Noname manuscript No.
(will be inserted by the editor)

Best Practices in Measuring Algorithm Performance for Dynamic Optimization Problems

Hajer Ben-Romdhane · Enrique Alba · Saoussen Krichen

Received: date / Accepted: date

Abstract Dynamic optimization problems (DOPs) have attracted considerable attention due to the wide range of problems they can be applied to. Lots of efforts have been expended in modeling dynamic situations, proposing algorithms, and analyzing the results (too often in a visual way). Numeric performance measurements and their statistical validation have been however barely used in the literature. Most of works in DOPs report only the *best-of-generation* fitness, due to its simplicity of computation. Although this measure indicates the best algorithm in terms of fitness, it does not provide any details about the actual strength and weakness of each algorithm. In this article, we conduct a comparative study among algorithms of different search modes via several performance measures to demonstrate their relative advantages. We discuss the role of using different performance measures in drawing balanced conclusions about algorithms for dynamic optimization problems.

Keywords Dynamic Optimization Problems · Evolutionary Algorithms · Genetic algorithms · Performance Measure

1 Introduction

In the last two decades, we have witnessed a growing interest in studying dynamic optimization problems (DOPs), as they have proven their usefulness in solving real-world complex changing tasks. In fact, realistic applications are more likely to happen with uncertain scenarios, in the sense that they involve the change of one or more of the problem specifications: i.e., the objective function, problem parameters, and problem constraints may vary in time [32]. In such environments, optimization algorithms are not only required to optimize the problem in its actual state, but also to adapt to the new optima whenever an environmental change is detected, and then, to continuously track the moving optima throughout the whole optimization process.

Several approaches and techniques were addressed over the years to solve DOPs [9], among them: particle swarm optimization, cooperative strategies, and stochastic diffusion search. However, a great deal of attention goes towards evolutionary algorithms (EAs) due to their suitability for modeling Natural evolution processes [4][22]. Although traditional EAs were essentially dedicated to solve static optimization problems in the past, several steps have been taken to adapt them to dynamic environments. These steps aim to enhance the performance of EAs to locate the moving optima in the landscape and avoid premature convergence. Among the most common approaches, we can mention *hyper-mutation* [11][15], *random immigrants* [27][31], and the use of *multiple populations* [1][8].

Developed EAs were tested on well-known DOPs: dynamic job shop scheduling problems [5][7], dynamic knapsack problems [12][23], dynamic traveling salesman problems [13][14], etc. The other way of evaluating EAs is to build up dynamic benchmark problems. Branke [6] developed the moving peaks benchmark problem which consists of a number of peaks changing in height, width, and location. Also, the XOR generator -introduced by Yang [29]- creates the dynamic counterpart of a given stationary problem via the bitwise exclusive-or operator. Another important test problem generator is the DF1 generator introduced by Morrison and De Jong [17].

H. Benromdhane
LARODEC Laboratory, ISG of Tunis, 41 Rue de la Liberté, Le Bardo, Tunisia
E-mail: hajer.ben.romdhan@hotmail.com

E. Alba
Universidad de Málaga, Boulevard Louis Pasteur s/n
E-mail: eat@lcc.uma.es

S. Krichen
FSJEG de Jendouba, Avenue de l'U.M.A , 8189 Jendouba, Tunisia
E-mail: saoussen.krichen@isg.rnu.tn

Likewise, a great deal of interest has been paid to develop appropriate metrics and statistical tests in order to measure and compare the proposed EAs. Some traditional performance measures from stationary problems have been transferred to DOPs (e.g. the *best-of-generation* and the *offline* fitness) and new metrics were designed recently for dynamic environments (e.g. the *area between curves* [2] and the *fitness degradation* [3]). However, the actual use of these performance measurement methods in literature is limited. Generally, the *best-of-generation* fitness is the most commonly reported measure. It averages the fitness of the best individuals in a given population over the number of evaluations, which indicates the best the EA can perform (upper bound for its numerical behavior). From a practical point of view, this measure responds to the aspirations of biological experts, among others, as they are concerned about the performance of the whole population [21]. Our claim here is that this measure is not accurate too often, and does not give insight about the final fitness achieved by the compared algorithms nor allow the statistical significance study of the results [16]. Besides, fitness-related measures do not serve as a meaningful tool in several real-world applications where the optimality is not the objective [18], or cannot be used at all [36]. Instead, the satisfaction of a given criteria (e.g. in agent control problems), the robustness of the solutions (e.g. in managing stock market portfolio), or the quick reaction after changes are examples of aspects that should be given priority over average best fitness metrics in many real-world optimizations problems [21].

This work seeks to throw some light on the importance of properly chosen performance measures to compare algorithms in a fluctuating environment. We will show, through an investigation over a number of algorithms, that evaluating two competing algorithms via an inappropriate tool could lead to false interpretations. We later use different metrics in our study to show in which cases they yield identical interpretations and when they contradict each other, and hence we finally show that they are complementary. We test three evolutionary algorithms, a local search hillclimber, and a random search algorithm over the dynamic knapsack problem for four test-beds of small, medium, and large size.

Our work is motivated by the following facts: (1) evolutionary algorithms are generally tested via a unique measure at a time, (2) few algorithms are subject to comparison in the same work, (3) the algorithms are run on too small instances, and (4) measurements are not statistically validated. We will try to show that all this can and should be done in every single paper to allow meaningful conclusions and avoid common pitfalls done in the past on static problems.

This paper is organized as follows. Section 2 will be devoted to define and draw the different algorithms. We present in Section 3 the experimental results and their analysis. Finally, we conclude in Section 4 and show several future lines of research.

2 Algorithms Designed to Solve DOPs

This section presents the different algorithms used in this study. We classify these algorithms into two categories: genetic algorithms for DOPs and non-standard algorithms adopted to DOPs. A full description as well as the relative pseudocode of the algorithms are reported in what follows.

2.1 Genetic Algorithms for Dynamic Environments

Genetic algorithms (GAs) have long been adopted to solve challenging DOPs. Since the first diploid GA [10] applied to the DKP, lots of efforts have been expended in studying GAs in order to enhance their performance when solving DOPs. Many researchers presented modified GAs using one or more of the following procedures: **1.** increase the diversity once a change occurs (*hyper-mutation* [11][15]), **2.** maintain population diversity throughout the evolutionary process (*random immigrants* [27][31]), **3.** introduce a memory to retrieve pertinent information from previous states (*memory scheme* [10][34]), **4.** head the search over different regions via multiple populations [1][8] and **5.** predict future changes in the landscape [25][26]. We will explain the algorithms that we are using in our study in the following sections.

```
begin
    $g \leftarrow 0$   /* Generation index */
    Initialize($P(g)$)   /* $P(g)$ is entirely randomly generated */
    Evaluate($P(g)$)
    while termination condition not satisfied do
        $g \leftarrow g+1$
        if Environmental change detected then
            Initialize($P(g)$)   /* Renew the entire population */
        else
            /* Select y individuals from P(g-1) */
            $P'(g) \leftarrow$ TournamentSelection($y$, $P(g-1)$)
            /* Cross individuals in $P'(g)$ & return the best offspring */
            $newIndiv \leftarrow$ Crossover($P'(g)$)
            Mutation($newIndiv$)
            Evaluation($newIndiv$)
            Replacement($newIndiv$, $P(g-1)$)
            $P(g) \leftarrow P(g-1)$
        end
    end
end
```

