Chapter 1
Evolutionary Dynamic Optimization: Test and Evaluation Environments

Shengxiang Yang, Trung Thanh Nguyen, and Changhe Li

Abstract. In the last two decades, dynamic optimization problems (DOPs) have drawn a lot of research studies from the evolutionary computation (EC) community due to the importance in real-world applications. A variety of evolutionary computation approaches have been developed to address DOPs. In parallel with developing new approaches, many benchmark and real-world DOPs have been constructed and used to compare them under different performance measures. In this chapter, we describe the concept of DOPs and review existing dynamic test problems that are commonly used by researchers to investigate their EC approaches in the literature. Some discussions regarding the major features of existing dynamic test environments are presented. Typical dynamic benchmark problems and real-world DOPs are described in detail. We also review the performance measures that are widely used by researchers to evaluate and compare their developed EC approaches for DOPs. Suggestions are also given for potential improvement regarding dynamic test and evaluation environments for the EC community.

1.1 Introduction

In the last two decades, dynamic optimization problems (DOPs) have drawn a lot of research studies from the evolutionary computation (EC) community. Especially, in

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

Trung Thanh Nguyen
School of Engineering, Technology and Maritime Operations, Liverpool John Moores University, Liverpool L3 3AF, U.K.
e-mail: T.T.Nguyen@ljmu.ac.uk

Changhe Li
School of Computer Science, China University of Geosciences, 388 Lumo Road, Wuhan 430074, China
e-mail: changhe.lw@gmail.com

S. Yang and X. Yao (Eds.): *Evolutionary Computation for DOPs*, SCI 490, pp. 3–37.
DOI: 10.1007/978-3-642-38416-5_1

recent years, there has been a growing interest in studying evolutionary algorithms (EAs) for DOPs due to its importance in real-world applications since many real-world optimization problems are DOPs. The research domain of EC for DOPs can be termed as evolutionary dynamic optimization (EDO). DOPs require EAs to track the trajectory of changing optima in the search space [11, 30]. This poses great challenges to traditional EAs due to the convergence problem: once converged, they can not track the changing optima well. Hence, researchers have developed several approaches into EAs to enhance their performance for DOPs [30, 83], e.g., diversity schemes [19, 27, 80], memory schemes [9, 70, 88], multi-population schemes [13, 55, 82], prediction and anticipation schemes [64], and adaptive schemes [45, 84, 85].

In order to study and compare the developed EA approaches for DOPs, there are two important tasks. One important task is to build up proper dynamic test environments. Over the years, in parallel with developing EA approaches for DOPs, researchers have also developed many dynamic benchmark problems, e.g., the moving peaks benchmark (MPB) problem by Branke [9], the XOR DOP generator by Yang and Yao [77, 87, 88], the generalized dynamic benchmark generator (GDBG) by Li and Yang [36], and modelled a number of real-world DOPs, e.g., dynamic knapsack problems [34, 43, 49], dynamic travelling salesman problems [35, 40], dynamic routing problems in communication networks [16, 17, 81], and dynamic vehicle routing problems in logistics and transportation networks [42, 75]. The other important task is to define proper performance measures to compare different EC approaches for DOPs. Over the years, researchers have developed a number of different performance measures to evaluate the developed EA approaches for DOPs, e.g., the offline error measure [9], the accuracy measure [72], and the best-of-generation measure [18], etc.

In this chapter, we present the concept of DOPs and review existing dynamic test problems commonly used by researchers to investigate their EC approaches in the literature. Some discussions regarding the major features and classification of existing dynamic test environments are presented. Some typical dynamic benchmark problems and real-world DOPs, which cover the binary, real, and combinatorial spaces, are also described in detail. We also review the performance measures that are widely used by researchers to evaluate and compare their developed EC approaches for DOPs. Suggestions are also given for potential improvement regarding dynamic test and evaluation environments for the EC community.

The rest of this chapter is organized as follows. The next section first introduces the concept of DOPs, then historically reviews dynamic test problems in the literature, and finally describes the major features and classification of existing dynamic test problems. Section 1.3 describes in detail some dynamic test problems and generators that are commonly used in the literature, covering the binary space, the real space, and the combinatorial space. Section 1.4 reviews the typical performance measures that are used by researchers to compare and justify their algorithms for DOPs. Section 1.5 presents the GDBG system. Finally, Section 1.6 concludes this chapter with some discussions on the future work on constructing dynamic test and evaluation environments for the EC community.

1.2 DOPs: Concepts, Brief Review, and Classification

1.2.1 Concepts of DOPs

In the literature of EC in dynamic environments, researchers usually define optimization problems that change over time as *dynamic problems* or *time-dependent problems*. In this chapter, we define *DOPs* as *a special class of dynamic problems that are solved online by an optimization algorithm as time goes by*.

It is notable that in many EDO studies, the terms "dynamic problems/time-dependent problems" and "DOPs" are not distinguished or are used interchangeably. In these studies, DOPs are either defined as a sequence of static problems linked up by some dynamic rules [4, 61, 62, 71, 73] or as a problem that has time-dependent parameters in its mathematical expression [5, 7, 20, 76], without explicitly mentioning whether the problems are solved online by an optimization algorithm or not. 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 [10, 30, 48]. Hence, only DOPs are relevant to EDO research.

1.2.2 Dynamic Test Problems: Brief Review

In order to compare the performance of the developed GA approaches in dynamic environments, researchers have developed a number of dynamic problem generators. Generally speaking, DOPs are constructed via changing (the parameters of) stationary base problem(s). And, ideally, through proper control, different dynamic environments can be constructed from the stationary base problem(s) regarding the characteristics of the environmental dynamics, such as the frequency, severity, predictability, and cyclicity of environmental changes. Below, we briefly review the dynamic test environments that have been used/developed by researchers to test their EC approaches roughly in the time order.

In the early days, the dynamic test environments were quite simple: the environment is just switched between two or more stationary problems or between two or more states of one problem. For example, Cobb and Grefenstette [19] used a dynamic environment that oscillates between two different fitness landscapes. The dynamic 0-1 knapsack problem where the knapsack capacity oscillates between two or more fixed values has been frequently used in the literature [34, 43, 49]. The dynamic bit-matching problem has also been used by researchers for analyzing the performance of EC approaches in dynamic environments [65].

Later in 1999, several researchers have independently developed several dynamic environment generators by changing a base fitness landscape predefined in the multi-dimensional real space [9, 28, 44, 67]. This base fitness landscape consists

of a number of peaks. Each peak can change its own morphology independently, such as the height, slope, and location of the peak. The center of the peak with the highest height is taken as the optimal solution of the landscape. Dynamic problems can be created through changing the parameters of each peak.

More recently, the dynamic 0-1 knapsack problem has been extended to dynamic multi-dimensional knapsack problems in several studies [14, 68]. In [78], Yang proposed a dynamic environment generator based on the concept of problem difficulty and unitation and trap functions. An XOR DOP generator, which can generate dynamic environments from any binary encoded stationary problem based on a bitwise exclusive-or (XOR) operator, has been proposed in [77, 87, 88]. In [57, 58, 60], Richter constructed spatio-temporal fitness landscapes based on coupled map lattices (CML) [15], where such properties as modality, ruggedness, information content, epistasis, dynamic severity, and Lyapunov exponents can be defined. Bosman [7] and Nguyen and Yao [52] investigated the online DOPs that have the time-linkage property, i.e., the current solution found by an optimization algorithm affects the future behaviour of the problem. In [53], Nguyen and Yao investigated dynamic constrained optimization problems where constraints change over time. The authors extended this study to provide a full set of dynamic constraine test problems in [50, 51]. In order to develop a unified approach of constructing dynamic problems across the binary space, the real space, and the combinatorial space, the GDBG system was recently proposed in [36, 37], which can be instantiated to construct dynamic test environments for all the three solution spaces.

In recent years, researchers have also studied a number of real-world DOPs. For example, Li *et al.* [35] studied the dynamic travelling salesman problem where the cities may change their locations. Mavrovouniotis and Yang investigated the dynamic travelling salesman problem where cities may join or leave the topology over time [40] and the traffic may change over time [41]. In [16, 17, 81], Cheng *et al.* studied the dynamic shortest path routing and dynamic multi-cast routing problems in mobile ad hoc networks (MANETs). In [42, 75], dynamic vehicle routing problems in logistics and transportation networks have been investigated by the EC community.

1.2.3 Major Characteristics and Classification of DOPs

As briefly reviewed above, many dynamic test problems have been used in the literature. These dynamic test problems have different characteristics and can be classified into different groups based on the following different criteria:

- Time-linkage: Whether the future behaviour of the problem depends on the current solution found by an algorithm or not.
- Predictability: Whether the generated changes are predictable or not.
- Visibility: Whether the changes are visible to the optimisation algorithm and, if so, whether changes can be detected by using just a few detectors.

- Constrained problem: Whether the problem is constrained or not.
- Number of objectives: Whether the problem is single objective or multiple objectives.
- Type of changes: Detailed explanation of how changes occur in the search space.
- Cyclicity: Whether the changes are cyclic/recurrent in the search space or not.
- Periodicity: Whether the changes are periodical or not in time.
- Factors that change: Changes may involve parameters of objective functions, domain of variables, number of variables, constraints, and other parameters.

The common characteristics of the general-purpose dynamic benchmark problems used in the literature are summarized as follows.

- Most dynamic test problems are non time-linkage problems. There are only a couple of general-purpose time-linkage test problems [7, 52] and some problem-specific time-linkage test problems [7, 8].
- Most of the dynamic test generators/problems in the continuous domain are unconstrained or domain constrained, except the two most recent studies [53, 59]
- In the default settings of most general-purpose dynamic test problems, changes are detectable by using just a few detectors. Exceptions are some problem instances in [19, 67] where only one or some peaks move, and in [53, 59] 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 one instance in [36] where the dimension also changes and the problems in [53, 59] where the constraints also change.
- Many generators/problems have unpredictable changes in their default settings. 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 cyclic/recurrent changes.
- Most benchmark generators/problems assume periodical changes, i.e., changes occur every fixed number of generations or fitness evaluations. An exception is the work in [64], which also studies the cases where the changes occur in some time pattern.
- Most generators/problems are single-objective. Only a few studies involve dynamic multi-objective problems, e.g., [23, 31].

