Soft Comput (2014) 18:1471–1497
DOI 10.1007/s00500-013-1154-z

METHODOLOGIES AND APPLICATION

Prediction in evolutionary algorithms for dynamic environments

Anabela Simões · Ernesto Costa

Published online: 20 October 2013

Abstract Evolutionary algorithms have been widely used to solve dynamic optimization problems. Memory-based evolutionary algorithms are often used when the dynamics of the environment follow some repeated behavior. Over the last few years, the use of prediction mechanisms combined with memory has been explored. These prediction techniques are used to avoid the decrease of the algorithm's performance when a change occurs. This paper investigates the use of prediction methods in memory-based evolutionary algorithms for two distinct situations: to predict *when* the next change will happen and *how* the environment will change. For the first predictor two techniques are explored, one based on linear regression and another supported by nonlinear regression. For the second, a technique based on Markov chains is explored. Several experiments were carried out using different types of dynamics in two benchmark problems. Experimental results show that the incorporation of the proposed prediction techniques efficiently improves the performance of evolutionary algorithms in dynamic optimization problems.

Keywords Evolutionary algorithms · Dynamic environments · Prediction · Linear regression · Nonlinear regression · Markov chains

Communicated by Y.-S. Ong.

A. Simões (✉)
Department of Computer and Systems Engineering,
Coimbra Institute of Engineering, Polytechnic of Coimbra,
Rua Pedro Nunes-Quinta da Nora, 3030-199 Coimbra, Portugal
e-mail: abs@isec.pt

E. Costa
Centre for Informatics and Systems, University of Coimbra-Polo II,
3030-290 Coimbra, Portugal
e-mail: ernesto@dei.uc.pt

1 Introduction

Evolutionary algorithms (EAs) have been successfully used in a wide area of applications. Traditionally, EAs are well suited to solve problems where the environment is static. For dynamic optimization problems, an effective EA must be able to detect changes and rapidly deal with that changes when they occur. Classical EAs are not suited for these kinds of problems, since they have the tendency to prematurely converge to a solution. Therefore, when the conditions of the environment change, the algorithm needs some time for this converged population to readapt and move towards the new optimum.

To deal with these limitations, some improvements have been proposed as extensions of the classical EA. These improvements include maintaining diversity, e.g. Cobb and Grefenstette (1993), Grefenstette (1992), using memory schemes, e.g. Karaman et al. (2005), Yang (2007), Simões and Costa (2007b), Barlow and Smith (2008), using multi-populations, e.g. Ursem (2000), Younes et al. (2006), Li and Yang (2012) or anticipating the changes in the environment, e.g. Bosman and La Poutré (2007), Simões and Costa (2008, 2009b), Richter and Yang (2009).

If the environment changes cyclically and past situations reappear later, the use of memory is beneficial, since that the memorized solutions can be used to help the EA when a change in the environment happens and a previous situation reappears. In general, memory-based approaches dealing with dynamic environments use the memory *after* a change is detected. As a consequence, the EA's performance decreases after the change and then, when the memory is used, it starts increasing. In order to minimize the effects of the environmental changes on the EA's performance, anticipation mechanisms have been explored by different authors. The goal of these approaches is to use past data to forecast future situ-

ations and use it to help the EA to deal with the new environment. The next section describes the most relevant works concerning these issues. The majority of these methods use prediction to estimate where the optimum will move, giving the algorithm the moment of the next change. Nevertheless, in real situations, the time when the next change will happen is also unknown. In our previous work, we proposed the use of a new prediction techniques in memory-based EAs in order to estimate both the time of the next change and the next environmental transition, with promising results. Simões and Costa (2008) mechanisms using linear regression and Markov chains, are used to estimate the generation when a change in the environment will occur, and also to predict to which state (or states) the environment may change, respectively. Simões and Costa (2009a), we investigate different methods to dynamically adjust the linear predictor in order to achieve higher adaptability and robustness. Simões and Costa (2009b) a new approach based a nonlinear regression predictor is proposed to estimate when the next change will occur. In this paper we unify and further extend the use of these techniques in memory-based EAs. The previously proposed approaches presented several limitations and failed in specific situations. In particular, the linear regression predictor used all available data to make the next prediction. Consequently, as time passed, the computational effort was bigger and if the dynamics of the change period was altered the performance of the EA was compromised. To overcome this problem in this paper we introduce the concept of *time window* that greatly enhances the linear predictor module, making it more efficient, robust and applicable to a wider set of situations. This new mechanism significantly reduces the computational effort of the previous versions and presents the best performance in the studied benchmark problems. Moreover, we enlarge the experimental setup with new test functions dealing with different and more challenging scenarios. Finally, we present an unified and complete description of the framework, that includes our memory-based evolutionary algorithm, the prediction module (linear and nonlinear) and the anticipation module.