Algorithm 1: Steady State Genetic Algorithm

Steady State Genetic Algorithm. In its most basic version, a GA evolves a population of solutions via three main steps:

```
begin
    ⋮
    if Environmental change detected then
        $P(g) \leftarrow$ Evaluate($P(g-1)$)
    else
        $P(g-1) \leftarrow$ ReplaceFractionPop($P(g-1)$, $p_r$)
        $P'(g) \leftarrow$ TournamentSelection($y$, $P(g-1)$)
        ⋮
    end
end
```

Algorithm 2: Random Immigrants Algorithm

```
begin
    $p_m \leftarrow p_{im}$   /* The mutation rate is initialized to a small value */
    ⋮
    if Environmental change detected then
        Increase($p_m, p_{hm}$)   /* Increase the mutation rate */
    else
        ⋮
        Reset($p_m, p_{im}$)   /* Reset the mutation rate to its initial value */
    end
end
```

Algorithm 3: Hyper-mutation Algorithm

selection, reproduction (crossover and mutation), and replacement. Algorithm 1 shows the GA we use in this study. The algorithm begins by initializing the population via generating a number of -binary encoded- individuals, and evaluate them in a second step. Each iteration, the algorithm verifies if a change in the environment has occurred, and this can be done in different ways. Several detection schemes were studied over the years [22], among which we point out: assumption-based (the change points in the time are either known a priori or can be easily predicted), population-based (by evaluating the population), and sensor-based (via introducing appropriate detectors in the fitness landscape) detection. The one used here is the assumption-based. If a change is detected, the whole population is re-initialized randomly, otherwise y individuals are selected from the last-generation population and submitted to recombination in pairs (crossover). The best obtained offspring goes through a mutation step before being introduced into the population to substitute the worst individual (replacement). Thus, the new population ($P(g)$) is created and the algorithm re-iterates until a certain termination condition is satisfied. We note that in this algorithm -as well as in the subsequent ones-, only one individual is replicated each generation (one evaluation per generation).

Random Immigrants Algorithm (RI). It is a modified GA trying to narrow down the problem of convergence, and this is done through a generational replacement of some individuals ($p_r \times$ *pop-size* individuals, where p_r is the replacement rate) by randomly generated ones. We show in Algorithm 2 just a fraction of RI, in order to indicate where to include the replacement instruction in a GA. In fact, RI differs from the GA presented in Algorithm 1 by the evaluation step (only used in the case that the change detection strategy is not population-based and to be ignored otherwise), which substitutes the population re-initialization, and the individuals replacement instruction, which comes before the selection step. The RI used here is inspired by the one in [27], with the simple difference of the number of evaluations per generation.

Hyper-mutation Algorithm (HM). Another enhanced version of the GA which consists in increasing the mutation rate whenever an environmental change is detected, in order to enhance the population diversity. That is, HM operates like an usual GA and with a low mutation rate p_{im} until a change in the environment occurs. In this case, a high mutation rate p_{hm} (called adaptive mutation rate) is adopted in the current generation, and reset to the initial rate at the following one. The main instructions of HM are stated in Algorithm 3.

2.2 Non-Standard Algorithms Adapted to Solve Dynamic Optimization Problems

We presented -up to now- three algorithms known to be efficient in solving DOPs. In this section, we involve algorithms barely used in the dynamic domain of research, with the intention to compare their performance to that of the 'standard' algorithms mentioned previously. Two algorithms of different search modes are considered: an exploitation algorithm that tries to make the best use of the already identified solutions (a local search algorithm), and an exploration algorithm with pure randomness (a random search algorithm).

Local Search Algorithm (LS). This kind of algorithms explore the neighborhood of a given point in the landscape looking for a better solution. We propose a LS that performs the search starting from a population of solutions. After randomly generating the initial population, selection, reproduction, and replacement are steps to repeat until the stopping criteria is met or an environmental change is detected. In this last case, the population is renewed before resuming optimization. The selection step consists in choosing one solution for improvement. Then, this solution undergoes a series of bit inversions: one bit is reversed at a time to create a new individual. This processing yields a number of solutions equal to the number of bits of the initial solution. The last step -in the current generation- is the replacement of the worst individual in the population by the best among all the created solutions. Therefore, only one new individual is introduced in each generation. LS is shown in Algorithm 4.

```
begin
    $g \leftarrow 0$
    Initialize($P(g)$)
    Evaluate($P(g)$)
    while termination condition not satisfied do
        $g \leftarrow g+1$
        if Environmental change detected then
            Initialize($P(g)$)
        else
            $I \leftarrow$ TournamentSelection(1, $P(g-1)$)
            for $i = 0$ to I.length() do
                /* Inverse the $i^{th}$ bit of individual I */
                $I' \leftarrow$ BitInversion($i$, $I$)
                Evaluate($I'$)
                $P'(g)$.add($I'$)   /* Append $I'$ to $P'(g)$ */
            end
            Replacement(BestOf($P'(g)$), $P(g-1)$)
            $P(g) \leftarrow P(g-1)$
        end
    end
end
```

Algorithm 4: Local Search Algorithm

Random Search Algorithm (RS). These are a class of algorithms that use the concept of randomness when designing solutions. In this study, a RS operates as follows: the population is initialized randomly, and then each generation a new individual -randomly generated- replaces the poorest one in terms of performance. If an environmental change is detected, the population is re-initialized. Algorithm 5 draws the different steps of RS. One might think that RS is a straightforward algorithm easy to beat. However, the actual scenario is quite different: researchers should always explicitly compare to RS to show that this is true, as a kind of sanity check for the final claims achieved.

3 Performance Measures

In order to assess how well the considered algorithm meets the goals intended, a performance measure is needed. Although most works in the literature ask for optimality, there are other perspectives to consider when comparing DOPs algorithms. Most known performance measures fall into two categories: fitness-based measures and behavior-based measures. The following provides a brief overview of some measures from the above-mentioned categories. Both traditional measures of static environments and measures designed specifically to DOPs are considered in this study.

3.1 Fitness-based Performance Measures

Fitness-based performance measures are those appraising the performance of algorithms in terms of their closeness to the global optimum (either by finding or coming near to the best

```
begin
    $g \leftarrow 0$
    Initialize($P(g)$)
    Evaluate($P(g)$)
    while termination condition not satisfied do
        $g \leftarrow g+1$
        if Environmental change detected then
            Initialize($P(g)$)
        else
            $I \leftarrow$ RandIndividual  /*Generate a random individual*/
            Replacement($I$, $P(g-1)$)
            $P(g) \leftarrow P(g-1)$
        end
    end
end
```

Algorithm 5: Random Search Algorithm

fitness value). We look at some of the most known in what follows.