In the next section, we describe in details some generators/problems that are commonly used in the domain of EC for DOPs in the real space, binary space, and combinatorial space, respectively.

1.3 Typical Dynamic Test Problems and Generators

1.3.1 Dynamic Test Problems in the Real Space

1.3.1.1 The DF1 Generator

The dynamic problem generator, called the DF1 generator, proposed by Morrison and De Jong [44], is a kind of moving peaks benchmark generators. Within DF1, the base landscape in the D-dimensional real space is defined as:

$$f(\mathbf{x}) = \max_{i=1,\ldots,p} \left[H_i - R_i \times \sqrt{\sum_{j=1}^{D} (x_j - X_{ij})^2} \right] \tag{1.1}$$

where $\mathbf{x} = (x_1, \cdots, x_D)$ is a point in the landscape, p specifies the number of peaks (or cones), and each peak i is independently specified by its height H_i, its slope R_i, and its center $X_i = (X_{i1}, \cdots, X_{iD})$. These peaks are blended together by the *max* function. The fitness at a point on the surface is assigned the maximum height of all optima at that point; the optima with the greatest height at a point is said to be visible at that point.

DF1 creates dynamic problems by changing the features, i.e., the location, height, and slope, of each peak independently. The dynamics are controlled by the Logistics function given by:

$$\Delta_t = A \cdot \Delta_{t-1} \cdot (1 - \Delta_{t-1}) \tag{1.2}$$

where A is a constant value in the range [1.0, 4.0] and Δ_t is used as the step size of changing a particular parameter (i.e., the location, height, or slope) of peaks at iteration t after scaled by a scale factor s in order to reduce step sizes that may be larger than intended for each step.

The logistics function allows a wide range of dynamic performance by a simple change of the value of A, from simple constant step sizes, to step sizes that alternate between two values, to step sizes that rotate through several values, to completely chaotic step sizes. More details on the DF1 generator can be found in [44].

1.3.1.2 The Moving Peaks Benchmark (MPB) Problem

Branke [9] proposed the MPB problem, which has been widely used as dynamic benchmark problems in the literature. Similar to the DF1 generator described above, the MPB problem consists of a multi-dimensional fitness landscape in the real space with a number of peaks, where each peak has three features, i.e., the height, width, and central position. Within the MPB problem, the optima can be changed by changing the three features of each peak independently or in a correlative way.

For the D-dimensional landscape, the MPB problem is defined as follows:

$$F(\mathbf{x}, t) = \max_{i=1,\ldots,p} \frac{H_i(t)}{1 + W_i(t) \sum_{j=1}^{D} (x_j(t) - X_{ij}(t))^2}, \tag{1.3}$$

Table 1.1 Default settings for the MPB problem

Parameter	Value
p (the number of peaks)	10
U (change frequency)	5000
height severity	7.0
width severity	1.0
peak shape	cone
basic function	no
s (the shift length)	1.0
D (the number of dimensions)	5
λ (the correlation coefficient)	0
S (the range of allele values)	[0, 100]
H (the range of the height of peaks)	[30.0, 70.0]
W (the range of the width of peaks)	[1, 12]
I (the initial height for all peaks)	50.0

where $W_i(t)$ and $H_i(t)$ are the height and width of peak i at time t, respectively, and $X_{ij}(t)$ is the j-th element of the location of peak i at time t. The p independently specified peaks are blended together by the *max* function. The position of each peak is shifted in a random direction by a vector $\mathbf{v_i}$ of a distance s (s is also called the shift length, which determines the severity of the problem dynamics), and the move of a single peak can be described as follows:

$$\mathbf{v}_i(t) = \frac{s}{|\mathbf{r} + \mathbf{v}_i(t-1)|}((1-\lambda)\mathbf{r} + \lambda\mathbf{v}_i(t-1)), \tag{1.4}$$

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

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

$$H_i(t) = H_i(t-1) + height\_severity * \sigma \tag{1.5}$$

$$W_i(t) = W_i(t-1) + width\_severity * \sigma \tag{1.6}$$

$$\mathbf{X}_i(t) = \mathbf{X}_i(t)(t-1) + \mathbf{v}_i(t) \tag{1.7}$$

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

The default settings for the MPB benchmark typically used in the literature can be found in Table 1.1, which are corresponding to Scenario 2 in [9]. In Table 1.1, the change frequency (U) means that environment changes every U fitness evaluations.

1.3.2 Dynamic Test Problems in the Binary Space

1.3.2.1 The Dynamic Bit-Matching Problem

The dynamic bit-matching problem has been used by researchers for the analysis of the performance of EAs in dynamic environments [21, 65]. For example, Stanhope and Daida [65] analyzed the behaviour of a simple (1+1) EA based on the dynamic bit-matching problem. In the dynamic bit-matching problem, an algorithm needs to find solutions that minimize the Hamming distance to an arbitrary target pattern (i.e., the match string) that may change over time. Given a solution $\mathbf{x} \in \{0,1\}^L$ (L is the length of the binary encoding of a solution for the problem) and the target pattern $\mathbf{a} \in \{0,1\}^L$ at time t, the Hamming distance between them is calculated as:

$$d_{Hamming}(\mathbf{x}(t), \mathbf{a}(t)) = \sum_{i=1}^{i=L} |x_i(t) - a_i(t)| \tag{1.8}$$

The dynamics of the problem is controlled by two parameters, g and d, which control the number of generations between changes and the degree (Hamming distance) by which the target pattern $\mathbf{a}$ is altered (for each change, d distinct and randomly chosen bits in $\mathbf{a}$ are inverted), respectively. For example, setting $(g,d) = (0,0)$ results in a stationary function whereas setting $(g,d) = (10,5)$ means that every 10 generations, the target pattern $\mathbf{a}$ changes by 5 bits randomly.

1.3.2.2 Dynamic Knapsack Problems (DKP) and Dynamic Multi-dimensional Knapsack Problems (DMKP)

The knapsack problem [32] is a classical NP-hard combinatorial optimization problem, where the solution space belongs to the binary space. Given a set of items, each of which has a weight and a profit, and a knapsack with a fixed capacity, the problem aims to select items to fill up the knapsack to maximize the total profit while satisfying the capacity constraint of the knapsack. Suppose there are n items, and $\mathbf{w}$, $\mathbf{p}$, and C, denote the weights of items, the profits of items, and the capacity of the knapsack, respectively. Then, the knapsack problem can be defined as follows:

$$Max\ f(\mathbf{x}) = \sum_{i=1}^{n} p_i \cdot x_i \tag{1.9}$$

$$subject\ to: \quad \sum_{i=1}^{n} w_i \cdot x_i \leq C, \tag{1.10}$$

where $\mathbf{x} \in \{0,1\}^n$ is a solution, $x_i \in \{0,1\}$ indicates whether item i is included in the subset or not, p_i is the profit of item i, and w_i is the weight of item i.

The above knapsack problem has been frequently used to test the performance of EAs in stationary environments, and its dynamic version has also been used by researchers to test the performance of EAs in dynamic environments [34, 43, 49]. In

the dynamic knapsack problem, the system dynamics can be constructed by changing the weights of items, the profits of items, and/or the knapsack capacity over time according to some dynamics, respectively. So, the dynamic knapsack problem can be described as follows:

$$Max\ f(\mathbf{x},t) = \sum_{i=1}^{n} p_i(t) \cdot x_i \tag{1.11}$$

$$subject\ to: \quad \sum_{i=1}^{n} w_i(t) \cdot x_i \leq C(t), \tag{1.12}$$

where the weight and profit of each item may be bounded in the range of $[l_w, u_w]$, $[l_p, u_p]$, and the capacity of knapsack may be bounded in the range of $[l_c, u_c]$.

Similarly, the static multi-dimensional knapsack problem (MKP) belongs to the class of NP-complete problems, which has a wide range of real-world applications, such as cargo loading, selecting projects to fund, budget management, etc. In the MKP, we have a number of resources (knapsacks), each of which has a capacity, and a set of items, each of which has a profit and consumes some amount of each resource. The aim is to select items to maximize the total profit while satisfying the capacity constraints for all resources.

The DMKP has recently been used to investigate the performance of EAs for DOPs [14, 68]. As in the DKP, in the DMKP, the profit and resource consumption for each item as well as the capacity of each resource may change over time. Let **r**, **p**, **c** denote the resource consumptions of items, the profits of items, and the capacities of resources, respectively. Then, the DMKP can be defined as follows:

$$Max\ f(\mathbf{x},t) = \sum_{i=1}^{n} p_i(t) \cdot x_i \tag{1.13}$$

$$subject\ to: \quad \sum_{i=1}^{n} r_{ij}(t) \cdot x_i \leq c_i(t), j = 1, 2, \cdots, m \tag{1.14}$$

where n is the number of items, m is the number of resources, x_i and p_i are as defined above, $r_{ij}(t)$ denotes the resource consumption of item i for resource j at time t, and $c_i(t)$ is the capacity constraint of resource i at time t. The system dynamics can be constructed by changing the profits of items, resource consumptions of items, and the capacity constraints of resources within certain upper and lower bounds over time according to some dynamics, respectively.