The rest of the paper is outlined as follows. The next section briefly reviews relevant work concerning prediction in dynamic environments. Section 3 gives an overview about different categorizations of dynamic environments, introducing a new one that is used in this work. Section 4 presents the memory-based EA used in this study. The prediction methods introduced in Simões and Costa (2008, 2009a,b) and the new improvements are described in Sect. 5. The used benchmarks, the experimental settings and the performance measures are detailed in Sect. 6. The results obtained are analyzed in Sect. 7. Finally, the relevant conclusions and directions for future work are discussed in Sect. 8.

2 Related work

This section describes related work proposed in the literature concerning anticipation in changing environments using EAs.

Stroud (2001) used a Kalman-extended genetic algorithm (KGA) in which a Kalman filter is applied to the fitness values associated with the population individuals. The goal of this Kalman filter is to determine when to generate a new individual, when to re-evaluate an existing individual, and which one to re-evaluate. This KGA is applied to the problem of maintaining a network configuration with minimized message loss in which the nodes are mobile and the transmission over a link is stochastic. As the nodes move, the optimal network changes, but the information contained within the population of solutions allows efficient discovery of better-adapted solutions.

Van Hemert et al. van Hemert et al. (2001) introduced an EA with a meta-learner to estimate, at time t, how the environment will be at time $t + \delta$. This approach uses two populations, one that searches the current optimum and another that uses the best individuals in the past to predict the future best value. The prediction is made based on observations from the past, using two types of predictors: a perfect predictor and a noisy predictor. Individuals from the latter are occasionally sent to the former. The prediction module is not implemented, but instead it is assumed a perfect predictor or a noisy predictor. The idea is tested with two benchmark problems: the knapsack problem and Ŏsmera's function.

An integrated system combining prediction, optimization and adaptation techniques is proposed in Schmidt et al. (2005) and applied to a real world application used to find the best distribution of cars of a particular model across the nation. The problem is complex and the implemented model addresses the issues of transportation, volume sensitivity effect, price depreciation, recent history, current inventory, risk factors and dynamic market changes. The system has three main modules: optimization, prediction and adaptation. The prediction module uses information from the past to give prediction about the sale prices. Schmidt et al. (2005), no information was given about which techniques are used to provide the predictions. The other two modules use the information provided by the prediction module and provide an answer to the actual problem. Finally, the adaptation module receives new information about the problem and adapts the parameters of the prediction module in order to decrease the prediction error. Later, in Michalewicz et al. (2007), this model was used in three different case studies.

Bosman and La Poutré (2006, 2007) proposed several approaches focused on the importance of using learning and anticipation in online dynamic optimization. These works analyzed the influence of time-linkage present in problems such as scheduling and vehicle routing. The presence of time-

linkage in this kind of problem can influence the overall performance of the system: if a decision is made just to optimize the score at a specific moment, it can negatively influence the results obtained in the future. Bosman's works proposed an algorithmic framework integrating evolutionary computation with machine learning and statistical learning techniques to estimate future situations. Predictions are made based on information collected from the past. The used predictor is a learning algorithm that approximates either the optimization function or several parameters.

