic problems; (ii) what are the common criteria to evaluate EDO algorithms; and more importantly (iii) whether these common characteristics and evaluation criteria reflect the common situations in real-world scenarios.

As mentioned before, the common assumptions of the EDO community about the characteristics of DOPs are not totally backed up by evidence from real-world applications. This leads to the question of whether these academic assumptions still hold in real-world DOPs and, if yes, then whether these assumptions are representative in real-world applications and in what type of applications do they hold. The review in [15] was the first study which attempts to answer the above questions and it has pointed out that there are certain gaps between current EDO academic research and real-world applications. In future research on EDO, further investigations should be made to close these gaps and accordingly to bring EDO research closer to realistic scenarios.

- *Methodology research*: Although a number of EDO approaches have been developed for solving DOPs, new efficient approaches are still greatly needed to address different types of DOPs. As the review has shown, different methods have different strengths and weaknesses for different DOPs. Hence, it is also worthy to further develop and investigate hybrid methods for DOPs in the future. Here, it is very important to develop adaptive systems that can deal with DOPs of different characteristics.

- *Theoretical research*: As mentioned before, the theoretical studies on EDO are quite limited so far. The relative lack of theoretical results on EDO makes it hard to fully justify the strengths and weaknesses of EDO algorithms and predict their behaviour for different DOPs. Hence, theoretical studies on EDO are greatly needed for the domain. As reviewed, the computational complexity analysis of EDO has started with a few results. But, this line of research needs to be enhanced significantly in order to gain insights as to what DOPs are hard or easy for what types of EDO algorithms. Here, techniques for analyzing evolutionary optimization for static problems, e.g., drift analysis [150, 166], may be applied or adapted to analyze EDO.

  Dynamic behaviour analysis of EDO algorithms needs also to be pursued or enhanced. For many real-world DOPs, it is more useful to know how well an algorithm tracks the moving optimal solution(s) within certain acceptable error levels rather than whether and how fast the algorithm could hit the moving optima. Hence, we need to analyse EDO regarding such properties as tracking error and tracking velocity.

- *Application research*: In this paper, we have mainly focused on reviewing academic research on EDO although there have been a number of real-world application studies on EDO, for example, see [101, 167, 168, 169, 15]. However, the number of EDO application studies so far is significantly below expectation. Researchers in the EDO domain need to consider and model more real-world DOPs, which is a challenging task, and apply EDO and other meta-heuristic methods to solve them in the future. This will further justify and promote the domain of EDO in particular and the domain of optimization in dynamic environments in general.

Acknowledgement

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of UK under Grant EP/E058884/1, Grant EP/E060722/1, Grant EP/E060722/2, and a UK ORS Award and a studentship from the School of Computer Science, University of Birmingham.

References

[1] J. Branke, Evolutionary approaches to dynamic environments - updated survey, in: GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems, 2001, pp. 27–30.

[2] S. Yang, Y. Jin, Y. S. Ong (Eds.), Evolutionary Computation in Dynamic and Uncertain Environments, Springer-Verlag Berlin Heidelberg, 2007.

[3] V. S. Aragon, S. C. Esquivel, An evolutionary algorithm to track changes of optimum value locations in dynamic environments, Journal of Computer Science and Technology 4 (3) (2004) 127–134.

[4] P. Rohlfshagen, X. Yao, Attributes of dynamic combinatorial optimisation, in: Proceedings of the 10th International Conference on Parallel Problem Solving From Nature (PPSN X), Vol. 5361 of Lecture Notes in Computer Science, 2008, pp. 442–451.

[5] P. Rohlfshagen, X. Yao, On the role of modularity in evolutionary dynamic optimisation, in: Proceedings of the 2010 IEEE Congress on Evolutionary Computation, CEC'10, 2010, pp. 3539–3546.

[6] K. Weicker, An analysis of dynamic severity and population size, in: M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, H.-P. Schwefel (Eds.), Parallel Problem Solving from Nature (PPSN VI), Vol. 1917 of LNCS, Springer, 2000.

[7] K. Weicker, Evolutionary algorithms and dynamic optimization problems, Der Andere Verlag, 2003.

[8] T. Bäck, On the behavior of evolutionary algorithms in dynamic environments, in: IEEE International Conference on Evolutionary Computation, IEEE, 1998, pp. 446–451.

[9] P. A. N. Bosman, Learning and anticipation in online dynamic optimization, in: S. Yang, Y.-S. Ong, Y. Jin (Eds.), Evolutionary Computation in Dynamic and Uncertain Environments, Vol. 51 of Studies in Computational Intelligence, Springer, 2007, pp. 129–152.

[10] Y. G. Woldesenbet, G. G. Yen, Dynamic evolutionary algorithm with variable relocation, IEEE Transactions on Evolutionary Computation 13 (3) (2009) 500–513.

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

[12] Y. Jin, J. Branke, Evolutionary optimization in uncertain environments—a survey, IEEE Transactions on Evolutionary Computation 9 (3) (2005) 303–317.

[13] R. W. Morrison, Designing Evolutionary Algorithms for Dynamic Environments, no. ISBN 3-540-21231-0, Springer-Verlag, Berlin, 2004.