1.3.2.3 The XOR DOP Generator

In [77, 87], an XOR DOP generator that can generate dynamic environments from any binary encoded stationary problem using a bitwise exclusive-or (XOR) operator has been proposed. Given a stationary problem $f(\mathbf{x})$ ($\mathbf{x} \in \{0,1\}^l$ where l is the length of binary encoding), DOPs can be constructed from it as follows. Suppose

the environment is changed every τ generations. For each environmental period k, an XORing mask $\mathbf{M}(k)$ is first incrementally generated as follows:

$$\mathbf{M}(k) = \mathbf{M}(k-1) \oplus \mathbf{T}(k), \tag{1.15}$$

where "$\oplus$" is the XOR operator (i.e., $1 \oplus 1 = 0$, $1 \oplus 0 = 1$, $0 \oplus 0 = 0$) and $\mathbf{T}(k)$ is an intermediate binary template randomly created with $\rho \times l$ ($\rho \in [0.0, 1.0]$) ones inside it for environmental period k. Initially, $\mathbf{M}(0)$ is set to a zero vector. Then, the individuals at generation t are evaluated using the following formula:

$$f(\mathbf{x}, t) = f(\mathbf{x} \oplus \mathbf{M}(k)), \tag{1.16}$$

where $k = \lfloor t/\tau \rfloor$ is the environmental period index.

With this XOR DOP generator, the environmental dynamics can be tuned by two parameters: τ controls the speed of environmental changes while ρ controls the severity of changes. The bigger the value of τ, the slower the environment changes. The bigger the value of ρ, the more severe the environment changes.

The aforementioned XOR DOP generator in fact can construct *random dynamic environments* because there is no guarantee that the environment will return to a previous one after certain changes. In order to test the performance of memory based EAs, the XOR generator has been extended to construct *cyclic dynamic environments* in [79] and *cyclic dynamic environments with noise* further in [88].

With the XOR generator, cyclic dynamic environments can be constructed as follows. First, we can generate $2K$ XOR masks $\mathbf{M}(0), \cdots, \mathbf{M}(2K-1)$ as the *base states* in the search space randomly or in a certain patter. Then, the environment can cycle among these base states in a fixed logical ring. Suppose the environment changes every τ generations, then an individual at generation t is evaluated as follows:

$$f(\mathbf{x}, t) = f(\mathbf{x} \oplus \mathbf{M}(I_t)) = f(\mathbf{x} \oplus \mathbf{M}(k\%(2K))), \tag{1.17}$$

where $k = \lfloor t/\tau \rfloor$ is the index of current environmental period and $I_t = k\%(2K)$ is the index of the base state that the environment is in at generation t.

The $2K$ XOR masks can be generated in the following way. First, we construct K binary templates $\mathbf{T}(0), \cdots, \mathbf{T}(K-1)$ that form a random partition of the search space with each template containing $\rho \times l = l/K$ bits of ones[1]. Let $\mathbf{M}(0) = \mathbf{0}$ denote the initial state. Then, the other XOR masks are generated iteratively as follows:

$$\mathbf{M}(i+1) = \mathbf{M}(i) \oplus \mathbf{T}(i\%K), i = 0, \cdots, 2K-1 \tag{1.18}$$

So, the templates $\mathbf{T}(0), \cdots, \mathbf{T}(K-1)$ are first used to create K masks till $\mathbf{M}(K) = \mathbf{1}$ and then orderly reused to construct another K masks till $\mathbf{M}(2K) = \mathbf{M}(0) = \mathbf{0}$. The Hamming distance between two neighbour XOR masks is the same and equals $\rho \times l$. Here, $\rho \in [1/l, 1.0]$ is the distance factor, determining the number of base states.

[1] In the partition each template $\mathbf{T}(i)$ ($i = 0, \cdots, K-1$) has randomly but exclusively selected $\rho \times l$ bits set to 1 while other bits set to 0. For example, $\mathbf{T}(0) = 0101$ and $\mathbf{T}(1) = 1010$ form a partition of the 4-bit search space.

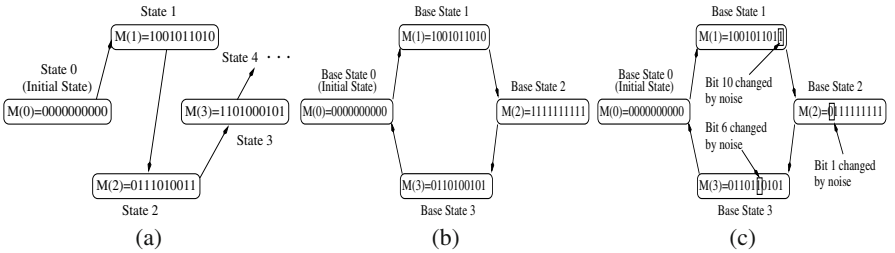

Fig. 1.1 Illustration of three kinds of dynamic environments constructed from a 10-bit encoded function with $\rho = 0.5$: (a) random, (b) cyclic, and (c) cyclic with noise

From the above cyclic environment generator, we can further construct cyclic dynamic environments with noise as below. Each time the environment is about to move to a next base state $\mathbf{M}(i)$, $\mathbf{M}(i)$ is bitwise flipped with a small probability p_n. Figure 1.1 illustrates the construction of random, cyclic, and cyclic with noise dynamic environments respectively from a 10-bit function with $\rho = 0.5$, where the XORing mask is used to represent the environmental state.

This XOR DOP generator has two properties. One is that the distances among the solutions in the search space remains unaltered after an environmental change. The other is that the properties of the fitness landscape are not changed after an environmental change, which facilitates the analysis of the behavior of algorithms. Recently, the XOR DOP generator has been extended to construct dynamic problems in the real space [66]. In [66], two continuous dynamic problem generators were proposed using the linear transformation of individuals. The first generator does the linear transformation by changing the direction of some axes of the search space while the second one uses successive rotations in different planes.

1.3.3 Dynamic Test Problems in the Combinatorial Space

1.3.3.1 The Spatio-Temporal Fitness Landscapes

In [57, 58], 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 [15]. In [59], 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.

In order to build spatio-temporal landscape based on CML, Richter defined a lattice grid with $I \times J$ cells in 2-dimension. A height at time t is assigned to each

cell referred to $h(i,j,t)$, where i and j denote the indices of cells. The change of the height of a cell is as follows:

$$\begin{aligned} h(i,j,t) &= (1-\varepsilon)g(h(i,j,t)) + \tfrac{\varepsilon}{4}[g(h(i-1,j,t)) \\ &\quad + g(h(i+1,j,t)) + g(h(i,j-1,t)) + g(h(i,j+1,t))] \end{aligned} \tag{1.19}$$

where ε is the diffusion coupling strength and $g(h(i,j,t))$ is logistic mapping function by:

$$g(h(i,j,t)) = \alpha h(i,j,t)(1-h(i,j,t)) \tag{1.20}$$

and the period boundary conditions is rendered by:

$$h(I+1,j,t) = h(1,j,t), h(i,J+1,t) = h(i,1,t) \tag{1.21}$$

To convert this integer system into real-valued fitness landscape, scaling factors s_1 and $s_2 \in \mathbb{R}$ are employed for the vertical (i) and horizontal extension (j) so that the search space variable $x = (x_1, x_2)^T$ is obtained by a rounding condition: $(i,j)^T = (\lceil s_1 x_1 \rceil, \lceil s_2 x_2 \rceil)^T$. Finally, the spatio-temporal fitness landscape in 2-D is produced by:

$$f(x,t) = \left\{ \begin{array}{cc} h(\lceil s_1 x_1 \rceil, \lceil s_2 x_2 \rceil, t) \; for & \begin{array}{c} 1 \le \lceil s_1 x_1 \rceil \le I \\ 1 \le \lceil s_2 x_2 \rceil \le J \end{array} \\ 0 & otherwise \end{array} \right\}, t \ge 0 \tag{1.22}$$

1.3.3.2 Dynamic Travelling Salesman Problems (DTSPs)

TSP is another classical NP-complete combinatorial problem. DTSPs have a wide range of real applications, especially in the optimization of dynamic networks, like network planning and designing, load-balance routing, and traffic management.

In [35], a DTSP is defined as a TSP with a dynamic cost (distance) matrix as:

$$D(t) = \{d_{ij}(t)\}_{n*n} \tag{1.23}$$

where $d_{ij}(t)$ is the cost from city i to city j, n is the number of cities. DTSP can be defined as $f(x,t)$, the objective of DTSP is to find a minimum-cost route containing all cities at time t. It can be described as:

$$f(x,t) = Min(\sum_{i=1}^{n} d_{T_i,T_{i+1}}(t)) \tag{1.24}$$

where $x_i \in 1,2,\cdots,n$ denotes the i-th city in the solution such that $x_{n+1} = x_1$ and, if $i \neq j$, $x_i \neq x_j$.

1.3.3.3 Dynamic Routing Problems in Communication Networks

In [16, 17, 81], Cheng *et al.* studied the dynamic shortest path routing and dynamic multi-cast routing problems in mobile ad hoc networks (MANETs).

The MANETs in [16, 17, 81] were modeled within a fixed geographical graph G_0, which is composed by a set of wireless nodes and a set of communication links connecting two neighbor nodes that fall into the radio transmission range. To simulate the wireless network in dynamic environments in real world, two different kind of models were created, which are the general dynamics model and the worst dynamics model, respectively. In the general dynamics model, periodically or randomly, some nodes are scheduled to sleep or some sleeping nodes are scheduled to wake up due to energy conservation. While, in the worst model, each change is produced manually by removal of a few links on the current best multi-cast tree.

The models can be described by a MANET $G(V,E)$ and a multi-cast communication request from node s to a set of receivers R with a delay upper bound δ. So, the dynamic delay-constrained multi-cast routing problem is to find a series of trees $\{T_i|i \in \{0,1,\dots\}\}$ over a series of graphs $\{G_i|i \in \{0,1,\dots\}\}$, which satisfy the dylay constraint and have the least tree cost as follows:

$$\max_{r_j \in R}\{\sum_{l \in P_T(s,r_j)} d_l\} \leq \delta \tag{1.25}$$

$$C(T_i) = \min_{T \in G_i}\{\sum_{l \in T(V_T,E_T)} c_l\} \tag{1.26}$$

where $G_i(V_i,E_i)$ is MANET topology after the i-th change, $R = \{r_0, r_1, \cdots, r_m\}$ is a set of receivers of a multi-cast request, $T_i(V_{T_i},E_{T_i})$ is a multi-cast tree with nodes V_{T_i} and links E_{T_i}, $P_T(s,r_j)$ is a path from s to r_j on the tree T_i, d_l represents the transmission delay on the communication link l, C_{T_i} is the cost of the tree T_i, and $\delta(P_i)$ is the total transmission delay on the path P_i.