Hatzakis and Wallace (2001) uses prediction techniques to forecast the location of the Pareto front in multiobjective problems. This approach stores the location of previous solutions and uses autoregressive models to predict the position of the new optimal solution in the next time step. The changes in the environment are known à priori in order to decide when to make the next prediction. A similar technique for multiobjective optimization was investigated by Zhou et al. (2007). This approach explores prediction techniques to re-initialize the population of an EA. The proposed method uses information from the past to guide future search. Each individual in the population is tracked and its history is modeled through a time series model. Predictions about each individual's position at the next time step are made using a linear model. These predictions are used to re-initialize the population after a change is detected. Two strategies for population re-initialization are investigated. The first, predicts the new location of individuals from the location changes that had occurred in the past. Then, the current population is updated using new individuals generated based on that prediction. The second strategy consists of perturbing the current population with Gaussian noise, whose variance is estimated according to the previous changes.

Rossi et al. (2008) compared different techniques to improve the search for tracking a moving optimum using the information provided by a predictor mechanism based on Kalman filters. The used predictor assumes that the changes in the environment are not random and could be learned, helping the EA to keep track of the current optimum.

The prediction methods investigated in the foregoing cited papers are used to estimate the likelihood of particular future situations and to decide what the algorithm must do in the present situation. Since information about the future typically is not available, it is attained through learning from past situations. The mentioned works are mainly focused on the estimation of where the optimum will move. The moment when the change happens is given to the algorithm or detected after its occurrence. Simões and Costa (2008, 2009b) introduced different methods to estimate the movement of the change and also the moment of the next change. With the correct use of the two predictors the EA starts preparing the change before it happens, by introducing useful information into the population. The proposed methods are based on Markov chains, linear and nonlinear regression techniques. The results obtained proves the efficacy of the proposed approaches, which will be further investigated in this article.

3 Dynamic environments

Dynamic environments can be classified in different ways. For example, Branke (2002) categorizes the environments using certain parameters of the problem: the frequency of change, the severity of change, the predictability of change and the periodicity of the change (i.e. the cycle length). A different categorization, suggested by De Jong (2006), uses a direct description of the problems, classifying them in (1) alternating problems, (2) problems with changing morphology, (3) drifting landscapes and (4) abrupt and discontinuous problems. Weicker (2003) proposes a classification of the dynamic environments aiming to establish a mapping between different types of dynamic optimization problems, techniques and performance measures. In order to achieve this, Weicker compares and classifies the dynamic problems using a mathematical framework. Moreover, Yang and Yao (2008b) proposes a classification that uses cyclic, cyclic with noise, and random environments constructed using the XOR Dynamic Optimization Problem generator. Since we are interested in assessing the effectiveness of two different predictors, in this paper we suggest an alternate way of classifying the dynamic environments. Therefore, we classify the changes into two main groups depending on *when* the environment changes and *how* the environment changes. In each group we only discuss the types of environments that are studied in this paper.

3.1 When does the environment change?

The time *when* the changes occur is defined by the change period, which consists of the number of function evaluations between two consecutive changes. Knowing the characteristics of the change period, different decisions can be made concerning the design of the EA. The change period has three main aspects to be considered: (1) the type, (2) the frequency and (3) the predictability.

(1) *Types of change period*: the change period can be classified as (a) *periodic or linear*: the changes are observed at fixed intervals; (b) *patterned*: the interval between the changes is not constant, but instead follows a repeated pattern (c) *nonlinear*: the moments where the changes are observed follow a nonlinear function; and (d) *random*: the changes happen at random points without any pattern or periodicity.
(2) *Frequency of the change*: determines how often the environment changes. Changes can occur every generation

or at larger intervals. Environments that change faster are typically harder to deal with. This is an important issue, since in most approaches using EAs, the algorithm only reacts after the change is detected and the frequency of change can determine if the EA is able to quickly readapt to the new environmental conditions.

(3) *Predictability of the change*: this aspect is directly related with the type of change period. If the change period follows a linear or a repeated trend, we can say that the moment of the next change can be predicted. However, if the change period is completely random, no prediction is possible. We introduce this aspect since we are interested in designing EAs that can react *before* the change happens.

3.2 How does the environment change?

Knowing how the environment changes is important in deciding if the incorporation of a memory component will be useful to the EA or if the application of other methods can be more effective.