Average Best-of-generation

The *average best-of-generation* ($\overline{BOG}$) is the most often reported measure in dynamic environments [16]. This measure derives its importance from the fact that it covers the entire optimization process: i.e., it records the *best-of-generation* fitness over all G generations, and over the total number of runs Q. Formally, it is computed by double-averaging the best fitness obtained at each generation for all Q runs:

$$\overline{BOG} = \frac{1}{Q}\frac{1}{G} \cdot \sum_{q=1}^{Q}\sum_{g=1}^{G} f(BOG_{qg}) \tag{1}$$

where $f(BOG_{qg})$ expresses the fitness value of the best solution at generation g of run q (among Q independent runs). $\overline{BOG}$ takes values in the interval $[0, f^*]$, with f^* is the global optima. The weak side here is that this measure is not normalized and does not test the statistical significance of the difference between any two algorithms.

Offline Performance

As in dynamic environments the optimum changes periodically, it would be reasonable to compare EAs according to the best solutions they can achieve during each period. A period, in the dynamic environment terminology, is the time interval between two landscape changes. The *offline* performance is defined as the average, over a given number of periods, of the best solution found within the same period (see Eq. 2).

$$offline = \frac{1}{H} \cdot \sum_{k=1}^{H} f(BestSoFar_k) \tag{2}$$

We denote by $f(BestSoFar_k)$ the fitness of the best solution found so far (in the current period k). For each EA -involved in the comparison- this measure assigns a value (belonging to $[0, f^*]$) that can be regarded as its ability to cope with changes. It should be pointed out that the *offline* performance can only be applied if the change steps are known a priori.

Accuracy

The *accuracy* (also known as the *relative error*) is among measures that were initially designed for static environments and has been later adapted for the dynamic case [28]. This measure aims to determine where is located the best solution found inside the interval defined by a lower bound representing the worst known solution in the search space, and an upper bound that indicates the best known solution. The *accuracy* of an algorithm at a point in time t is the best found solution in the current period minus the worst known solution (Min_t) divided by the difference between the best (Max_t) and the worst known solutions in the landscape:

$$accuracy_t = \frac{f(BOG_t) - Min_t}{Max_t - Min_t} \tag{3}$$

The advantage of such a measure is that it minimizes the possible biases -caused by the difference of fitness at different periods- via a 'min-max normalization': it rescales the solution fitness from its initial range of values $[Min_t, Max_t]$ to $[0,1]$. The higher the obtained value, the better the EA performs. However, the computation of this measure is not always possible as the Max and the Min values are not necessarily available in practical situations. Besides, the formula in Eq. 3 yields only a value describing the quality of the solution at a given point in the landscape, so that this measure does not allow a comparison along the whole search process.

Area Between Curves

This is a newly proposed measure, introduced in [2], that provides a different way to compare algorithms performance in dynamic environments: it quantifies the distance between the performance curves of each pair of algorithms instead of measuring their closeness to the optimum. The *area between curves* (ABC) for a pair of algorithms A_1 and A_2 is the integral of the difference between two functions $p_{A_1(x)}$ and $p_{A_2(x)}$ over the total number of generations:

$$ABC_p^{A_1,A_2} = \frac{1}{G} \cdot \int_1^G p_{A_1}(x) - p_{A_2}(x)dx \tag{4}$$

where the $p_{A_i}(x)$, with $i \in \{1,2\}$, are to be replaced by a measure of population quality as $\overline{BOG}$, *offline*, etc. The ABC can assume both positive and negative values. A positive $ABC_p^{A_1,A_2}$ implies A_1 curve is higher than A_2 curve, and inversely. Measuring the area between performance curves can be a good alternative to decide which of two algorithms is better when the usual measures cannot distinguish between them.

3.2 Behavior-based Performance Measures

Behavior-based performance measures are the type of measures designed to control other behavioral aspects of EAs than the solutions fitness values. These aspects are of particular importance in dynamic environments as they reflect the algorithms abilities in terms of diversity, the quick adaptability, limiting the fitness degradation, etc. Despite their relevance, these measures are seldom reported in literature compared to the fitness-based measures. We focus on two measures of this category in this study: the *stability* measure and the *fitness degradation* measure.

Stability

The *stability* is a reflection of how much an algorithm is able to recover after a change. This measure is accuracy-dependent: an algorithm is said to be stable if it maintains its accuracy from a time step to the next one. The *stability* at a generation g is the difference between the current *accuracy* and the *accuracy* of the previous generation ($g-1$) if this difference yields a positive value, and zero otherwise.

$$stability_g = \max\{0, accuracy_g - accuracy_{g-1}\} \tag{5}$$

This measure lies in the range $[0,1]$: the closer to zero, the higher is the stability. We must however notice that a stable algorithm is not necessarily the best performing algorithm in terms of producing quality solutions.

Fitness Degradation Measure: $\beta_{degradation}$

Measuring the ability of an algorithm to track the moving optima was the topic of most measures discussed so far (in different ways). However, it is equally important to know how long this algorithm is able to keep following the optima. A recent study [3] has addressed the concern that the performance of an algorithm tends to degrade (in terms of fitness quality) as the number of changes becomes larger, and proposed the $\beta_{degradation}$ measure which quantifies that degradation over time. The $\beta_{degradation}$ is the slope of the regression line of the final accuracy values (recorded at the end of each period). The regression line is defined as:

$$u = \beta_{degradation}\, \overline{x} + \varepsilon \tag{6}$$

where u denotes the approximation of the total *accuracy*, $\overline{x}$ a vector of size equal to the total number of periods, and $\beta_{degradation}$ is the slope of the line. The final accuracy of each period k is given by:

$$x_k = \frac{1}{Q} \cdot \sum_{q=1}^{Q} f(BestSoFar_{qk}) \tag{7}$$

The loss of solution quality is expressed by a negative slope ($\beta_{degradation} < 0$), while a positive value implies the algorithm is able to compensate well after changes by producing high quality solutions.

4 Experimental Study

In this section, we examine the behavior of the algorithms identified so far according to several dynamic environment metrics. Our purpose is essentially to emphasize the relevance of properly choosing the measures of comparison: selecting appropriate measures that yield results in accordance with researchers conclusions and summarize the results in meaningful values. This is hopefully an exercise to illustrate new best practices in DOPs for future researchers. We begin this section with a brief overview about the benchmark problem used to evaluate the discussed algorithms: the 0-1 knapsack problem combined with the XOR-generator to create the dynamic knapsack problem. This method of including dynamism into the static knapsack problem has been largely used in the field of EAs to conduct experimentations (e.g. in [31][32][33]). The third part of this section is devoted to the experimental settings, followed by the numerical results, and their analysis.