1.3.3.4 Dynamic Vehicle Routing Problems

In [42, 75], dynamic vehicle routing problems have been investigated by the EC community. In [75], a freight transportation planning models was proposed. A standardized container with an extent of roughly 7.5m×2.6m×2.7m, termed by a swap body b, was considered as the basic unit of freight. All possible transportation means are referred as F, and all trucks $tr \in F$ can carry $\hat{v}(tr) = 2$ swap bodies at once whereas the capacity limits of trains $z \in F$ are usually between 30 and 60 ($\hat{v}(z) \in [30,60]$). The schedules of trains, e.g., routes, departure times, and arrival times, are fixed. Freight trucks can take any route on the map, but must perform cyclic tours. The locations where a freight may be collected or delivered are referred to L. Each transportation order has a fixed time window $[\check{t}_s, \hat{t}_s]$ in which it must be collected from its source $l_s \in L$ and a destination location and time window $[\check{t}_d, \hat{t}_d]$ in which it must be delivered to its destination $l_d \in L$. It also has a volume v that is the capacity of a swap body times by an integer. Therefore, a transportation order o can be described by a tuple $o = \langle l_s, l_d, [\check{t}_s, \hat{t}_s], [\check{t}_d, \hat{t}_d], v\rangle$. In the model, all orders that need more than one ($v > 1$) swap body are split into multiple orders where each requires one swap body.

Finally, the planning process becomes a set of R tours, each tour r is described by a tuple $r = \langle l_s, l_d, f, \check{t}, \hat{t}, \underline{b}, \underline{o} \rangle$ where l_s and l_d are the start and destination locations, $\check{t} and \hat{t}$ are the departure and arrival time, $\underline{b} = \{b_1, b_2, \ldots\}$ is a set of swap bodies which are carried by the vehicle $f \in F$ and contains the goods assigned to the orders $\underline{o} = \{o_1, o_2, \ldots\}$.

1.4 Performance Metrics

In addition to develop different dynamic benchmark generators and problems for testing EAs for DOPs, another relevant issue is how to compare different algorithms. Over the years, researchers have developed a number of different performance measures to evaluate the developed EA approaches for DOPs. The widely used performance measures, which can be classified into two main groups: optimality-based and behaviour-based, are reviewed as follows.

1.4.1 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:

1.4.1.1 Best-of-Generation

This measure is calculated as the averages for many runs of the best values at each generation on the same problem. It is usually used in two ways: First, the best value in each generation is plotted against the time axis to create a performance curve. This measure has been used since the early research in [5, 18, 25, 27, 28]. This measure is still one of the most commonly used measures in the literature. The advantage of such performance curves is that they can show the whole picture of how the tested algorithm has performed. However, because the performance curve is not quantitative, it is difficult to compare the final outcome of different algorithms and to see if the difference between two algorithms is statistically significant [47].

To improve the above disadvantage, a variation of the measure is proposed where the best-of-generation values is averaged over all generations [86]. The measure is described below:

$$\overline{F}_{BOG} = \frac{1}{G} \times \sum_{i=1}^{i=G} \left(\frac{1}{N} \times \sum_{j=1}^{j=N} F_{BOG_{ij}} \right) \tag{1.27}$$

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

to the $\overline{F}_{BOG}$, but with different name, the *collective mean fitness*, was proposed by Morrison [47] at the same time.

Recently the idea of plotting performance curves was adapted in [3] to create two measures: the *area below a curve*, which is calculated as the definite integral of F_{BOG} (or other measures such as F_C or offline error/performance) over the optimisation process; and the *area between curves*, which is the area spanned between the performance curves of two algorithms.

The $\overline{F}_{BOG}$ 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.

1.4.1.2 Best-Error-Before-Change

Proposed in [67] and named *Accuracy* by the authors, 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).

$$E_B = \frac{1}{m} \sum_{i=1}^{m} e_B(i) \tag{1.28}$$

where $e_B(i)$ 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 [36].

1.4.1.3 Modified Offline Error and Offline Performance

Proposed in [11] and [12], 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.

$$E_{MO} = \frac{1}{n}\sum_{j=1}^{n} e_{MO}(j) \tag{1.29}$$

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

$$P_{MO} = \frac{1}{n}\sum_{j=1}^{n} F_{MO}(j) \tag{1.30}$$

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.

With this type of measures, the faster the algorithm to find a good solution, the higher the score. 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 [51][Sect. 5.3.2], the offline error/performance was modified to measure the performance of algorithms in dynamic constrained environments. Specifically, when calculating Eq. (1.29) 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.

Recently based on the modified offline error a new measure named *best known peak error* (BKPE) [6] was proposed to measure the convergence speed of the algorithm in tracking optima. Different to the modified offline error, in BKPE at each generation the error is calculated for each known peak, i.e. it is the difference between the best found solution in the peak and the top of the peak. Then immediately before a change, the error of the best individual on a known peak is added to the total error for the run.

1.4.1.4 Optimisation Accuracy

The *optimisation accuracy* measure (also known as the *relative error*) was initially proposed in [24] and was adopted in [72] for the dynamic case:

$$accuracy_{F,EA}^{(t)} = \frac{F(best_{EA}^{(t)}) - Min_F^{(t)}}{Max_F^{(t)} - Min_F^{(t)}} \tag{1.31}$$

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.

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 [72], 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. (1.31) at t would be equal to zero.

1.4.1.5 Normalised Scores

Another useful way to avoid the possible biases caused different fitness scales in different change periods and/or different problems is to use the *normalised score* [51], which evaluates the overall performance of an algorithm compared to other peer algorithms in solving a group of problems in a normalised way. 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.

Given a group of n tested algorithms and m test instances, a formal description of the *normalised score* of the ith algorithm is given in Eq. (1.32):

$$S_{norm}(i) = \frac{1}{m}\sum_{j=1}^{m} \frac{|e_{\max}(j) - e(i,j)|}{|e_{\max}(j) - e_{\min}(j)|}, \forall i = 1:n. \tag{1.32}$$

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 is unbiased. The fact that an algorithm might get a very large or very small error on a particular problem or on a particular change period will not bias the overall score as it does when we use the traditional measures. 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.

1.4.1.6 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 [74], a performance measure, which is calculated as the minimum distance from the individuals in the population to the global optimum, was proposed. In [63], another distance-based measure was introduced. This measure is calculated as the distance from the mass centre of the population to the global optimum.

Euclidean distance-based measures are also commonly used to evaluate the performance of dynamic multi-objective (DMO) optimisation algorithms. In [91] the performance of DMO algorithms are evaluated based on the *generational distance* (GD) [69] between the approximated front (which contains the current best function values) and the Pareto optimal front at the moment just before a change occurs. In [23] two measures, one is based on the minimum Euclidean distance between members of the approximated front and the Pareto front, and the other is based on the minimum Euclidean distance between members of the approximated set and the Pareto set, were proposed. In [29], these two measures were extended using the idea of modified offline-error. In [38], a modified version of the original GD named *reversed GD* was proposed for the dynamic case. In [26], an offline measure named *variable space generational distance* was also proposed and was calculated based on the distance between the approximated set and the Pareto set at each time step.

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 [72].

1.4.2 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.

1.4.2.1 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* [43], *Hamming distance* [54, 56, 80], *moment-of-inertia* [46], *peak cover* [11], and *maximum spread* [26] 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 [54] where the *all possible pair-wise Hamming distance* among all individuals of the population was used as the diversity measure. In [56] 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* [46], 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 [46] 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:

$$c_i = \frac{\sum_{j=1}^{P} x_{ij}}{P} \tag{1.33}$$

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:

$$I = \sum_{i=1}^{N} \sum_{j=1}^{P} (x_{ij} - c_i)^2 \tag{1.34}$$

In [46], the authors proved that the moment-of-inertia measure is equal to the pair-wise Hamming distance measure. 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* [11], 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.

Diversity measures are also used in dynamic multi-objective approaches. In [26] the *maximum spread* commonly used in static MO was modified for the dynamic case by calculating the average value of the maximum spread over all generations when time goes by. In [38], the diversity-based *hypervolume* (HV) measure [69] commonly used in static MO was extended to a dynamic measure HVR(t), which is the ratio between the dynamic HV of the approximated front and the Pareto front.

In dynamic constrained environments, a diversity-related measure was also proposed [51][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.

1.4.2.2 Performance Drop 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* [72], *satisficability* and *robustness* [56].

The measure *stability* is evaluated by calculating the difference in the fitness-based *accuracy* measure (see Eq. (1.31)) of the considered algorithm between each two time steps

$$stab_{F,EA}^{(t)} = max\{0, accuracy_{F,EA}^{(t-1)} - accuracy_{F,EA}^{(t)}\} \tag{1.35}$$

where $accuracy_{F,EA}^{(t)}$ has already been defined in Eq. (1.31).

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.

1.4.2.3 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 [72], 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:

$$react^{(t)}_{F,A,\varepsilon} = min \left\{ t' - t | t < t' \leq maxgen, t' \in \mathbb{N}, \frac{accuracy^{(t')}_{F,A}}{accuracy^{(t)}_{F,A}} \geq (1-\varepsilon) \right\} \cup \{maxgen - t\} \tag{1.36}$$

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, the value of the measure *reactivity* is always zero and nothing can be said about how well the algorithm reacts to changes. In situations like the dynamic constrained benchmark problems in [53] where the total fitness level of the search space may increase after a change, the measure *reactivity* cannot be used.