(1) *Types of environmental changes*: The environmental changes can be (a) *cyclic*, where situations from the past reappear in the future in a cyclic manner. In this type of environment the number of different environments can determine the difficulty of the problem; (b) *cyclic with noise*, where environments from past reappear but with small differences introduced by a noise factor; (c) *probabilistic*, when the transitions between a fixed number of environments are governed by some probability; and (d) *random*, where the environments change from a state to another completely different state without any correlation with the past.

(2) *Severity of change*: the severity of change measures the strength of the modifications in the environment. The environment can change to a completely different state or to a similar one.

(3) *Predictability of the new environment*: this is directly related with the type of environmental change. If the transitions between the environments are cyclic or follow a trend that can be captured by the algorithm, it is possible to predict which modifications will be observed at the next change. All of the previously mentioned types of environments, except the random ones, present some predicability.

4 Memory-based EAs

In dynamic problems, memory is used to store successful past solutions with the assumption that the optimum may return to its former value. When certain aspects of the problem exhibit some kind of periodic behavior, old solutions might be used to bias the search in their vicinity and reduce the time needed to recover. The use of memory is beneficial on those types of environments.

Memory-based approaches can be divided in two categories: implicit memory and explicit memory. Implicit memory is characterized by the use of redundant representations. By using diploid or multiploid chromosomes, the best individual of the population is implicitly memorized. A dominance scheme controls which genes are expressed in the phenotype (Goldberg and Smith 1987; Ng and Wong 1995; Uyar and Harmanci 2002, 2005; Yang 2006b). The approaches using explicit memory need an extra space to explicitly store information about the individuals or the environments. In these methods, one population performs the search for the optimum and the other population, called memory, stores useful information that is reused later (Yang 2006a; Simões and Costa 2007b, 2012; Barlow and Smith 2008).

This work uses an explicit memory-based EA proposed in Yang (2005) and called memory-enhanced genetic algorithm (MEGA). This algorithm is enhanced by adding two prediction modules (see Sect. 5). In MEGA, the population and the memory are initialized at random. The time to update memory (t_m) is decided using a random integer generator producing an integer between 5 and 10. If the memory is updated at generation t, the next update will occur at generation $t_m = t + rand(5, 10)$. In order to store the most relevant information to an environment in the memory, each time an environmental change is detected, the memory is also updated. If the memory update is due to $t = t_m$, then the current best individual of the population is stored in the memory; if the memory update is because of a change detection, the elite from the previous population is stored. To store a new individual into the memory, another must be replaced: if there are random individuals in the memory, one of them is selected to be replaced; otherwise, the memory point closest to the individual being stored is selected for replacement, if it is a better solution. When an environmental change is detected, a new set of individuals is formed by merging the memory and the search population. Then, these individuals are evaluated in the context of the new environment, and the best p (population size) individuals are selected to become the new search population, which evolves through selection, crossover and mutation. Through this process, the memory remains unchanged. The best individual from the previous population is preserved and transferred to the next population replacing the worst individual (elitism of size 1). More details about this algorithm can be found in Yang (2005, 2008a).

5 Prediction

In this work, the previously described EA is empowered with two prediction modules. For the first predictor, to estimate *when* the next change will take place, two different methods are studied, one using linear regression, another using nonlinear regression. The second predictor is based on Markov chains and is used to estimate *how* the next environment will change. The combined and correct application of these two predictors allows to effectively anticipate the change and prepare the EA to the future environmental conditions. Knowing the time when the next change will happen and how the environment will change, it is possible to retrieve useful information from the memory and introduce it into the population *before* the occurrence of the environmental changes.

5.1 Predicting *when*

Usually, memory-based EAs for dynamic environments detect the changes when they occur. *After* the change is detected the information is retrieved from the memory and inserted into the population (Karaman et al. 2005; Simões and Costa 2007b; Yang 2007). Nevertheless, in certain types of dynamic environments some repeated behavior can be observed and it is possible to make predictions about when the next change will happen. For instance, if the environment changes periodically after a fixed number of generations, the generation when the next change will occur can be correctly predicted. Even in non-periodic environments, if some repeated pattern is present, prediction methods can be successfully applied. Two different methods were previously explored: a linear regression predictor (Simões and Costa 2008) and a nonlinear regression predictor (Simões and Costa 2009b). Those methods, which aim to estimate the time of the next change, are described next. We also describe a novel approach based on the concept of *time-window* that greatly improved the efficiency of the prediction module.