4.1 Test Problem: the Dynamic Knapsack Problem

In order to address the purpose of this study, a test problem is required. We use the well-known 'dynamic knapsack problem' (DKP) to conduct our experimentations, as being one of the most widely employed benchmarks for testing new algorithms targeted to dynamic environments. It draws its importance from several facts including (but not limited to): it is one of the very few constrained benchmark problems [24], it has been extensively addressed both in its static and dynamic versions with a panoply of penalty functions and repair algorithms [24], and it is not restricted to a specific problem but rather satisfies a large range of real applications in industry [24][35] (e.g. cutting stock problem, selecting projects to fund, and cargo loading). This problem, referred in some publications to as 'time-varying knapsack problem', was first studied in [10], and has been investigated by several researchers since then.

The DKP seeks to fill a limited capacity knapsack with the best subset of items among a larger set, with the aim of maximizing the value of the knapsack contents. The dynamic character of this problem is gained when the items weights, the items values, and/or the knapsack capacity become time-dependent: some (at least) of these parameters are subject to variations each period. Formally, a DKP is stated as follows.

Given a set of n items, each of which has a weight $w_i(t)$ and a value $v_i(t)$, and a knapsack of capacity $C(t)$, the DKP is to load the best subset which guarantees the maximization of the total value of the loaded items without violating the capacity constraint:

$$\begin{array}{llll} \text{Max} & Z(x,t) & = & \sum_{i=1}^{n} v_i(t)x_i(t) \\ \text{s.t.} & \sum_{i=1}^{n} w_i(t)x_i(t) & \leq & C(t) \\ & x_i(t) & \in & \{0,1\}, \qquad i=1,\dots,n \end{array} \tag{8}$$

Solutions for this problem are binary-encoded, that is: $x_1(t)x_2(t)...x_n(t)$ where $x_i(t)=1$ means that item i is chosen at t, and $x_i(t)=0$ means it is discarded.

In this work, the solutions are evaluated as proposed in [33]: the fitness of a given solution x is computed as the profit sum of the selected items ($F(x,t)=Z(x,t)$) if the solution does not exceed the capacity of the knapsack (as in Eq. 8), otherwise the fitness is set to a small value computed based on the sum of all items' weight and the weight of the solution x:

$$F(x,t) = Z_2(x,t) = 10^{-10} \times (\sum_{i=1}^{n} w_i - \sum_{i=1}^{n} w_i x_i) \tag{9}$$

4.2 Dynamic Problem Generator: the XOR Generator

The XOR generator is a dynamic problem generator for dynamic environments. It yields the dynamic counterpart of any static binary-encoded problem, for instance the DKP [31], the one-max problem [30], and others [33]. The justification of interest and globality of this DOP generator is that it keeps the optimum value unchanged and, alternatively, moves the population (through the displacement of its individuals to new positions in the landscape) after each change [18].

Given a static problem $F(x)$, the XOR generator constructs its dynamic version $F(x,t)$ through the following evaluation function:

$$F(x,t) = F(x \oplus M(k)) \tag{10}$$

where the bitwise exclusive-or operator '$\oplus$' is applied to the solution x and $M(k)$ according to the following principle:

$$x_i \oplus x_j = \begin{cases} 0 \text{ if } & x_i = x_j \\ 1 \text{ otherwise} \end{cases} ; \quad x_i, x_j \in \{0,1\} \tag{11}$$

The population is evaluated thereafter according to the given fitness function (as described in section 4.1). We note $M(k)$ the mask at period k: a binary chromosome of length n, updated every new period on the basis of the mask of the previous period $M(k-1)$ and the intermediate template $T(k)$,

$$M(k) = M(k-1) \oplus T(k), \tag{12}$$

with $T(k)$ a binary string generated randomly and made up of $\rho \times n$ ones. The period index k is given by $k = \lfloor t/\tau \rfloor$ and the initial mask $M(0)$ is a zero vector. The environmental dynamics are adjusted by two parameters: the speed of change ($\tau \in [1,G]$) and the severity of change ($\rho \in [0.0, 1.0]$).

The XOR generator could furthermore create different types of dynamic environments: noncyclic, cyclic, and partially cyclic. The DKP we consider here involves a cyclic environment in which the system moves among a number of base states in a given order (two states are repeatedly encountered in our case). The steps to follow to create such environment are: given the number of base states $2K$, an equal number of masks is created, then at any time point t of the evolutionary process the solution x is evaluated using the formula:

$$F(x,t) = F(x \oplus M(k\%(2K))) \tag{13}$$

with $k\%(2K)$ denotes the index of the current base state. Masks are created the same way as described above. The reader is referred to [31] for more details.

4.3 Settings of the Experimentation

The experimental design adopted in this work is described in what follows. We run the five algorithms on four DKP instances of different sizes: 17, 100, 500, and 1000-item.

Table 1: Optimal fitness for the utilized instances

Problem size	**Fitness value**
17	87
100	47 254
500	230 502
1000	486 678

Except for the small instance of 17-item used here for leading literature comparison [10], we use the test instances generator of Pisinger [20] to generate the parameters of static KPs. We set the capacity value C of the 17-item problem to 100 (one of the two values predefined by the authors in [10]) and to 75% of the sum of all items' weight for the rest of problems. Table 1 contains the global optima for the static phases of the studied KPs. Optima were generated by the exact algorithm described in [19]. We define optimality as the highest value that the objective function can have, while respecting the capacity constraint. Therefore, we consider the theoretical optima fitness values as numerical goals inside each static period.

The XOR generator is applied thereafter to these instances (as described in section 4.2) to construct dynamic test problems. The stopping criteria is the maximum number of generations. Knowing that the XOR generator creates the dynamism based on the speed of change, we assume that dates of change are known. That is, every τ generations a new environment is met. Different number of environmental changes were allowed for each instance: 30 changes in the case of the two smallest instances (size 17 and 100), 100 changes for the instance with size 500, and 300 changes in the case of the biggest instance. We note that the algorithms were allowed to run for longer times on the biggest instances in order to be sufficiently exposed to the problem dynamics. In the experiments, 30 independent runs were performed. Table 2 reports the parameterization of the above mentioned algorithms.

Table 2: Set of configuration parameters

Algorithms parameters	
Population size	100 individuals
Selection	Binary tournament
Crossover	Two point crossover ($p_c = 1$)
Mutation	Two-bit flip ($p_{im} = \frac{2}{n}$ & $p_{hm} = 0.5$)
Replacement strategy	The worst in the current population
Replacement rate	$p_r = 0.3^*$
Dynamism parameters	
Change detection	generations count
Severity of change	$\rho = 0.6$
Rate of change	$\tau = 10$ generations
Number of states	$2K = 2$

* We use the p_r value that gave the better tracking performance results according to [11].

4.4 Results Discussion

The remainder of this section is concerned by the experimental results and their analysis. Results were carried out to compare the five selected algorithms according to several performance measures for dynamic environments. Besides, we performed a statistical significance test to assess whether the algorithms are significantly different in their performance. The results of the significance test and those of the performance measures are reported and summarized briefly (in terms of comparison of algorithms performance) in Tables 3-5, and analyzed measure by measure in the ensuing paragraphs.