[14] X. Yu, Y. Jin, K. Tang, X. Yao, Robust optimization over time – a new perspective on dynamic optimization problems, in: Proceedings of the 2010 IEEE Wolrd Congress on Computational Intelligence, WCCI 2010, Spain, 2010, pp. 3998–4003.

[15] T. T. Nguyen, Continuous Dynamic Optimisation Using Evolutionary Algorithms, Ph.D. thesis, School of Computer Science, University of Birmingham, `http://etheses.bham.ac.uk/1296` and `http://www.staff.ljmu.ac.uk/enrtngu1/theses/phd_thesis_nguyen.pdf` (January 2011).

[16] T. T. Nguyen, X. Yao, Dynamic time-linkage problem revisited, in: M. Giacobini, P. Machado, A. Brabazon, J. McCormack, et al. (Eds.), Proceedings of the 2009 European Workshops on Applications of Evolutionary Computation, EvoWorkshops 2009, Vol. 5484 of Lecture Notes in Computer Science, 2009, pp. 735–744.

[17] L. Fogel, A. Owens, M. Walsh, Artificial intelligence through simulated evolution, John Wiley & Sons Inc,, 1966.

[18] D. E. Goldberg, R. E. Smith, Nonstationary function optimization using genetic algorithms with dominance and diploidy, in: J. J. Grefenstette (Ed.), International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, 1987, pp. 59–68.

[19] J. Branke, Evolutionary Optimization in Dynamic Environments, Kluwer, 2001.

[20] C.-K. Goh, K. C. Tan, Evolutionary Multi-objective Optimization in Uncertain Environments: Issues and Algorithms, Springer Publishing Company, Incorporated, 2009.

[21] J. Branke, Memory enhanced evolutionary algorithms for changing optimization problems, in: Congress on Evolutionary Computation CEC99, Vol. 3, IEEE, 1999, pp. 1875–1882.

[22] A. Younes, Adapting evolutionary approaches for optimization in dynamic environments, Doctor of Philosophy (PhD) in Systems Design Engineering, Faculty of Engineering, University of Waterloo, Canada, Faculty of Engineering, University of Waterloo, Canada (2006).

[23] S. Yang, Constructing dynamic test environments for genetic algorithms based on problem difficulty, in: Congress on Evolutionary Computation, Vol. 2, 2004, pp. 1262–1269.

[24] P. A. N. Bosman, Learning, anticipation and time-deception in evolutionary online dynamic optimization, in: S. Yang, J. Branke (Eds.), GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization, 2005.

[25] T. T. Nguyen, X. Yao, Benchmarking and solving dynamic constrained problems, in: Proceedings of the IEEE Congress on Evolutionary Computation CEC2009, IEEE Press, 2009, pp. 690–697.

[26] H. Richter, Memory design for constrained dynamic optimization problems, in: Proceedings of the European Conference on the Applications of Evolutionary Computation 2010, EvoApplications 2010, Vol. 6024 of Lecture Notes in Computer Science, Springer, 2010, pp. 552–561.

[27] H. G. Cobb, J. J. Grefenstette, Genetic algorithms for tracking changing environments, in: International Conference on Genetic Algorithms, Morgan Kaufmann, 1993, pp. 523–530.

[28] K. Trojanowski, Z. Michalewicz, Searching for optima in non-stationary environments, in: Congress on Evolutionary Computation, Vol. 3, IEEE, 1999, pp. 1843–1850.

[29] C. Li, S. Yang, T. T. Nguyen, E. L. Yu, X. Yao, Y. Jin, H.-G. Beyer, P. . N. Suganthan, Benchmark Generator for CEC 2009 Competition on Dynamic Optimization, Tech. rep., University of Leicester and University of Birmingham, UK (2008).

[30] M. Abello, L. T. Bui, Z. Michalewicz, An adaptive approach for solving dynamic scheduling with time-varying number of tasks - part i, in: Evolutionary Computation (CEC), 2011 IEEE Congress on, 2011, pp. 1711 –1718.

[31] Y. Jin, B. Sendhoff, Constructing dynamic optimization test problems using the multi-objective optimization concept, in: G. R. Raidl (Ed.), Applications of evolutionary computing, Vol. 3005 of LNCS, Springer, 2004, pp. 525–536.

[32] M. Farina, K. Deb, P. Amato, Dynamic multiobjective optimization problems: test cases, approximations, and applications, IEEE Transactions on Evolutionary Computation 8 (5) (2004) 425–442.

[33] M. Helbig, A. Engelbrecht, Archive management for dynamic multi-objective optimisation problems using vector evaluated particle swarm optimisation, in: Evolutionary Computation (CEC), 2011 IEEE Congress on, 2011, pp. 2047 –2054.

[34] A. Zhou, Y. Jin, Q. Zhang, B. Sendhoff, E. Tsang, Prediction-based population re-initialization for evolutionary dynamic multi-objective optimization, in: S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, T. Murata (Eds.), Evolutionary Multi-Criterion Optimization, Vol. 4403 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2007, pp. 832–846.