To avoid the disadvantage of *reactivity* and to provide more insights on the convergence behaviour of algorithms, recently a pair of measures, the *recovery rate* (RR) and the *absolute recovery rate* (ARR) were proposed [51]. The RR measure is used to analyse *how quick it is for an algorithm to recover from an environmental change and to start converging on a new solution before the next change occurs*.

$$RR = \frac{1}{m} \sum_{i=1}^{m} \frac{\sum_{j=1}^{p(i)} \left[f_{best}(i,j) - f_{best}(i,1) \right]}{p(i) \left[f_{best}(i,p(i)) - f_{best}(i,1) \right]} \tag{1.37}$$

where $f_{best}(i,j)$ 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 $p(i), i = 1 : m$ is the number of generations at each change period i. The RR score would be equal to 1 in the best case where the algorithm is able to recover and converge on a solution immediately after a change, and would be equal to zero in case the algorithm is unable to recover from the drop at all.

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*:

$$ARR = \frac{1}{m} \sum_{i=1}^{m} \frac{\sum_{j=1}^{p(i)} \left[f_{best}(i,j) - f_{best}(i,1) \right]}{p(i) \left[f^*(i) - f_{best}(i,1) \right]} \tag{1.38}$$

where $f_{best}(i,j), i, j, m, p(i)$ are the same as in Eq. (1.37) and $f^*(i)$ 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.

Beside the advantage of working even in case there is no drop in performance, the RR and ARR measures can also be used together in an *RR-ARR diagram* (see [51][Fig. 5.2, page 118]) to indicate if an algorithm is able to converge on the global optimum; or if it has suffered from slow convergence or pre-mature convergence.

1.4.2.4 Fitness Degradation over Time

A recent experimental observation [2] 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 [2] 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 [53] mentioned above, a positive $\beta - degradation$ can be achieved even when the algorithm stays at the same place.

1.4.3 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., [51, 56, 90], 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 [51, 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, as shown in the literature review in this section, many optimality-based measures are not normalised and hence might be biased by fitness rescalings and other disproportionate factors caused by the changing landscapes. The *accuracy* measure [72] is among the few studies that tried to overcome this disadvantage by normalising the fitness values at each change period using a window of the maximum and minimum possible values. This approach, however, requires full

knowledge of the maximum and minimum possible values at each change period, which might not be available in practical situations. The normalised score proposed in [51] offers an alternative way to compare the performance of algorithms in a normalised way without using 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 [2] 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" [1, 2]. 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.

1.5 The Generalized Dynamic Benchmark Generator (GDBG)

The dynamic test problems described in Section 1.3 are based on different search spaces. In order to develop a unified approach of constructing dynamic problems across the binary, real, and combinatorial spaces, a GDBG system has recently been proposed in [36, 37]. For the GDBG system, DOPs can be defined as follows:

$$F = f(x, \phi, t) \tag{1.39}$$

where F is the optimization problem, f is the cost function, x is a feasible solution in the solution set $\mathbf{X}$, t is the real-world time, and ϕ is the system control parameter, which determines the solution distribution in the fitness landscape. The objective is to find a global optimal solution x^* such that $f(x^*) \leq f(x) \forall x \in \mathbf{X}$ (without loss of generality, minimization problems are considered).

In the GDBG system, the dynamism results from a deviation of solution distribution from the current environment by tuning the system control parameters. It can be described as follows:

$$\phi(t+1) = \phi(t) \oplus \Delta\phi \tag{1.40}$$

where $\Delta\phi$ is a deviation from the current system control parameters. Then, we can get the new environment at the next moment $t+1$ as follows:

$$f(x, \phi, t+1) = f(x, \phi(t) \oplus \Delta\phi, t) \tag{1.41}$$

The system control parameters decide the distribution of solutions in the solution space. They may be different from one specific instance to another instance. The GDBG system constructs dynamic environments by changing the values of these system control parameters. There are seven change types of the system control parameters in the GDBG system, which are defined as follows:

- T1 (small step):

$$\Delta\phi = \alpha \cdot \|\phi\| \cdot r \cdot \phi_{severity} \tag{1.42}$$

- T2 (large step):

$$\Delta\phi = \|\phi\| \cdot (\alpha \cdot sign(r) + (\alpha_{max} - \alpha) \cdot r) \cdot \phi_{severity} \tag{1.43}$$

- T3 (random):

$$\Delta\phi = N(0,1) \cdot \phi_{severity} \tag{1.44}$$

- T4 (chaotic):

$$\phi(t+1) = A \cdot \phi(t) \cdot (1 - \phi(t)/\|\phi\|) \tag{1.45}$$

- T5 (recurrent):

$$\phi(t+1) = \phi_{min} + \|\phi\|(\sin(\frac{2\pi}{P}t + \varphi) + 1)/2 \tag{1.46}$$

- T6 (recurrent with noise):

$$\phi(t+1) = \phi_{min} + \|\phi\|(\sin(\frac{2\pi}{P}t + \varphi) + 1)/2 + N(0,1) \cdot noise_{severity} \tag{1.47}$$

- T7 (dimensional change):

$$D(t+1) = D(t) + sign \cdot \Delta D, \tag{1.48}$$

where $\|\phi\|$ is the change range of ϕ, $\phi_{severity} \in (0,1)$ is change severity of ϕ, ϕ_{min} is the minimum value of ϕ, $noisy_{severity} \in (0,1)$ is noisy severity in recurrent with noisy change. $\alpha \in (0,1)$ and $\alpha_{max} \in (0,1)$ are constant values, which are set to 0.02 and 0.1 in the GDBG system. A logistics function is used in the chaotic change type, where A is a positive constant between $(1.0, 4.0)$, if ϕ is a vector, the initial values of the items in ϕ should be different within $\|\phi\|$ in chaotic change. P is the period of recurrent change and recurrent change with noise, φ is the initial phase, r is a random number in $(-1,1)$, $sign(x)$ returns 1 when x is greater than 0, returns -1 when x is less than 0, otherwise, returns 0. $N(0,1)$ denotes a normally distributed one dimensional random number with mean zero and standard deviation one. For T7, ΔD is a predefined constant, which the default value of is 1. If $D(t) = Max\_D$, $sign = -1$; if $D(t) = Min\_D$, $sign = 1$. Max_D and Min_D are the maximum and minimum number of dimensions. When the number of dimensions deceases by 1, just the last dimension is removed from the fitness landscape, and the fitness landscape of the left dimensions does not change. When the number of dimensions increases by 1, a new dimension with a random value is added into the fitness landscape. Dimensional change *only happens following* the non-dimensional change.

The GDBG system can be instantiated to construct specific DOP instances in the binary space, real space, and combinatorial space, respectively. Both the DF1 [44] and MPB [9] generators have a disadvantage: the challenge per change is unequal for algorithms when the position of a peak bounces back from the search boundary. From the GDBG system, we can construct two different real space DOPs using a rotation method on the peak position to overcome that shortcoming. The two benchmark

instances are: dynamic rotation peak benchmark generator (DRPBG) and dynamic composition benchmark generator (DCBG), described as follows.

1.5.1 Dynamic Rotation Peak Benchmark Generator

The DRPBG uses a similar peak-composition structure to those of the MPB [9] and DF1 [44] generators. Given a problem $f(x,\phi,t), \phi = (\mathbf{H},\mathbf{W},\mathbf{X})$, where $\mathbf{H}$, $\mathbf{W}$, and $\mathbf{X}$ denote the peak height, width, and position, respectively. The function of $f(x,\phi,t)$ is defined as follows:

$$f(x,\phi,t) = \max_{i=1}^{m}(\mathbf{H}_i(t)/(1+\mathbf{W}_i(t)\cdot\sqrt{\sum_{j=1}^{n}\frac{(x_j-\mathbf{X}_j^i(t))^2}{n}})) \tag{1.49}$$

where m is the number of peaks and n is the number of dimensions.

$\mathbf{H}$ and $\mathbf{W}$ change as follows:

$$\mathbf{H}(t+1) = \text{DynamicChanges}(\mathbf{H}(t)) \tag{1.50}$$

$$\mathbf{W}(t+1) = \text{DynamicChanges}(\mathbf{W}(t)) \tag{1.51}$$

where in the height change, *height_severity* should read $\phi\_h_{severity}$ according to Eq. (1.49) and $\|\phi\_h\|$ is the height range. Accordingly, *width_severity* should read $\phi\_w_{severity}$ and $\|\phi\_w\|$ is the width change.

A rotation matrix $R_{ij}(\theta)$ is obtained by rotating the projection of $\overrightarrow{x}$ in the plane $i-j$ by an angle θ from the i-th axis to the j-th axis. The peak position $\mathbf{X}$ is changed by the following algorithm:

Step 1: Randomly select l dimensions (l is an even number) from the n dimensions to compose a vector $r = [r_1, r_2, ..., r_l]$.

Step 2: For each pair of dimension $r[i]$ and dimension $r[i+1]$, construct a rotation matrix $R_{r[i],r[i+1]}(\theta(t))$, where:

$$\theta(t) = \text{DynamicChanges}(\theta(t-1)). \tag{1.52}$$

Step 3: A transformation matrix $A(t)$ is obtained by:

$$A(t) = R_{r[1],r[2]}(\theta(t))\cdot R_{r[3],r[4]}(\theta(t))\cdots R_{r[l-1],r[l]}(\theta(t)) \tag{1.53}$$

Step 4: Update the peak position by:

$$\mathbf{X}(t+1) = \mathbf{X}(t)\cdot A(t) \tag{1.54}$$

where the change severity of θ ($\phi\_\theta_{severity}$) is set to l in Eq. (4), the range of θ should read $\|\phi\_\theta\|$, $\|\phi\_\theta\| \in (-\pi,\pi)$. For the value of l, if n is an even number,

$l = n$; otherwise $l = n-1$. Note that, for recurrent and recurrent with noisy changes, $\|\phi\_\theta\|$ is within $(0, \pi/6)$.