Statistical significance test. All results in this article are statistically validated. This is seldom done in dynamic problem solving papers, especially because authors just report figures (visual inspection) and not numerical data. We think it is a must: we do not need to go for another twenty years as we did in static problem solving to realize that non deterministic algorithms need a statistical validation study in every single paper on them, also in DOPs.

As statistical significance test we used the ANOVA test with a confidence level of 95%. Because this test requires the normality of the data, we apply first a Kolmogorov-Smirnov test to check whether the results follow a normal distribution. If it is the case, an ANOVA test is applied, otherwise we will perform a Kruskal-Wallis test.

Additionally, we performed a statistical significance test using the *final fitness* as the base measure: the best fitness reached at the last generation of each of the 30 runs.

Let us start with the analytical study of algorithms to show the comparative advantages in concluding with data what researchers would conclude themselves after experiencing with the problems. A first conclusion from Table 3 shows that HM is significantly better than all the others except for the problem of size 17. In addition, LS performs better than

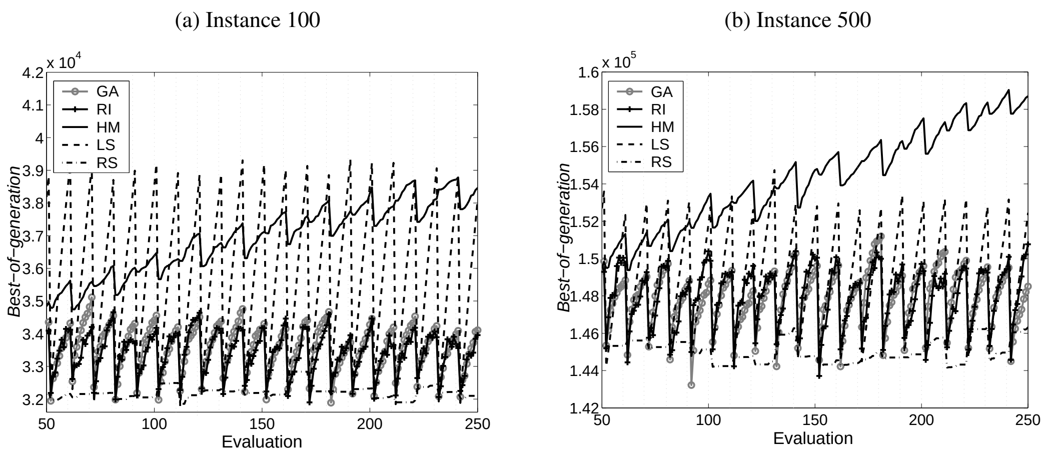

Fig. 1: Comparison of the five algorithm performance on $\overline{BOG}$ with n=100 and n=500*

* The vertical dotted lines in this figure (as well as in the subsequent ones) indicate the start of a new period.

GA, RI, and RS in most cases (11 out of 12) and outperforms HM in the smallest instance.

Table 3: ANOVA test results

	n	**GA**	**RI**	**HM**	**LS**
RS	17	▽	–	▽	▽
	100	▽	▽	▽	▽
	500	▽	▽	▽	▽
	1000	▽	▽	▽	▽
LS	17	▲	▲	▲	
	100	▲	▲	–	
	500	▲	▲	▽	
	1000	▲	–	▽	
HM	17	–	▲		
	100	▲	▲		
	500	▲	▲		
	1000	▲	▲		
RI	17	▽			
	100	–			
	500	–			
	1000	–			

▲, ▽, and – mean respectively: the algorithm in row is significantly better than, significantly worse than, or statistically equivalent to the algorithm in column.

Average Best-of-generation. Among the five competing algorithms, LS and HM are the most interesting ones: while LS outperforms with the smallest instance, HM shows a better performance with the rest. Fig. 1 draws 200 evaluations of each algorithm behavior across the landscape dynamics for the two intermediate problem sizes (100 and 500-item). The performance curves show that HM outperforms LS after a considerable number of evaluations according to the first problem of size 100, but after just 130 evaluations for the second instance. GA and RI revealed a very similar behavior throughout the number of evaluations and for all problem sizes: they showed a significant similarity through $\overline{BOG}$ values (very close values) as well as the running BOG plotted in Fig. 1 (curves cannot be identified clearly). The algorithm with the lowest curve is RS, which kept the poorest performance with all instances as expected.

Despite the fact that this figure shows us the whole picture of the algorithms performance, it is difficult to draw out conclusions about the final result of the compared algorithms by visual inspection of these curves. So let us see if the numerical values of metrics confirm what we (hardly) see in figures.

Offline. This measure does not confirm the previous interpretations (those of the $\overline{BOG}$ measure) with regards to HM and LS: the *offline* fitness shows that LS gives better *offline* performance with the instances 17 and 100, and HM with the two others. Fig. 2 illustrates the *offline* fitness for instances 100 and 500 (just the first 50 periods for the sake of clarity), averaged across the number of runs.

Accuracy. The computation of this measure requires the Min_t and Max_t values, which we set respectively to 0 and the fitness value of the optima at t. From Table 4 we can see that all the studied algorithms fulfil a very good performance according to the small instance (more than 80%), and accuracy values begin to decrease as the problem size grows up. No further information can be induced from these results except the fact that they support the previous interpretations which show that HM is the most efficient in most cases.

Area Between Curves. The ABC measure is applied to each pair of algorithms according to the $\overline{BOG}$ property ($p = \overline{BOG}$). The results obtained for each pair of algorithms can be seen in Table 5. Aside from RS, which received negative ABC

Table 4: Numerical comparison of GA, RI, HM, LS, and RS with respect to different performance measures

	Metric	n	GA	RI	HM	LS	RS	Summary[*]
Fitness-based	$\overline{BOG}$	17	74.4	73.7	77.4	**79.9**[**]	72.6	HM > LS > GA ≃ RI ≫ RS
		100	33 475.1	33 402.6	**36 613.7**	35 592.2	32 217.9	
		500	147 912.0	148 110.6	**164 980.8**	148 938.8	145 055.3	
		1000	281 599.9	281 770.7	**323 439.4**	281 416.3	277 586.5	
	Offline	17	76.0	74.3	78.2	**82.1**	72.7	HM ≃ LS ≫ GA ≃ RI ≫ RS
		100	34 289.5	34 110.2	37 004.8	**38 746.2**	32 253.6	
		500	149 472.1	149 736.2	**165 669.1**	152 909.9	145 143.4	
		1000	283 612.2	283 912.8	**324 137.4**	285 438.9	277 719.5	
	Accuracy	17	8.5e-01	8.5e-01	8.9e-01	**9.2e-01**	8.3e-01	HM > LS > GA ≡ RI ≫ RS
		100	7.1e-01	7.1e-01	**7.7e-01**	7.5e-01	6.8e-01	
		500	6.4e-01	6.4e-01	**7.1e-01**	6.5e-01	6.3e-01	
		1000	5.8e-01	5.8e-01	**6.6e-01**	5.8e-01	5.7e-01	
Behavior-based	*Stability*	17	4.1e-01	3.7e-03	1.8e-03	1.1e-03	**6.5e-04**	RS ≫ HM ≫ GA > LS ≃ RI
		100	4.5e-03	4.4e-03	1.6e-03	1.4e-02	**3.4e-04**	
		500	1.9e-03	2.1e-01	5.6e-04	3.4e-03	**1.6e-04**	
		1000	1.3e-03	1.6e-03	7.7e-04	1.6e-03	**1.5e-04**	
	$\beta_{degradation}$	17	3.0e-04	-2.1e-04	**1.1e-03**	4.7e-04	2.1e-05	HM ≫ LS > GA > RS ≃ RI
		100	1.3e-04	8.3e-05	**3.8e-03**	9.8e-04	-1.1e-05	
		500	2.8e-05	3.2e-05	**1.4e-03**	1.5e-05	4.8e-06	
		1000	-5.9e-07	-8.6e-07	**5.3e-03**	1.7e-06	2.4e-06	