[35] R. W. Morrison, K. A. DeJong, A test problem generator for non-stationary environments, in: Congress on Evolutionary Computation, Vol. 3, IEEE, 1999, pp. 2047–2053.

[36] R. Morrison, Performance measurement in dynamic environments, in: J. Branke (Ed.), GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems, 2003, pp. 5–8.

[37] J. J. Grefenstette, Evolvability in dynamic fitness landscapes: A genetic algorithm approach, in: Congress on Evolutionary Computation, Vol. 3, IEEE, 1999, pp. 2031–2038.

[38] K. Weicker, N. Weicker, Dynamic rotation and partial visibility, in: Congress on Evolutionary Computation, 2000, pp. 1125–1131.

[39] W. Tfaili, J. Dréo, P. Siarry, Fitting of an ant colony approach to dynamic optimization through a new set of test functions, International Journal of Computational Intelligence Research 3 (2007) 205–218.

[40] R. Tinos, S. Yang, Continuous dynamic problem generators for evolutionary algorithms, in: Proceedings of the 2007 IEEE Congress on Evolutionary Computation, 2007, pp. 236–243.

[41] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, S. Tiwari, Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization, Tech. rep., Nanyang Technology University, Singapore (2005).

[42] R. Salomon, Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions: A survey of some theoretical and practical aspects of genetic algorithms, BioSystems 39 (1996) 263–278.

[43] H. Richter, Detecting change in dynamic fitness landscapes, in: Proceedings of the IEEE 2009 Congress on Evolutionary Computation, CEC2009, 2009, pp. 1613–1620.

[44] S. A. Stanhope, J. M. Daida, Optimal mutation and crossover rates for a genetic algorithm operating in a dynamic environment, in: Evolutionary Programming VII, no. 1447 in LNCS, Springer, 1998, pp. 693–702.

[45] S. Yang, Non-stationary problems optimization using the primal-dual. genetic algorithm, in: Proceedings of Congress on Evolutionary Computation 2003 CEC'03, Vol. 3, 2003, pp. 2246–2253.

[46] S. Yang, X. Yao, Dual population-based incremental learning for problem optimization in dynamic environments, in: Proceedings of the 7th Asia Pacific Symposium on Intelligent and Evolutionary Systems, 2003, pp. 49–56.

[47] S. Yang, Memory-enhanced univariate marginal distribution algorithms for dynamic optimization problems, in: Congress on Evolutionary Computation, Vol. 3, IEEE press, 2005, pp. 2560–2567.

[48] S. Yang, X. Yao, Population-based incremental learning with associative memory for dynamic environments, IEEE Transactions on Evolutionary Computation 12 (5) (2008) 542 –561.

[49] H. G. Cobb, An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuouis, time-dependent nonstationary environments, Technical Report AIC-90-001, Naval Research Laboratory, Washington, USA (1990).

[50] J. J. Grefenstette, Genetic algorithms for changing environments, in: R. Maenner, B. Manderick (Eds.), Parallel Problem Solving from Nature 2, North Holland, 1992, pp. 137–144.

[51] A. Gaspar, P. Collard, From GAs to Artificial Immune Systems: Improving Adaptation in Time Dependent Optimization, in: Congress on Evolutionary Computation, Vol. 3, IEEE, 1999, pp. 1859 – 1866.

[52] J. Branke, H. Schmeck, Designing evolutionary algorithms for dynamic optimization problems, in: S. Tsutsui, A. Ghosh (Eds.), Theory and Application of Evolutionary Computation: Recent Trends, Springer, 2003, pp. 239–262.

[53] W. Feng, T. Brune, L. Chan, M. Chowdhury, C. Kuek, Y. Li, Benchmarks for testing evolutionary algorithms, Tech. rep., Center for System and Control, University of Glasgow (1997).

[54] K. Weicker, Performance measures for dynamic environments, in: J. Merelo, P. Adamidis, H.-G. Beyer, J. Fernández-Villacañas, H.-P. Schwefel (Eds.), Parallel Problem Solving from Nature, Vol. 2439 of LNCS, Springer, 2002, pp. 64–73.

[55] K. Weicker, N. Weicker, On evolution strategy optimization in dynamic environments, in: Congress on Evolutionary Computation, Vol. 3, 1999, pp. 2039–2046.

[56] R. Salomon, P. Eggenberger, Adaptation on the evolutionary time scale: A working hypothesis and basic experiments, in: J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, D. Snyers (Eds.), 3rd European Conference on Artificial Evolution, no. 1363 in LNCS, Springer, 1997, pp. 251–262.

[57] N. Mori, S. Imanishi, H. Kita, Y. Nishikawa, Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm, in: T. Bäck (Ed.), International Conference on Genetic Algorithms, Morgan Kaufmann, 1997, pp. 299–306.

[58] F. Oppacher, M. Wineberg, The Shifting Balance Genetic Algorithm: Improving the GA in a Dynamic Environment, in: W. Banzhalf (Ed.), Genetic and Evolutionary Computation Conference, Vol. 1, Morgan Kaufmann, 1999, pp. 504–510.

[59] W. Rand, R. Riolo, Measurements for understanding the behavior of the genetic algorithm in dynamic environments: A case study using the shaky ladder hyperplane-defined functions, in: S. Yang, J. Branke (Eds.), GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization, 2005.