1.5.2 Dynamic Composition Benchmark Generator

The dynamic composition functions, which are extended from the static composition functions devised by Suganthan et al. [39], can be described as follows:

$$F(x,\phi,t) = \sum_{i=1}^{m} (w_i \cdot (f_i'((x - \mathbf{O}_i(t) + O_{iold})/\lambda_i \cdot \mathbf{M}_i) + \mathbf{H}_i(t))) \tag{1.55}$$

where the system control parameter $\phi = (\mathbf{O}, \mathbf{M}, \mathbf{H})$, $F(x)$ is the composition function, $f_i(x)$ is i-th basic function used to construct the composition function, m is the number of basic functions, $\mathbf{M}_i$ is orthogonal rotation matrix for each $f_i(x)$, $\mathbf{O}_i(t)$ is the optimum of the changed $f_i(x)$ caused by rotating the landscape at the time t, O_{iold} is the optimum of the original $f_i(x)$ without any change. O_{iold} is 0 for all the basic functions. The weight value w_i for each $f_i(x)$ is calculated as:

$$w_i = exp(-sqrt(\frac{\sum_{k=1}^{n}(x_k - o_i^k + o_{iold}^k)^2}{2n\sigma_i^2})) \tag{1.56}$$

$$w_i = \begin{cases} w_i & \text{if } w_i = max(w_i) \\ w_i \cdot (1 - max(w_i)^{10}) & \text{if } w_i \neq max(w_i) \end{cases} \tag{1.57}$$

$$w_i = w_i / \sum_{i=1}^{m} w_i \tag{1.58}$$

where σ_i is the converge range factor of $f_i(x)$, whose default value is 1.0, λ_i is the stretch factor for each $f_i(x)$, which is defined as:

$$\lambda_i = \sigma_i \cdot \frac{X_{max} - X_{min}}{x_{max}^i - x_{min}^i} \tag{1.59}$$

where $[X_{max}, X_{min}]^n$ is the search range of $F(x)$ and $[x_{max}^i, x_{min}^i]^n$ is the search range of $f_i(x)$.

In Eq. (1.55), $f_i'(x) = C \cdot f_i(x)/|f_{max}^i|$, where C is a constant, which is set to 2000, and f_{max}^i is the estimated maximum value of $f_i(x)$, which is estimated as:

$$f_{max}^i = f_i(x_{max} \cdot M_i) \tag{1.60}$$

In the DCBG, $\mathbf{M}$ is initialized using the above transformation matrix construction algorithm and then remains unchanged. The dynamism of the system control parameter $\mathbf{H}$ and $\mathbf{O}$ are changed as the parameters $\mathbf{H}$ and $\mathbf{X}$ in dynamic rotation peak benchmark generator. Note that, for both DRPBG and DCBG, chaotic change of peaks locations directly operates on the value of each dimension instead of using rotation matrix due to simulating chaotic systems in real applications.

Table 1.2 Details of the basic benchmark functions

Name	Function	Range
Sphere	$f(x) = \sum_{i=1}^{n} x_i^2$	[-100,100]
Rastrigin	$f(x) = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10)$	[-5,5]
Weierstrass	$f(x) = \sum_{i=1}^{n} (\sum_{k=0}^{k_{max}} [a^k \cos(2\pi b^k (x_i + 0.5))])$	
	$-n \sum_{k=0}^{k_{max}} [a^k \cos(\pi b^k)], a = 0.5, b = 3, k_{max} = 20$	[-0.5,0.5]
Griewank	$f(x) = \frac{1}{4000} \sum_{i=1}^{n} (x_i)^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$	[-100,100]
Ackley	$f(x) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)) + 20 + e$	[-32,32]

Table 1.3 Default settings for the GDBG used for CEC 2009 Competition on Dynamic Optimization

Parameter	Value
number of dimensions D	fixed: 10; changing: [5-15]
search range	$x \in [-5,5]^D$
number of functions or peaks p	$p = 10$
change frequency U	$10,000 \times D$ fitness evaluations
number of changes K	$K = 60$
period P	$P = 12$
severity of noisy $noisy_{severity}$	$noisy_{severity} = 0.8$
chaotic constant A	$A = 3.67$
step severity α	$\alpha = 0.04$
maximum of α	$\alpha_{max} = 0.1$
height range	$h \in [10,100]$
initial height $initial\_height$	$initial\_height = 50$
severity of height change $\phi\_h_{severity}$	$\phi\_h_{severity} = 5.0$
sampling frequency s_f	$s\_f = 100$

Five basic benchmark functions are used in the GDBG system. Table 1.2 shows the details of the five functions.

1.5.3 Dynamic Test Problems for the CEC 2009 Competition

The GDBG was used to construct dynamic test problems for the CEC 2009 Competition on Dynamic Optimization [37], where the following seven different particular functions are defined: the rotation peak function with 10 peaks (F_1 with $p = 10$), the rotation peak function with 50 peaks (F_1 with $p = 50$), the composition of Sphere's functions (F_2), the composition of Rastrigin's functions (F_3), the composition of Griewank's functions (F_4), the composition of Ackley's functions (F_5), and the hybrid composition function (F_6). The detailed settings of each function can be found in [37] and the general parameter settings are given in Table 1.3.

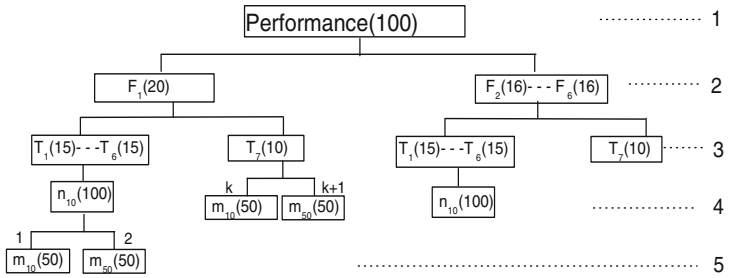

Fig. 1.2 Overall performance measurement

There are 49 test cases in total constructed from the seven test problems in the GDBG benchmark. For an algorithm on each test case, the offline error (e_{off}) and its standard variance (STD) are recorded, which are defined as in [37] as follows:

$$e_{off} = \frac{1}{R*K}\sum_{r=1}^{R}\sum_{k=1}^{K} e_{r,k}^{last} \tag{1.61}$$

$$STD = \sqrt{\frac{\sum_{r=1}^{R}\sum_{k=1}^{K}(e_{r,k}^{last} - e_{off})^2}{R*K-1}} \tag{1.62}$$

where R and K are the total number of runs and the number of environmental changes for each run, respectively, and $e_{r,k}^{last} = |f(\mathbf{x}_{best}(r,k)) - f(\mathbf{x}^*(r,k))|$, where $\mathbf{x}^*(r,k)$ is the global optimum of the k-th environment and $\mathbf{x}_{best}(r,k)$ is the position of the best particle of the last generation of the k-th environment during the r-th run.

The calculation of the overall performance of an algorithm on all 49 test cases is as illustrated in Fig. 1.2, where F_1-F_6 denote the six functions defined in the GDBG benchmark in [37], T_1-T_7 represent the seven change types, n_{10} means that the number of dimensions is ten, and m_{10} and m_{50} denote that the number of peaks is 10 and 50, respectively. Each test case i is assigned a weight w_i and the sum of weights of all the test cases is 1.0. The mark obtained by an algorithm on test case $i \in \{1,\ldots,49\}$ is calculated by:

$$mark_i = \frac{w_i}{R*K}\sum_{r=1}^{R}\sum_{k=1}^{K}\left(r_{rk}^{last}/\left(1+\frac{1}{S}\sum_{s=1}^{S}(1-r_{rk}^{s})\right)\right) \tag{1.63}$$

where r_{rk}^{last} is the relative ratio of the best particle fitness of the last generation to the global optimum of the k-th environment, r_{rk}^{s} is the relative ratio of the best particle's fitness to the global optimum at the s-th sampling during the k-th environment

(*initial population should be sampled*), and $S = U/s\_f$ is the total number of samples for each environment. The relative ratio r^s_{rk} is defined by

$$r^s_{rk} = \begin{cases} \frac{f(\mathbf{x}_{best}(r,k,s))}{f(\mathbf{x}^*(r,k))}, & f = F_1 \\ \frac{f(\mathbf{x}^*(r,k))}{f(\mathbf{x}_{best}(r,k,s))}, & f \in \{F_2, F_3, F_4, F_5, F_6\} \end{cases} \tag{1.64}$$

where $\mathbf{x}_{best}(r,k,s)$ is the position of the best particle up to the s-th sample in the k-th environment during the r-th run.

The overall performance of an algorithm on all the test cases is then calculated as follows:

$$performance = 100 \times \sum_{i=1}^{49} mark_i \tag{1.65}$$

1.6 Conclusions and Discussions

Developing proper dynamic test and eveluation environments is an important task in studying EC for DOPs. In this chapter, we present the concept of DOPs and review existing dynamic test problems commonly used by researchers to investigate their EC approaches in the literature. Some discussions regarding the major features and classification of existing dynamic test environments are presented. Some typical dynamic benchmark problems and real-world DOPs, which cover the binary, real, and combinatorial spaces, are also described in detail. We also review the performance measures that are widely used by researchers to evaluate and compare their developed EC approaches for DOPs.

The review has 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 1.4.1. Some studies do pay attention to developing other complementary measures (e.g. the behaviour-based measures in Section 1.4.2), 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 almost 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*: 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.
- *Reason for tracking*: The main assumption is that the optima (local or global) after change is close to the optima (local or global) before change, as shown in a majority of benchmark problems (although in the Moving peaks [9] and DF1 [44] benchmarks the new global optima are not close to the previous global optima, they are still close to a previous local optima). Due to that, tracking is preferred to restarting.
- *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 as *memory* and *prediction*.

The review in this chapter 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 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. So far there is very little reserach aiming at answering these questions. One exception is the recent study in [51, chap. 3] where a large set of recent "real"[2] real-world dynamic optimisation problems has been reviewed to investigate the real characteristics of real-world problems and how they relate to the characteristics of current academic benchmark problems. The reserch in [51] 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.

Acknowledgements. This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of U.K. under Grant numbers EP/E060722/1, EP/K001310/1, and EP/E058884/1, and partially supported by an UK ORS Award and a studentship from the School of Computer Science, University of Birmingham and an EU-funded project named "Intelligent Transportation for Dynamic Environment (InTraDE)".