[*] Results of each measure are summarized by the following symbols: "$A_1 > A_2$", "$A_1 \simeq A_2$", "$A_1 \gg A_2$", and "$A_1 \equiv A_2$" which indicate respectively A_1 outperforms A_2 in most cases, A_1 and A_2 outperforms each other in an equal number of instances, A_1 outperforms A_2 in all cases, and A_1 and A_2 perform equivalently.

[**] Boldface values indicate the algorithm with the best performance: we note that these results are validated through statistical significance tests (ANOVA).

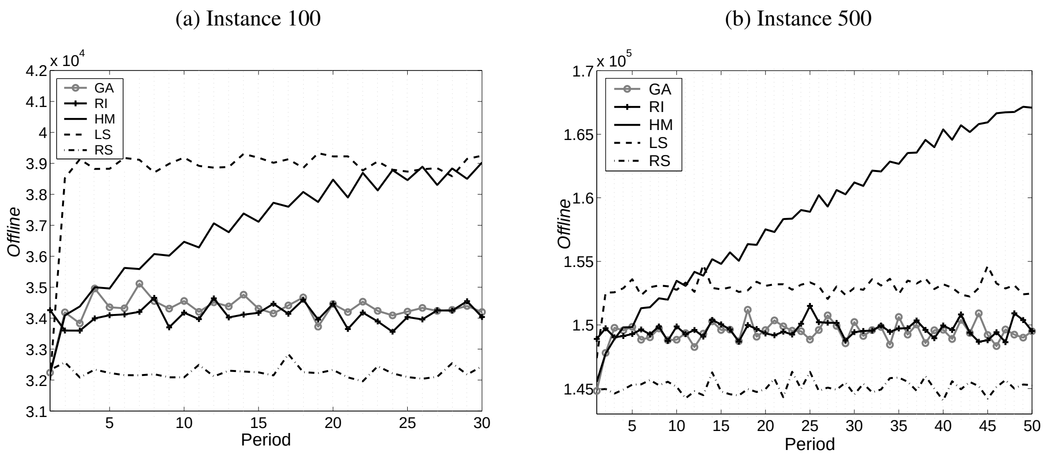

Fig. 2: Average *offline* fitness along the time-period obtained via GA, RI, HM, LS, and RS for n=100 and n=500

values with the four algorithms for all instances, the rest of algorithms take positive and negative values with regard to which algorithm they are compared to. This measure leads towards the same previous interpretations:

- RS rows are entirely negative, which means that RS curves always lie under the other curves.
- *ABC* values achieved by LS and HM are the largest ones, that is LS and HM are considerably distant from the rest of algorithms. Since a researcher could easily conclude that they two are the best algorithms, it seems that ABC is a very appropriate measure to quantify what human experts see only qualitatively.

– HM obtains positive *ABC*s when compared to LS in three out of four instances: HM performs better than LS with the large instances.
– the smallest *ABC* values were those of the pair (GA, RI), which implies a low distance between the algorithms in terms of performance: GA and RI performs similarly.

Table 5: Comparisons of $ABC_{BOG}^{A_1,A_2}$ values achieved by each pair of algorithms (A_1 in row and A_2 in column)

	n	**GA**	**RI**	**HM**	**LS**
RS	17	-1.8e+00	-1.2e+00	-4.8e+00	-7.3e+00
	100	-1.2e+03	-1.2e+03	-4.4e+03	-3.4e+03
	500	-2.8e+03	-3.0e+03	-2.0e+04	-3.9e+03
	1000	-4.0e+03	-4.2e+03	-4.6e+04	-3.8e+03
LS	17	**5.5e+00**	**6.1e+00**	**2.5e+00**	
	100	**2.1e+03**	**2.2e+03**	-1.0e+03	
	500	**1.0e+03**	**8.3e+02**	-1.6e+04	
	1000	-1.8e+02	-3.5e+02	-4.2e+04	
HM	17	**2.9e+00**	**3.6e+00**		
	100	**3.1e+03**	**3.2e+03**		
	500	**1.7e+04**	**1.7e+04**		
	1000	**4.2e+04**	**4.2e+04**		
RI	17	-5.9e-01			
	100	-7.2e+01			
	500	**2.0e+02**			
	1000	**1.7e+02**			

Significant values (>0.05) are boldfaced, otherwise the difference is considered to be negligible.

Stability. As a low *stability* value implies a fast adaptation to change, the results carried out using the *stability* measure (shown in Table 4) can be interpreted as follows: RS is the most stable, HM comes second in terms of stability, and LS and RI are the most unstable. The most surprising finding is that the previously reported worst performing algorithm -in terms of solutions quality- is now the most stable and then the fastest to react to change. This bear out what the authors in [3] already claimed about the independence between the stability behavior of an algorithm and the quality of its solutions. Indeed, this can be explained by the fact that poor performing algorithms will have normally low *accuracy* values which are not that hard to re-attain after the environment changes, but this is not always possible in the opposite case (with high *accuracy*). But whatever is the reason, the *stability* measure has revealed a weak side of good solutions quality algorithms (as LS), and the strength of the other algorithms. Subsequently, it is up to practitioners to decide which EA suits their expectations. For instance, a vehicle routing modeler would prefer a stable algorithm that guarantees a non-optimum but useful performance level over a high performing algorithm that can cause performance degradation after running for a long time [36].

Fitness degradation. Results of the β slope are presented in Table 4 and algorithms regression lines (instance of size 500) are drawn in Fig. 3. HM and LS provide positive slopes for all instances, which means a non-degradation in the solutions quality. Indeed, HM values are the larger for all instances (the highest slopes in Fig. 3 (c)) and the most steady, what means that HM is the most powerful in terms of continuous adaptation. Results also show that GA loses quality with the largest instance, RI degrades in terms of performance in two of the four instances, and RS shows a declining slope with the problem of size 100.

The results of this measure can be summarized in three facts:

– Although the previous results prove that all the competing algorithms are better than RS, $\beta_{degradation}$ results showed that RS provides positive slopes whereas other algorithms obtained negative slopes. This is also expected, since continuous adaptation is a priori feature of random search. Not a good adaptation usually, but a steady adaptation in time.
– HM shows its actual behavior when measured with the fitness degradation masure and compared to LS.
– GA is better than RI (higher slopes) in most instances.