[60] S. Yang, Genetic algorithms with memory- and elitism-based immigrants in dynamic environments, Evolutionary Computation 16 (3) (2008) 385–416.

[61] R. Morrison, K. De Jong, Measurement of population diversity, in: P. Collet, C. Fonlupt, J.-K. Hao, E. Lutton, M. Schoenauer (Eds.), Artificial Evolution, Vol. 2310 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2002, pp. 1047–1074.

[62] C.-K. Goh, K. C. Tan, A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization, IEEE Transactions on Evolutionary Computation 13 (1) (2009) 103–127.

[63] T. T. Nguyen, X. Yao, Continuous dynamic constrained optimisation - the challenges, IEEE Transactions on Evolutionary Computation (Accepted)[online].
URL `\url{http://www.staff.ljmu.ac.uk/enrtngu1/Papers/Nguyen_Yao_DCOP.pdf}`

[64] E. Alba, B. Sarasola, Measuring fitness degradation in dynamic optimization problems, in: Proceedings of the European Workshops on Applications of Evolutionary Computation, EvoApplicatons 2010, Part I, 2010, pp. 572–581.

[65] I. Hatzakis, D. Wallace, Dynamic multi-objective optimization with evolutionary algorithms: a forward-looking approach, in: GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation, ACM Press, New York, NY, USA, 2006, pp. 1201–1208.

[66] X. Li, J. Branke, M. Kirley, On performance metrics and particle swarm methods for dynamic multiobjective optimization problems, in: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007, 2007, pp. 576–583.

[67] C. Azevedo, A. Araujo, Generalized immigration schemes for dynamic evolutionary multiobjective optimization, in: Evolutionary Computation (CEC), 2011 IEEE Congress on, 2011, pp. 2033 –2040.

[68] M. Camara, J. Ortega, F. Toro, Parallel processing for multi-objective optimization in dynamic environments, in: Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, 2007, pp. 1 –8.

[69] E. Alba, J. Saucedo Badia, G. Luque, A Study of Canonical GAs for NSOPs, in: . Doerner et al (Ed.), Metaheuristics, Vol. 39 of Operations Research/Computer Science Interfaces Series, Springer US, 2007, pp. 245–260.

[70] X. Hu, R. Eberhart, Adaptive particle swarm optimisation: detection and response to dynamic systems, in: Proceedings of the IEEE Congress on Evolutionary Computation, CEC2002, 2002, pp. 1666–1670.

[71] X. Li, J. Branke, T. Blackwell, Particle swarm with speciation and adaptation in a dynamic environment, in: GECCO 06: Proceedings of the 8th annual conference on Genetic and evolutionary computation, ACM Press, New York, NY, USA, 2006, pp. 51–58.

[72] G. R. Kramer, J. C. Gallagher, Improvements to the *CGA enabling online intrinsic, in: Proceedings of the 2003 NASA DoD Conference on Evolvable Hardware, EH 03, IEEE Computer Society, Washington, DC, USA, 2003, pp. 235–231.

[73] X. Zou, M. Wang, A. Zhou, B. Mckay, Evolutionary optimization based on chaotic sequence in dynamic environments, in: 2004 IEEE International Conference on Networking, Sensing and Control, Vol. 2, 2004, pp. 1364 – 1369.

[74] T. T. Nguyen, X. Yao, Solving dynamic constrained optimisation problems using repair methods, IEEE Transactions on Evolutionary Computation (submitted).
URL `\url{http://www.staff.ljmu.ac.uk/enrtngu1/Papers/Nguyen_Yao_dRepairGA.pdf}`

[75] A. Carlisle, G. Dozier, Adapting particle swarm optimisationto dynamic environments, in: Proceedings of the International Conference on Artificial Intelligence, 2000, pp. 429–434.

[76] A. Carlisle, G. Dozier, Tracking changing extrema with adaptive particle swarm optimizer, in: Proc. World Automation Cong., Orlando FL USA, 2002, pp. 265–270.

[77] H. K. Singh, A. Isaacs, T. T. Nguyen, T. Ray, X. Yao, Performance of infeasibility driven evolutionary algorithm (IDEA) on constrained dynamic single objective optimization problems, in: Proceedings of the IEEE Congress on Evolutionary Computation CEC2009, IEEE Press, Trondheim, Norway,, 2009, pp. 3127–3134.

[78] I. Moser, T. Hendtlass, A simple and efficient multi-component algorithm for solving dynamic function optimisation problems, in: Proceedings of the IEEE Congress on Evolutionary Computation, 2007. CEC 2007., 2007, pp. 252–259.

[79] T. T. Nguyen, Tracking optima in dynamic environments using evolutionary algorithms - rsmg report 5, Tech. rep., School of Computer Science, University of Birmingham, [online] (2008).
URL \url{http://www.cs.bham.ac.uk/~txn/unpublished/reports/Report_5_Thanh.pdf}

[80] S. Janson, M. Middendorf, A hierarchical particle swarm optimizer for noisy and dynamic environments., Genetic Programming and Evolvable Machines 7 (4) (2006) 329–354.