References

[1] Alba, E., Saucedo Badia, J., Luque, G.: A study of canonical gas for nsops. In: Doerner, et al. (eds.) Metaheuristics. Operations Research/Computer Science Interfaces Series, vol. 39, pp. 245–260. Springer (2007)

[2] Only references that actual 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.

[2] Alba, E., Sarasola, B.: Measuring fitness degradation in dynamic optimization problems. In: Di Chio, C., et al. (eds.) EvoApplicatons 2010, Part I. LNCS, vol. 6024, pp. 572–581. Springer, Heidelberg (2010)
[3] Alba, E., Sarasola, B.: Abc, a new performance tool for algorithms solving dynamic optimization problems. In: Proc. 2010 IEEE World Congr. on Comput. Intell. (WCCI 2010), pp. 734–740 (2010)
[4] Aragón, V.S., Esquivel, S.C.: An evolutionary algorithm to track changes of optimum value locations in dynamic environments. J. of Comp. Sci. and Tech. 4(3), 127–134 (2004)
[5] Bäck, T.: On the behavior of evolutionary algorithms in dynamic environments. In: Proc. 1998 IEEE Int. Conf. Evol. Comput., pp. 446–451 (1998)
[6] Bird, S., Li, X.: Informative performance metrics for dynamic optimisation problems. In: Proc. 9th Annual Conf. on Genetic and Evol. Comput., pp. 18–25 (2007)
[7] Bosman, P.A.N.: Learning and anticipation in online dynamic optimization. In: Yang, S., Ong, Y.-S., Jin, Y. (eds.) Evolutionary Computation in Dynamic and Uncertain Environments. SCI, vol. 51, pp. 129–152. Springer, Heidelberg (2007)
[8] Bosman, P.A.N., Poutré, H.L.: Learning and anticipation in online dynamic optimization with evolutionary algorithms: the stochastic case. In: Proc. 9th Annual Conf. on Genetic and Evol. Comput., pp. 1165–1172 (2007)
[9] Branke, J.: Memory enhanced evolutionary algorithms for changing optimization problems. In: Proc. 1999 IEEE Congr. Evol. Comput., pp. 1875–1882 (1999)
[10] Branke, J.: Evolutionary approaches to dynamic environments - updated survey. In: Proc. 2001 GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems, pp. 27–30 (2001)
[11] Branke, J.: Evolutionary Optimization in Dynamic Environments. Kluwer Academic Publishers (2002)
[12] Branke, J., Schmeck, H.: Designing evolutionary algorithms for dynamic optimization problems. In: Tsutsui, S., Ghosh, A. (eds.) Theory and Application of Evolutionary Computation: Recent Trends, pp. 239–262. Springer (2003)
[13] Branke, J., Kaußler, T., Schmidth, C., Schmeck, H.: A multi-population approach to dynamic optimization problems. In: Proc. 4th Int. Conf. Adaptive Comput. Des. Manuf., pp. 299–308 (2000)
[14] Branke, J., Orbayı, M., Uyar, Ş.: The role of representations in dynamic knapsack problems. In: Rothlauf, F., et al. (eds.) EvoWorkshops 2006. LNCS, vol. 3907, pp. 764–775. Springer, Heidelberg (2006)
[15] Chazottes, J., Fernandez, B.: Dynamics of Coupled Map Lattices and of Related Spatially Extended Systems. Springer, Heidelberg (2005)
[16] Cheng, H., Yang, S.: Multi-population genetic algorithms with immigrants scheme for dynamic shortest path routing problems in mobile ad hoc networks. In: Di Chio, C., et al. (eds.) EvoApplicatons 2010, Part I. LNCS, vol. 6024, pp. 562–571. Springer, Heidelberg (2010)
[17] Cheng, H., Yang, S.: Genetic algorithms with immigrants schemes for dynamic multicast problems in mobile ad hoc networks. Engg. Appl. of Artif. Intell. 23(5), 806–819 (2010)
[18] Cobb, H.G.: 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)
[19] Cobb, H.G., Grefenstette, J.J.: Genetic algorithms for tracking changing environments. In: Proc. 5th Int. Conf. Genetic Algorithms, pp. 523–530 (1993)

[20] Cruz, C., Gonzalez, J.R., Pelta, D.A.: Optimization in dynamic environments: A survey on problems, methods and measures. Soft Comput. 15(7), 1427–1448 (2011)
[21] Droste, S.: Analysis of the (1+1) EA for a dynamically changing onemax-variant. In: Proc. 2002 IEEE Congr. Evol. Comput., pp. 55–60 (2002)
[22] Eberhart, R.C., Kennedy, J.: A new optimizer using particle swarm theory. In: Proc. 6th Int. Symp. on Micro Machine and Human Science, pp. 39–43 (1995)
[23] Farina, M., Deb, K., Amato, P.: Dynamic multiobjective optimization problems: test cases, approximations, and applications. IEEE Trans. Evol. Comput. 8(5), 425–442 (2004)
[24] Feng, W., Brune, T., Chan, L., Chowdhury, M., Kuek, C., Li, Y.: Benchmarks for testing evolutionary algorithms. Technical Report, Center for System and Control, University of Glasgow (1997)
[25] Gaspar, A., Collard, P.: From gas to artificial immune systems: Improving adaptation in time dependent optimization. In: Proc. 1999 IEEE Congr. Evol. Comput., vol. 3, pp. 1859–1866 (1999)
[26] Goh, C.K., Tan, K.C.: A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization. IEEE Trans. Evol. Comput. 13(1), 103–127 (2009)
[27] Grefenstette, J.J.: Genetic algorithms for changing environments. In: Proc. 2nd Int. Conf. Parallel Problem Solving from Nature, pp. 137–144 (1992)
[28] Grefenstette, J.J.: Evolvability in dynamic fitness landscapes: a genetic algorithm approach. In: Proc. 1999 IEEE Congr. Evol. Comput., vol. 3, pp. 2031–2038 (1999)
[29] Hatzakis, I., Wallace, D.: Dynamic multi-objective optimization with evolutionary algorithms: a forward-looking approach. In: Proc. 8th Annual Conf. on Genetic and Evol. Comput., pp. 1201–1208 (2006)
[30] Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments – a survey. IEEE Trans. Evol. Comput. 9(3), 303–317 (2005)
[31] Jin, Y., Sendhoff, B.: Constructing dynamic optimization test problems using the multi-objective optimization concept. In: Raidl, G.R., et al. (eds.) EvoWorkshops 2004. LNCS, vol. 3005, pp. 525–536. Springer, Heidelberg (2004)
[32] Kellerer, K., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer (2004)
[33] Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: Proc. 1995 IEEE Int. Conf. on Neural Networks, pp. 1942–1948 (1995)
[34] Lewis, J., Hart, E., Ritchie, G.: A comparison of dominance mechanisms and simple mutation on non-stationary problems. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 139–148. Springer, Heidelberg (1998)
[35] Li, C., Yang, M., Kang, L.: A new approach to solving dynamic traveling salesman problems. In: Wang, T.-D., Li, X., Chen, S.-H., Wang, X., Abbass, H.A., Iba, H., Chen, G.-L., Yao, X. (eds.) SEAL 2006. LNCS, vol. 4247, pp. 236–243. Springer, Heidelberg (2006)
[36] Li, C., Yang, S.: A generalized approach to construct benchmark problems for dynamic optimization. In: Li, X., et al. (eds.) SEAL 2008. LNCS, vol. 5361, pp. 391–400. Springer, Heidelberg (2008)
[37] Li, C., Yang, S., Nguyen, T.T., Yu, E.L., Yao, X., Jin, Y., Beyer, H.-G., Suganthan, P.N.: Benchmark generator for CEC 2009 competition on dynamic optimization. Technical Report 2008, Department of Computer Science, University of Leicester, U.K. (2008)
[38] Li, X., Branke, J., Kirley, M.: On performance metrics and particle swarm methods for dynamic multiobjective optimization problems. In: Proc. 2007 IEEE Congr. Evol. Comput., pp. 576–583 (2007)

[39] Liang, J.J., Suganthan, P.N., Deb, K.: Novel composition test functions for numerical global optimization. In: Proc. 2005 IEEE Congr. Evol. Comput., pp. 68–75 (2005)
[40] Mavrovouniotis, M., Yang, S.: A memetic ant colony optimization algorithm for the dynamic traveling salesman problem. Soft Comput. 15(7), 1405–1425 (2011)
[41] Mavrovouniotis, M., Yang, S.: Memory-based immigrants for ant colony optimization in changing environments. In: Di Chio, C., et al. (eds.) EvoApplications 2011, Part I. LNCS, vol. 6624, pp. 324–333. Springer, Heidelberg (2011)
[42] Mei, Y., Tang, K., Yao, X.: Capacitated arc routing problem in uncertain environments. In: Proc. 2010 IEEE Congr. Evol. Comput., pp. 1400–1407 (2010)
[43] Mori, N., Imanishi, S., Kita, H., Nishikawa, Y.: Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm. In: Proc. 1997 Int. Conf. on Genetic Algorithms, pp. 299–306 (1997)
[44] Morrison, R.W., De Jong, K.A.: A test problem generator for non-stationary environments. In: Proc. 1999 IEEE Congr. Evol. Comput., pp. 2047–2053 (1999)
[45] Morrison, R.W., De Jong, K.A.: Triggered hypermutation revisited. In: Proc. 2000 IEEE Congr. Evol. Comput., pp. 1025–1032 (2000)
[46] Morrison, R.W., De Jong, K.A.: Measurement of population diversity. In: Collet, P., Fonlupt, C., Hao, J.-K., Lutton, E., Schoenauer, M. (eds.) EA 2001. LNCS, vol. 2310, pp. 31–41. Springer, Heidelberg (2002)
[47] Morrison, R.W.: Performance measurement in dynamic environments. In: Proc. 2003 GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems, pp. 5–8 (2003)
[48] Morrison, R.W.: Designing Evolutionary Algorithms for Dynamic Environments. Springer, Berlin (2004)
[49] Ng, K.P., Wong, K.C.: A new diploid scheme and dominance change mechanism for non-stationary function optimization. In: Proc. 6th Int. Conf. on Genetic Algorithms, pp. 159–166 (1995)
[50] Nguyen, T.T.: A proposed real-valued dynamic constrained benchmark set. Technical report, School of Computer Science, Univesity of Birmingham (2008)
[51] Nguyen, T.T.: Continuous Dynamic Optimisation Using Evolutionary Algorithms. PhD thesis, School of Computer Science, University of Birmingham (January 2011), `http://etheses.bham.ac.uk/1296` and `http://www.staff.ljmu.ac.uk/enrtngu1/theses/phd_thesis_nguyen.pdf`
[52] Nguyen, T.T., Yao, X.: Dynamic time-linkage problems revisited. In: Giacobini, M., et al. (eds.) EvoWorkshops 2009. LNCS, vol. 5484, pp. 735–744. Springer, Heidelberg (2009)
[53] Nguyen, T.T., Yao, X.: Benchmarking and solving dynamic constrained problems. In: Proc. 2009 IEEE Congr. Evol. Comput., pp. 690–697 (2009)
[54] Oppacher, F., Wineberg, M.: The shifting balance genetic algorithm: Improving the ga in a dynamic environment. In: Proc. 1999 Genetic and Evol. Comput. Conf., vol. 1, pp. 504–510 (1999)
[55] Parrott, D., Li, X.: Locating and tracking multiple dynamic optima by a particle swarm model using speciation. IEEE Trans. Evol. Comput. 10(4), 440–458 (2006)
[56] Rand, W., Riolo, R.: Measurements for understanding the behavior of the genetic algorithm in dynamic environments: A case study using the shaky ladder hyperplane-defined functions. In: Yang, S., Branke, J. (eds.) GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization (2005)