To sum up, three main facts deserve to be mentioned. First, all fitness-based measures are in agreement upon the two best performing algorithms, HM and LS, and the worst one, RS. Second, the behavioral measure *stability* gave rise to unexpected results: RS is the most able to adapt after changes, while LS showed a low stability. Third, the measure of fitness degradation over time indicates that HM is the best in keeping track of the optimum in all the studied cases, and RS is better than RI for two of the four instances.

Accordingly, we can say that the analysis of the behavior-based measures results revealed interpretations different from those propounded by the fitness-based measures: the highest solutions quality algorithm can no longer be considered to actually represent any behavioral aspect of an algorithm. Other metrics are needed to measure the quick adaptation and the ability to produce good solutions for a long time. As there is no perfect algorithm, all these interpretations are to be taken into consideration. The first type of measures (fitness-based) does not allow to decide which of the two best algorithms (HM and LS) is better, but the $\beta_{degradation}$ helps to judge. This, since HM slopes are always higher than the slopes of LS. So $\beta_{degradation}$ is a very needed metric in actual works with DOPs. Besides, *stability* measure showed a high adaptation ability of an algorithm that it is expected to be the worst in dynamic environments (RS), so *stability* should be used only with careful justification. These two facts, among

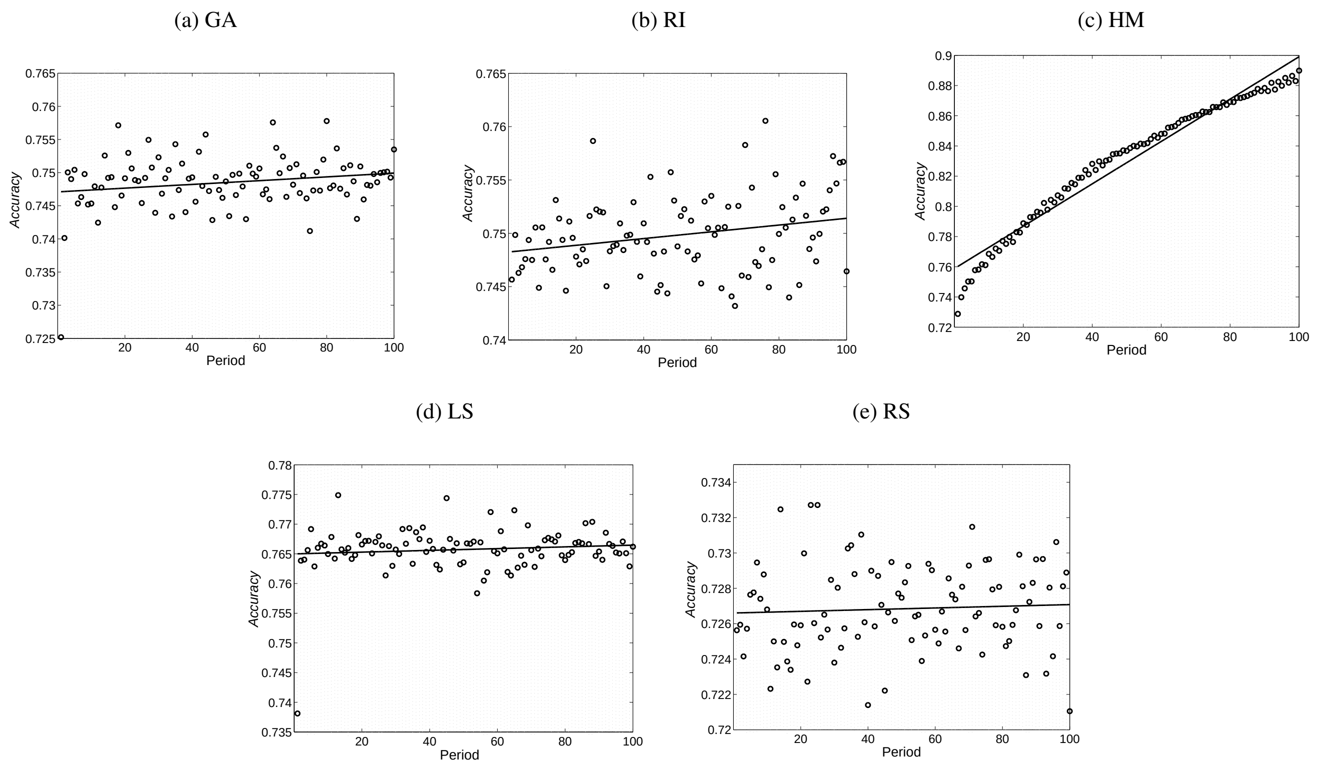

Fig. 3: $\beta_{degradation}$ after 100 periods for GA, RI, HM, LS, and RS (from top left to bottom right, respectively)

others, lead us to the conclusion that every measurement is important and can provide a further information about the studied algorithm. Fitness-based measures and behavior-based measures are complementary and should be used in every work in addition to the statistical significance to assess the quality of the proposed approaches. These are the basic metrics that will allow future meaningful and reproductible works in DOPs in the future.

5 Conclusion

In this paper we focused on performance measures in dynamic environments. We have begun by applying classical metrics ($\overline{BOG}$, *offline*, and *accuracy*) which yielded identical interpretations for the compared algorithms (no conclusion, then), and when we tested via a behavior-based measure (*stability*), it turned out that even algorithms poor in terms of solutions quality excel with respect to it. The use of the second behavioral measure also presented the compared algorithms differently from what the rest of measures did. This evokes the importance of using both types of measures together to get meaningful conclusions.

We have here also a clear comparative analysis of well-known algorithms. Like many other similar studies, we can draw conclusions on their relative interested for DOPs. Despite the fact that LS outperforms HM with the smallest instance and provides good performance in the rest, HM performs better with large problems. Besides to the fitness function, the scalability is of an important consideration when comparing DOPs. Therefore, we can conclude that HM is the best choice among the five competing algorithms. We have showed that we would not be able to achieve this conclusion without making recourse to different and complementary performance measures because of the many issues we need to evaluate in DOPs.

In short, best practices require first using both, fitness and behavioral measures. Second, report numerical data and aside statistical assessment. Finally, taking care of scalability in the problem is the third important step to level up results in this area and create a body of knowledge in time.

As a future work, we want to create a methodology for evaluating DOPs allowing researchers to easily compare against the state of the art and thus concentrate in ideas on algorithms and problems. Moreover, we aim at improving the results of HM found in this paper with DKP to other problems and try to advance to a state of the art set of techniques that, at present, are unclear in their performances.

Acknowledgements
Authors acknowledge funds from the CICE of the Junta de Andalucia, under contract P07-TIC-03044 (DIRICOM http://diricom.lcc.uma.es) and Spanish Ministry of Sciences and Innovation (MICINN) and FEDER under contracts TIN2011-28194 (RoadMe http://roadme.lcc.uma.es)

and TIN2008-06491-C04-01 (M* http://mstar.lcc.uma.es). Also, from the European COADVISE project number 230833.