[81] F. Vavak, K. Jukes, T. C. Fogarty, Learning the local search range for genetic optimisation in nonstationary environments, in: IEEE Intl. Conf. on Evolutionary Computation ICEC'97, IEEE Publishing, 1997, pp. 355–360.

[82] F. Vavak, K. A. Jukes, T. C. Fogarty, Performance of a genetic algorithm with variable local search range relative to frequency for the environmental changes, in: K. et al. (Ed.), International Conference on Genetic Programming, Morgan Kaufmann, 1998.

[83] F. Vavak, T. C. Fogarty, K. Jukes, A genetic algorithm with variable range of local search for tracking changing environments, in: H.-M. Voigt (Ed.), Parallel Problem Solving from Nature, no. 1141 in LNCS, Springer Verlag Berlin, 1996.

[84] M. Riekert, K. M. Malan, A. P. Engelbrecht, Adaptive genetic programming for dynamic classification problems, in: Proceedings of the Eleventh IEEE Congress on Evolutionary Computation, CEC'09, IEEE Press, Piscataway, NJ, USA, 2009, pp. 674–681.

[85] M. Daneshyari, G. Yen, Dynamic optimization using cultural based pso, in: Evolutionary Computation (CEC), 2011 IEEE Congress on, 2011, pp. 509 –516.

[86] D. Parrott, X. Li, Locating and tracking multiple dynamic optima by a particle swarm model using speciation., IEEE Trans. Evolutionary Computation 10 (4) (2006) 440–458.

[87] H. Richter, S. Yang, Learning behavior in abstract memory schemes for dynamic optimization problems, Soft Computing 13 (12) (2009) 1163–1173.

[88] A. Simões, E. Costa, Memory-based chc algorithms for the dynamic traveling salesman problem, in: Proceedings of the 13th annual conference on Genetic and evolutionary computation, GECCO '11, ACM, New York, NY, USA, 2011, pp. 1037–1044.

[89] H. C. Andersen, An investigation into genetic algorithms, and the relationship between speciation and the tracking of optima in dynamic functions, Honours thesis, Queensland University of Technology, Brisbane, Australia (Nov. 1991).

[90] N. Mori, H. Kita, Y. Nishikawa, Adaptation to a changing environment by means of the thermodynamical genetic algorithm, in: H.-M. Voigt (Ed.), Parallel Problem Solving from Nature, no. 1141 in LNCS, Springer Verlag Berlin, 1996, pp. 513–522.

[91] S. Yang, X. Yao, Experimental study on population-based incremental learning algorithms for dynamic optimization problems, Soft Computing - A Fusion of Foundations, Methodologies and Applications 9 (11) (2005) 815–834.

[92] S. Janson, M. Middendorf, A hierarchical particle swarm optimizer and its adaptive variant, IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics 35 (2005) 1272–1282.

[93] T. M. Blackwell, P. J. Bentley, Dynamic search with charged swarms, in: W. B. L. et al. (Ed.), Genetic and Evolutionary Computation Conference, Morgan Kaufmann, 2002, pp. 19–26.

[94] T. Blackwell, J. Branke, Multiswarms, exclusion, and anti-convergence in dynamic environments., IEEE Trans. Evolutionary Computation 10 (4) (2006) 459–472.

[95] T. Blackwell, Particle swarm optimization in dynamic environment, in: S. Yang, Y.-S. Ong, Y. Jin (Eds.), Evolutionary Computation in Dynamic and Uncertain Environments, Studies in Computational Intelligence, Springer-Verlag, NJ, USA, 2007, pp. 28–49.

[96] L. Bui, H. Abbass, J. Branke, Multiobjective optimization for dynamic environments, in: Congress on Evolutionary Computation, Vol. 3, IEEE press, 2005, pp. 2349 – 2356.

[97] H. A. Abbass, K. Deb, Searching under multi-evolutionary pressures., in: Proceedings of the Evolutionary Multi-Criterion Optimization, Second International Conference, EMO 2003, 2003, pp. 391–404.

[98] A. Toffolo, E. Benini, Genetic diversity as an objective in multi-objective evolutionary algorithms, Evolutionary Computation 11 (2) (2003) 151–167.

[99] Y. Wang, M. Wineberg, Estimation of evolvability genetic algorithm and dynamic environments, Genetic Programming and Evolvable Machines 7 (4) (2006) 355–382.

[100] L. Liu, D. Wang, S. Yang, Compound particle swarm optimization in dynamic environments, in: Proceedings of the 2008 conference on Applications of evolutionary computing, Evo'08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 616–625.

[101] H. Cheng, S. Yang, Multi-population genetic algorithms with immigrants scheme for dynamic shortest path routing problems in mobile ad hoc networks, in: . Di Chio et al (Ed.), Applications of Evolutionary Computation, Vol. 6024 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2010, pp. 562–571.

[102] F. O. de França, F. J. Von Zuben, A dynamic artificial immune algorithm applied to challenging benchmarking problems, in: Proceedings of the Eleventh IEEE Congress on Evolutionary Computation, CEC'09, IEEE Press, Piscataway, NJ, USA, 2009, pp. 423–430.