[57] Richter, H.: Behavior of evolutionary algorithms in chaotically changing fitness landscapes. In: Yao, X., et al. (eds.) PPSN 2004. LNCS, vol. 3242, pp. 111–120. Springer, Heidelberg (2004)
[58] Richter, H.: Evolutionary optimization in spatio–temporal fitness landscapes. In: Runarsson, T.P., Beyer, H.-G., Burke, E.K., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) PPSN 2006. LNCS, vol. 4193, pp. 1–10. Springer, Heidelberg (2006)
[59] Richter, H.: Evolutionary optimization and dynamic fitness landscapes: From reaction-diffusion systems to chaotic CML. In: Zelinka, I., Celikovsky, S., Richter, H., Chen, G. (eds.) Evolutionary Algorithms and Chaotic Systems. SCI, vol. 267, pp. 409–446. Springer, Heidelberg (2010)
[60] Richter, H.: Memory design for constrained dynamic optimization problems. In: Di Chio, C., et al. (eds.) EvoApplicatons 2010, Part I. LNCS, vol. 6024, pp. 552–561. Springer, Heidelberg (2010)
[61] Rohlfshagen, P., Yao, X.: Attributes of dynamic combinatorial optimisation. In: Li, X., et al. (eds.) SEAL 2008. LNCS, vol. 5361, pp. 442–451. Springer, Heidelberg (2008)
[62] Rohlfshagen, P., Yao, X.: Dynamic combinatorial optimisation problems: an analysis of the subset sum problem. Soft Comput. 15(9), 1723–1734 (2011)
[63] Salomon, R., Eggenberger, P.: Adaptation on the evolutionary time scale: A working hypothesis and basic experiments. In: Hao, J.-K., Lutton, E., Ronald, E., Schoenauer, M., Snyers, D. (eds.) AE 1997. LNCS, vol. 1363, pp. 251–262. Springer, Heidelberg (1998)
[64] Simões, A., Costa, E.: Evolutionary algorithms for dynamic environments: Prediction using linear regression and Markov chains. In: Rudolph, G., Jansen, T., Lucas, S., Poloni, C., Beume, N. (eds.) PPSN 2008. LNCS, vol. 5199, pp. 306–315. Springer, Heidelberg (2008)
[65] Stanhope, S.A., Daida, J.M.: Optimal mutation and crossover rates for a genetic algorithm operating in a dynamic environment. In: Porto, V.W., Waagen, D. (eds.) EP 1998. LNCS, vol. 1447, pp. 693–702. Springer, Heidelberg (1998)
[66] Tinos, R., Yang, S.: Continuous dynamic problem generators for evolutionary algorithms. In: Proc. 2007 IEEE Congr. Evol. Comput., pp. 236–243 (2007)
[67] Trojanowski, K., Michalewicz, Z.: Searching for optima in non-stationary environments. In: Proc. 1999 IEEE Congr. Evol. Comput., vol. 3, pp. 1843–1850 (1999)
[68] Uyar, Ş., Uyar, H.T.: A critical look at dynamic multi-dimensional knapsack problem generation. In: Giacobini, M., et al. (eds.) EvoWorkshops 2009. LNCS, vol. 5484, pp. 762–767. Springer, Heidelberg (2009)
[69] Van Veldhuizen, D.A.: Multiobjective evolutionary algorithms: classifications, analyses, and new innovations. PhD thesis, Air Force Institute of Technolog, Wright Patterson AFB, OH, USA (1999)
[70] Wang, H., Wang, D.-W., Yang, S.: Triggered memory-based swarm optimization in dynamic environments. In: Giacobini, M. (ed.) EvoWorkshops 2007. LNCS, vol. 4448, pp. 637–646. Springer, Heidelberg (2007)
[71] Weicker, K.: An analysis of dynamic severity and population size. In: Deb, K., Rudolph, G., Lutton, E., Merelo, J.J., Schoenauer, M., Schwefel, H.-P., Yao, X. (eds.) PPSN 2000. LNCS, vol. 1917, pp. 159–168. Springer, Heidelberg (2000)
[72] Weicker, K.: Performance measures for dynamic environments. In: Guervós, J.J.M., Adamidis, P.A., Beyer, H.-G., Fernández-Villacañas, J.-L., Schwefel, H.-P. (eds.) PPSN 2002. LNCS, vol. 2439, pp. 64–73. Springer, Heidelberg (2002)
[73] Weicker, K., Weicker, N.: Dynamic rotation and partial visibility. In: Proc. 2003 IEEE Congr. Evol. Comput., pp. 1125–1131 (2003)

[74] Weicker, K., Weicker, N.: On evolution strategy optimization in dynamic environments. In: Proc. 1999 IEEE Congr. Evol. Comput., vol. 3, pp. 2039–2046 (1999)
[75] Weise, T., Podlich, A., Reinhard, K., Gorldt, C., Geihs, K.: Evolutionary freight transportation planning. In: Giacobini, M., et al. (eds.) EvoWorkshops 2009. LNCS, vol. 5484, pp. 768–777. Springer, Heidelberg (2009)
[76] Woldesenbet, Y.G., Yen, G.G.: Dynamic evolutionary algorithm with variable relocation. IEEE Trans. Evol. Comput. 13(3), 500–513 (2009)
[77] Yang, S.: Non-stationary problem optimization using the primal-dual genetic algorithm. In: Proc. 2003 IEEE Congr. Evol. Comput., pp. 2246–2253 (2003)
[78] Yang, S.: Constructing dynamic test environments for genetic algorithms based on problem difficulty. In: Proc. 2004 IEEE Congr. Evol. Comput., vol. 2, pp. 1262–1269 (2004)
[79] Yang, S.: Memory-enhanced univariate marginal distribution algorithms for dynamic optimization problems. In: Proc. 2005 IEEE Congr. Evol. Comput., vol. 3, pp. 2560–2567 (2005)
[80] Yang, S.: Genetic algorithms with memory and elitism based immigrants in dynamic environments. Evol. Comput. 16(3), 385–416 (2008)
[81] Yang, S., Cheng, H., Wang, F.: Genetic algorithms with immigrants and memory schemes for dynamic shortest path routing problems in mobile ad hoc networks. IEEE Trans. Syst., Man, and Cybern. Part C: Appl. and Rev. 40(1), 52–63 (2010)
[82] Yang, S., Li, C.: A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. IEEE Trans. Evol. Comput. 14(6), 959–974 (2010)
[83] Yang, S., Ong, Y.-S., Jin, Y. (eds.): Evolutionary Computation in Dynamic and Uncertain Environments. Springer (2007)
[84] Yang, S., Richter, H.: Hyper-learning for population-based incremental learning in dynamic environments. In: Proc. 2009 IEEE Congr. Evol. Comput., pp. 682–689 (2009)
[85] Yang, S., Tinos, R.: Hyper-selection in dynamic environments. In: Proc. 2008 IEEE Congr. Evol. Comput., pp. 3185–3192 (2008)
[86] Yang, S., Yao, X.: Dual population-based incremental learning for problem optimization in dynamic environments. In: Proc. 7th Asia Pacific Symp. on Intell. and Evol. Syst., pp. 49–56 (2003)
[87] Yang, S., Yao, X.: Experimental study on population-based incremental learning algorithms for dynamic optimization problems. Soft Comput. 9(11), 815–834 (2005)
[88] Yang, S., Yao, X.: Population-based incremental learning with associative memory for dynamic environments. IEEE Trans. Evol. Comput. 12(5), 542–561 (2008)
[89] Yao, X., Liu, Y.: Fast evolutionary programming. In: Proc. 5th Annual Conf. on Evolutionary Programming, pp. 451–460 (1996)
[90] Yu, X., Jin, Y., Tang, K., Yao, X.: Robust optimization over time – a new perspective on dynamic optimization problems. In: Proc. 2010 IEEE World Congr. Comput. Intell., pp. 3998–4003 (2010)
[91] Zeng, S., Chen, G., Zheng, L., Shi, H., de Garis, H., Ding, L., Kang, L.: A dynamic multi-objective evolutionary algorithm based on an orthogonal design. In: Proc. 2006 IEEE Congr. Evol. Comput., pp. 573–580 (2006)