References

1. Alba, E.: Parallel Metaheuristics. John Wiley & Sons, Inc. (2005)
2. Alba, E., Sarasola, B.: Abc, a new performance tool for algorithms solving dynamic optimization problems. In: IEEE Congress on Evolutionary Computation, pp. 1–7 (2010)
3. Alba, E., Sarasola, B.: Measuring fitness degradation in dynamic optimization problems. In: Applications of Evolutionary Computation, *Lecture Notes in Computer Science*, vol. 6024, pp. 572–581. Springer Berlin / Heidelberg (2010)
4. Back, T.: On the behavior of evolutionary algorithms in dynamic environments. In: The 1998 IEEE International Conference on Evolutionary Computation Proceedings, pp. 446 – 451 (1998)
5. Bierwirth, C., Mattfeld, D.: Production scheduling and rescheduling with genetic algorithms. Evolutionary Computation **7**(1), 1–17 (1999)
6. Branke, J.: Memory enhanced evolutionary algorithms for changing optimization problems. In: Proceedings of the 1999 Congress on Evolutionary Computation, vol. 3, pp. 1875–1882. IEEE (1999)
7. Branke, J., Mattfeld, D.C.: Anticipation in dynamic optimization: The scheduling case. In: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature, pp. 253–262 (2000)
8. Cheng, H., Yang, S.: Multi-population genetic algorithms with immigrants scheme for dynamic shortest path routing problems in mobile ad hoc networks. In: Proceedings of the 2010 international conference on Applications of Evolutionary Computation, pp. 562–571. Springer-Verlag (2010)
9. Cruz, C., González, J., Pelta, D.: Optimization in dynamic environments: a survey on problems, methods and measures. Soft Computing **15**(7), 1427–1448 (2011)
10. Goldberg, D.E., Smith, R.E.: Nonstationary function optimization using genetic algorithms with dominance and diploidy. In: Proceedings of the Second International Conference on Genetic Algorithms and their Application, pp. 59–68 (1987)
11. Grefenstette, J.: Genetic algorithms for changing environments. In: Parallel Problem Solving from Nature 2, pp. 137–144. Elsevier (1992)
12. Hadad, B.S., Eick, C.F.: Supporting polyploidy in genetic algorithms using dominance vectors. In: Evolutionary Programming VI, *Lecture Notes in Computer Science*, vol. 1213, pp. 223–234. Springer Berlin / Heidelberg (1997)
13. Li, C., Yang, M., Kang, L.: A new approach to solving dynamic traveling salesman problems. In: Proceedings of the 6th International Conference on Simulated Evolution and Learning, pp. 236–243 (2006)
14. Mavrovouniotis, M., Yang, S.: A memetic ant colony optimization algorithm for the dynamic travelling salesman problem. Soft Computing **15**, 1405–1425 (2011)
15. Morrison, R., De Jong, K.: Triggered hypermutation revisited. In: Proceedings of the 2000 Congress on Evolutionary Computation, vol. 2, pp. 1025–1032 (2000)
16. Morrison, R.W.: Performance measurement in dynamic environments. In: A.M. Barry (ed.) Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference, pp. 99–102. AAAI, Chigaco (2003)
17. Morrison, R.W., De Jong, K.A.: A test problem generator for non-stationary environments. In: Proceedings of the 1999 Congress on Evolutionary Computation, pp. 2047–2053 (1999)
18. Nguyen, T.T., Yang, S., Branke, J.: Evolutionary dynamic optimization: A survey of the state of the art. Swarm and Evolutionary Computation **6**(0), 1 – 24 (2012)
19. Pisinger, D.: A minimal algorithm for the 0-1 knapsack problem. Operations Research **45**(5), 758–767 (1997)
20. Pisinger, D.: Core problems in knapsack algorithms. Operations Research **47**, 570–575 (1999)
21. 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: GECCO Workshops, pp. 32–38 (2005)
22. Richter, H.: Detecting change in dynamic fitness landscapes. In: Proceedings of the Eleventh conference on Congress on Evolutionary Computation, pp. 1613–1620. IEEE Press (2009)
23. Rohlfshagen, P., Bullinaria, J.: Alternative splicing in evolutionary computation: Adaptation in dynamic environments. In: Proceedings of the 2006 IEEE Congress on Evolutionary Computation, pp. 2277–2284 (2006)
24. Rohlfshagen, P., Yao, X.: The dynamic knapsack problem revisited: A new benchmark problem for dynamic combinatorial optimisation. In: Applications of Evolutionary Computing, *Lecture Notes in Computer Science*, vol. 5484, pp. 745–754 (2009)
25. Simões, A., Costa, E.: Evolutionary algorithms for dynamic environments: Prediction using linear regression and markov chains. In: Parallel Problem Solving from Nature PPSN X, *Lecture Notes in Computer Science*, vol. 5199, pp. 306–315. Springer Berlin / Heidelberg (2008)
26. Simões, A., Costa, E.: Improving prediction in evolutionary algorithms for dynamic environments. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, GECCO '09, pp. 875–882. ACM (2009)
27. Tinós, R., Yang, S.: A self-organizing random immigrants genetic algorithm for dynamic optimization problems. Genetic Programming and Evolvable Machines **8**, 255–286 (2007)
28. Weicker, K.: Performance measures for dynamic environments. In: Parallel Problem Solving from NaturePPSN VII, pp. 64–73. SpringerVerlag (2002)
29. Yang, S.: Non-stationary problem optimization using the primal-dual genetic algorithm. In: Proc. of the 2003 Congr. on Evol. Comput, pp. 2246–2253 (2003)
30. Yang, S.: Non-stationary problem optimization using the primal-dual genetic algorithm. In: Proceedings of the 2003 Congress on Evolutionary Computation, vol. 3, pp. 2246–2253 (2003)
31. Yang, S.: Genetic algorithms with memory-and elitism-based immigrants in dynamic environments. Evolutionary Computation **16**, 385–416 (2008)
32. Yang, S., Tinós, R.: A hybrid immigrants scheme for genetic algorithms in dynamic environments. International Journal of Automation and Computing **4**, 243–254 (2007)
33. Yang, S., Yao, X.: Experimental study on population-based incremental learning algorithms for dynamic optimization problems. Soft Computing **9**, 815–834 (2005)
34. Yang, S., Yao, X.: Population-based incremental learning with associative memory for dynamic environments. Evolutionary Computation, IEEE Transactions on **12**(5), 542–561 (2008)
35. Younes, A., Calamai, P., Basir, O.: Generalized benchmark generation for dynamic combinatorial problems. In: Proceedings of the 2005 workshops on Genetic and evolutionary computation, GECCO '05, pp. 25–31 (2005)
36. Yu, X., Jin, Y., Tang, K., Yao, X.: Robust optimization over time - a new perspective on dynamic optimization problems. In: IEEE Congress on Evolutionary Computation, pp. 1–6. IEEE (2010)