[103] K. Deb, U. B. Rao, S. Karthik, Dynamic multi-objective optimization and decision-making using modified NSGA-II: A case study on hydro-thermal power scheduling, in: Evolutionary Multi-Criterion Optimization, 4th International Conference, EMO 2007, Matsushima, Japan, March 5-8, 2007, Proceedings, Vol. 4403 of Lecture Notes in Computer Science, Springer, 2007, pp. 803–817.

[104] M. Gouvêa, Jr., A. Araújo, Adaptive evolutionary algorithm based on population dynamics for dynamic environments, in: Proceedings of the 13th annual conference on Genetic and evolutionary computation, GECCO '11, ACM, New York, NY, USA, 2011, pp. 909–916.

[105] W. Cedeno, V. R. Vemuri, On the use of niching for dynamic landscapes, in: International Conference on Evolutionary Computation, IEEE, 1997.

[106] J. Lewis, E. Hart, G. Ritchie, A comparison of dominance mechanisms and simple mutation on non-stationary problems, in: A. E. Eiben, T. Bäck, M. Schoenauer, H.-P. Schwefel (Eds.), Parallel Problem Solving from Nature, no. 1498 in LNCS, Springer, 1998, pp. 139–148.

[107] K. P. Ng, K. C. Wong, A new diploid scheme and dominance change mechanism for non-stationary function optimization, in: Sixth International Conference on Genetic Algorithms, Morgan Kaufmann, 1995, pp. 159–166.

[108] A. S. Uyar, A. E. Harmanci, A new population based adaptive domination change mechanism for diploid genetic algorithms in dynamic environments, Soft Computing - A Fusion of Foundations, Methodologies and Applications 9 (11) (2005) 803–814.

[109] S. Yang, On the design of diploid genetic algorithms for problem optimization in dynamic environments, in: Proceedings of the 2006 IEEE Congress on Evolutionary Computation. CEC 2006., 2006, pp. 1362 –1369.

[110] C. Ryan, The degree of oneness, in: First Online Workshop on Soft Computing, 1996, pp. 43–49.

[111] E. Collingwood, D. Corne, P. Ross, Useful diversity via multiploidy, in: Proceedings of IEEE International Conference on Evolutionary Computation, 1996, 1996, pp. 810 –813.

[112] C. N. Bendtsen, T. Krink, Dynamic memory model for non-stationary optimization, in: Congress on Evolutionary Computation, IEEE, 2002, pp. 145–150.

[113] S. J. Louis, Z. Xu, Genetic algorithms for open shop scheduling and rescheduling, in: M. E. Cohen, D. L. Hudson (Eds.), ISCA Eleventh International Conference on Computers and their Applications, 1996, pp. 99–102.

[114] N. Mori, H. Kita, Y. Nishikawa, Adaptation to a changing environment by means of the feedback thermodynamical genetic algorithm, in: A. E. Eiben, T. Bäck, M. Schoenauer, H.-P. Schwefel (Eds.), Parallel Problem Solving from Nature, no. 1498 in LNCS, Springer, 1998, pp. 149–158.

[115] S. Yang, Memory-based immigrants for genetic algorithms in dynamic environments, in: H.-G. Beyer, et al. (Eds.), Genetic and Evolutionary Computation Conference, ACM, 2005, pp. 1115–1122.

[116] S. Yang, Associative memory scheme for genetic algorithms in dynamic environments, in: F. Rothlauf, et al. (Eds.), Applications of Evolutionary Computing, Vol. 3907 of LNCS, Springer, 2006, pp. 788–799.

[117] E. L. Yu, P. N. Suganthan, Evolutionary programming with ensemble of explicit memories for dynamic optimization, in: Proceedings of the Eleventh IEEE Congress on Evolutionary Computation, CEC'09, IEEE Press, Piscataway, NJ, USA, 2009, pp. 431–438.

[118] S. Zeng, H. Shi, L. Kang, L. Ding, Orthogonal dynamic hill climbing algorithm: Odhc, in: S. Yang, Y.-S. Ong, Y. Jin (Eds.), Evolutionary Computation in Dynamic and Uncertain Environments, Studies in Computational Intelligence, Springer-Verlag New York, Inc., 2007, pp. 79–105.

[119] J. Lepagnot, A. Nakib, H. Oulhadj, P. Siarry, Brain cine mri segmentation based on a multiagent algorithm for dynamic continuous optimization, in:

Evolutionary Computation (CEC), 2011 IEEE Congress on, 2011, pp. 1695 –1702.

[120] M. Mavrovouniotis, S. Yang, Memory-based immigrants for ant colony optimization in changing environments, in: Proceedings of the 2011 international conference on Applications of evolutionary computation - Volume Part I, EvoApplications'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 324–333.

[121] J. Eggermont, T. Lenaerts, S. Poyhonen, A. Termier, Raising the dead; extending evolutionary algorithms with a case-based memory, in: J. F. Miller, et al. (Eds.), Genetic Programming, Proceedings of EuroGP'2001, Vol. 2038, Springer, 2001, pp. 280–290.

[122] C. L. Ramsey, J. J. Grefenstette, Case-based initialization of genetic algorithms, in: S. Forrest (Ed.), International Conference on Genetic Algorithms, Morgan Kaufmann, 1993, pp. 84–91.

[123] A. Simes, E. Costa, Evolutionary algorithms for dynamic environments: Prediction using linear regression and markov chains, in: G. Rudolph, T. Jansen, S. Lucas, C. Poloni, N. Beume (Eds.), Parallel Problem Solving from Nature PPSN X, Vol. 5199 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2008, pp. 306–315.

[124] H. Richter, S. Yang, Memory based on abstraction for dynamic fitness functions, in: . Mario Giacobini et al (Ed.), Applications of Evolutionary Computing, Vol. 4974 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2008, pp. 596–605.

[125] A. Simões, E. Costa, An immune system-based genetic algorithm to deal with dynamic environments: Diversity and memory, in: D. W. Pearson, N. C. Steele, R. Albrecht (Eds.), Proceedings of the Sixth international conference on neural networks and genetic algorithms (ICANNGA03), Springer, 2003, pp. 168–174.

[126] S. Yang, A comparative study of immune system based genetic algorithms in dynamic environments, in: GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation, ACM Press, New York, NY, USA, 2006, pp. 1377–1384.

[127] A. Simões, E. Costa, Improving memory's usage in evolutionary algorithms for changing environments, in: Proceedings of the 2007 IEEE Congress on Evolutionary Computation, CEC'07, 2007, pp. 276–283.

[128] T. Zhu, W. Luo, Z. Li, An adaptive strategy for updating the memory in evolutionary algorithms for dynamic optimization, in: Computational Intelligence in Dynamic and Uncertain Environments (CIDUE), 2011 IEEE Symposium on, 2011, pp. 8 –15.

[129] J. Branke, Evolutionary approaches to dynamic optimization problems – introduction and recent trends, in: J. Branke (Ed.), GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems, 2003, pp. 2–4.

[130] C. Rossi, M. Abderrahim, J. C. Díaz, Tracking moving optima using kalman-based predictions, Evolutionary Computation 16 (1) (2008) 1–30.

[131] A. Simões, E. Costa, Improving prediction in evolutionary algorithms for dynamic environments, in: . Raidl et al (Ed.), GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, ACM, Montreal, Québec, Canada, 2009, pp. 875–882.

[132] P. A. N. Bosman, H. L. Poutré, Learning and anticipation in online dynamic optimization with evolutionary algorithms: the stochastic case, in: GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, ACM, New York, NY, USA, 2007, pp. 1165–1172.

[133] J. Branke, D. Mattfeld, Anticipation and flexibility in dynamic scheduling, International Journal of Production Research 43 (15) (2005) 3103–3129.

[134] R. K. Ursem, Multinational GA optimization techniques in dynamic environments, in: D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, H.-G. Beyer (Eds.), Genetic and Evolutionary Computation Conference, Morgan Kaufmann, 2000, pp. 19–26.

[135] P. J. Angeline, Tracking extrema in dynamic environments, in: P. J. Angeline, R. G. Reynolds, J. R. McDonnell, R. Eberhart (Eds.), Sixth International Conference on Evolutionary Programming, Vol. 1213 of LNCS, Springer, 1997, pp. 335–345.

[136] D. V. Arnold, H.-G. Beyer, Random Dynamics Optimum Tracking with Evolution Strategies, in: J. Merelo, P. Adamidis, H.-G. Beyer, J. Fernández-Villacañas, H.-P. Schwefel (Eds.), Parallel Problem Solving from Nature, Springer, Heidelberg, 2002, pp. 3–12.

[137] D. V. Arnold, H.-G. Beyer, Optimum tracking with evolution strategies, Evolutionary Computation 14 (3) (2006) 291–308.

[138] J. Branke, T. Kaußler, C. Schmidt, H. Schmeck, A multi-population approach to dynamic optimization problems, in: Adaptive Computing in Design and Manufacturing 2000, Springer, 2000.

[139] R. I. Lung, D. Dumitrescu, A new collaborative evolutionary-swarm optimization technique, in: GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation, ACM, New York, NY, USA, 2007, pp. 2817–2820.

[140] R. Mendes, A. Mohais, Dynde: a differential evolution for dynamic optimization problems, in: Proceedings of the 2005 IEEE Congress on Evolutionary Computation, IEEE, 2005, pp. 2808–2815.

[141] J. L. Fernández, J. L. Arcos, Adapting particle swarm optimization in dynamic and noisy environments, in: Proceedings of the 2010 IEEE Congress on Evolutionary Computation CEC'2010, 2010, pp. 765–772.

[142] C. Li, S. Yang, A clustering particle swarm optimizer for dynamic optimization, in: Proceedings of the Eleventh IEEE Congress on Evolutionary Computation, CEC'09, IEEE Press, Piscataway, NJ, USA, 2009, pp. 439–446.

[143] S. Yang, C. Li, A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments, IEEE Trans. Evolutionary Computation 14 (6) (2010) 959–974.

[144] S. Tsutsui, Y. Fujimoto, A. Ghosh, Forking genetic algorithms: Gas with search space division schemes., Evolutionary Computation 5 (1) (1997) 61–80.

[145] I. Moser, Review - all currently known publications on approaches which solve the moving peaks problem, Tech. rep., Swinburne University of Technology, Melbourne, Australia (2007).

[146] T. Jansen, C. Zarges, Analysis of evolutionary algorithms: from computational complexity analysis to algorithm engineering, in: Proceedings of the 11th Workshop on Foundations of Genetic Algorithms, 2005, pp. 1–14.

[147] A. Nix, M. Vose, Modelling genetic algorithms with markov chains, Annals of Mathematics and Artificial Intelligence 5 (1) (1998) 79–88.

[148] H. Mühlenbein, The equation for response to selection and its use for prediction, Evolutionary Computation 5 (3) (1998) 303–346.

[149] J. He, X. Yao, Drift analysis and average time complexity of evolutionary algorithms, Artificial Intelligence 127 (1) (2001) 57–85.

[150] J. He, X. Yao, From an individual to a population: An analysis of the first hitting time of population-based evolutionary algorithms, IEEE Transactions on Evolutionary Computation 6 (5) (2002) 495–511.

[151] S. A. Stanhope, J. M. Daida, Genetic algorithm fitness dynamics in a changing environment, in: Congress on Evolutionary Computation, Vol. 3, IEEE, 1999, pp. 1851–1858.

[152] J. Branke, W. Wang, Theoretical analysis of simple evolution strategies in quickly changing environments, in: Proceedings of the 2003 Int Conf on Genetic and Evolutionary Computation, 2003, pp. 537–548.

[153] S. Droste, Analysis of the (1+1) ea for a dynamically changing onemax-variant, in: Congress on Evolutionary Computation, IEEE Press, 2002, pp. 55–60.

[154] T. Jansen, U. Schellbach, Theoretical analysis of a mutation-based evolutionary algorithm for a tracking problem in lattice, in: H.-G. Beyer, et al. (Eds.), Genetic and Evolutionary Computation Conference, ACM, 2005, pp. 841–848.

[155] K. Weicker, Analysis of local operators applied to discrete tracking problems, Soft Computing - A Fusion of Foundations,Methodologies and Applications 9 (11) (2005) 778–792.

[156] P. Rohlfshagen, P. K. Lehre, X. Yao, Dynamic evolutionary optimisation: An analysis of frequency and magnitude of change, in: Proceedings of the 2009 Genetic and Evolutionary Computation Conference GECCO'09, 2009, pp. 1713–1720.

[157] J. Branke, E. Salihoglu, S. Uyar, Towards an analysis of dynamic environments, in: H.-G. Beyer, et al. (Eds.), Genetic and Evolutionary Computation Conference, ACM, 2005, pp. 1433–1439.

[158] J. Branke, M. Orbayi, S. Uyar, The role of representations in dynamic knapsack problems, in: F. Rothlauf, et al. (Eds.), Applications of Evolutionary Computing, Vol. 3907 of LNCS, Springer, 2006, pp. 764–775.

[159] R. Tinos, S. Yang, An analysis of the XOR dynamic problem generator based on the dynamical system, in: Proceedings of the Parallel Problems Solving from Nature (PPSN XI), 2010.

[160] M. D. Vose, The Simple Genetic Algorithm: Foundations and Theory, The MIT Press, 1999.

[161] H. Richter, Behavior of evolutionary algorithms in chaotically changing fitness landscapes, in: X. Yao, et al. (Eds.), Parallel Problem Solving from Nature, Vol. 3242 of LNCS, Springer, 2004, pp. 111–120.

[162] H. Richter, Evolutionary optimization in spatiotemporal fitness landscapes, in: Parallel Problem Solving from Nature, Springer, 2006, pp. 1–10.

[163] J. Chazottes, B. Fernandez, Dynamics of Coupled Map Lattices and of Related Spatially Extended Systems, Springer, Heidelberg, 2005.

[164] D. H. Wolpert, W. G. Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation 1 (1) (1997) 67–82.

[165] R. K. Ursem, T. Krink, M. T. Jensen, Z. Michalewicz, Analysis and modeling of control tasks in dynamic systems, IEEE Transactions on Evolutionary Computation 6 (4) (2002) 378–389.

[166] J. He, X. Yao, A study of drift analysis for estimating computation time of evolutionary algorithms, Natural Computing 3 (1) (2004) 21–35.

[167] H. Cheng, S. Yang, Genetic algorithms with immigrants schemes for dynamic multicast problems in mobile ad hoc networks, Engineering Applications of Artificial Intelligence 23 (5) (2010) 806–819.

[168] D. M. Chitty, M. L. Hernandez, A hybrid ant colony optimization technique for dynamic vehicle routing, in: Proceedings of the 2005 International Conference on Genetic and Evolutionary Computation, 2004, pp. 48–59.

[169] L. Xing, P. Rohlfshagen, Y. Chen, X. Yao, A hybrid ant colony optimisation algorithm for the extended capacitated arc routing problem, IEEE Transactions on Systems, Man and Cybernetics, Part B. Cybernetics 41 (4) (2011) 1110 – 1123.