5.1.1 Linear regression predictor

Simple linear regression analyzes the relationship between a response variable y and a single explanatory variable x. This statistical method assumes that for each value of x, the observed values of y are normally distributed around a mean that depends on x. These means are usually denoted by μ_y. In general the means μ_y can change according to any sort of pattern as x changes. In simple linear regression it is assumed that they all lie on a line when plotted against x. Linear regression allows inferences not only for samples for which the data is known, but also for those corresponding to x's not present in the data.

In this work we propose the use of a linear regression predictor to estimate when the next change will happen. If the environment changes periodically at fixed time steps, the generation when the next change will occur can be successfully predicted by linear regression. In this case the explanatory variable x is the number of the change (1, 2, 3,. . .) and the response variable y is the generation where that change occurred (10, 20, 30, . . ., for periodic change periods changing every 10 generations). To predict when the next change will occur using linear regression we proceed as follows: (1) the first two changes of the environment are memorized after they happen (no prediction could be made yet); (2) the $k\ th$ changes, $k > 2$, can be predicted using the equation of the regression line. The first two changes in the environment cannot be predicted since they are needed to calculate the slope and the intercept of the regression line. After the first two changes, and using the values where those changes occurred, an approximation of the regression line is built and predictions about the next possible moment of change are provided. Then, each time a change occurs, new values for the slope and the intercept are computed and the regression line is updated.

In previous work, this predictor used all available data to calculate the slope and the intercept of the regression line and to provide the estimation for the next change. Therefore, as time passed, the predictor became slower. Another limitation of using all available observations was that, if the type of the change period was other than periodic, the prediction accuracy was seriously affected. In this paper we introduce the idea of *time-window*, which consists of the number of observations that is used to estimate the next value. Therefore, instead of using all available data, the linear predictor uses only a fraction of the available information to give the future prediction. Using the appropriate time-window value, the linear regression predictor is faster and also more robust, as the experimental results show.

5.1.2 Nonlinear regression predictor

The linear regression method is not suitable to fit data that follows a nonlinear pattern. For theses cases, nonlinear regression is often used because it allows to model a wide range of situations. The basic idea of nonlinear regression is the same as that of linear regression, namely to relate a response y to a vector of predictor variables x (McCabe and Moore 2003). Nonlinear regression is characterized by the fact that the prediction equation depends nonlinearly on one or more unknown parameters θ. Nevertheless, this method has a major limitation: the nonlinear function must be known. Moreover, an additional task is needed: the estimation of the nonlinear equation parameters. The task of parameter estimation for nonlinear regression is not straightforward. Usually, statistical software using numerical algorithms is used to analyze the data and produce the best parameter's choice for that data (McCabe and Moore 2003). A nonlinear para-

meter estimation problem is an optimization problem whose goal is to minimize the sum of squared errors given by Eq. 1.

$$Sum_{e_i} = \sum_{i=1}^{n} (y_i - f(x_i, \theta_i))^2 \quad (1)$$

Rather than minimizing the sum of squared errors, other techniques minimize the sum of absolute deviations. Several function minimization methods are used in parameter estimation, for instance, weighted least squares, maximum likelihood, Quasi-Newton method, Simplex procedure or Hooke–Jeeves pattern moves (McCabe and Moore 2003; Nash and Walker-Smith 1987). In general, these methods are not easily controllable and require much auxiliary information to work correctly. Another option, more general and easy to apply, is to use a genetic algorithm to evolve a population of individuals that minimize an objective function. This approach was introduced and successfully tested by Pan et al. (1995) and is used in this paper to estimate the parameters vector $\theta_i, i = 1, 2, \ldots, n$ (n is the number of parameters). The GA uses a population of binary strings which corresponds to different values of θ. The required number of genes is determined using the desired precision and the domain size for each parameter. The fitness function used to evaluate the population is the function given by Eq. 1. Because individuals with a higher fitness are selected more often, after some generations the best individual represents the optimal solution for θ. The initial population is generated at random and is evolved using tournament selection, uniform crossover and flip mutation. The best individual of the previous population is transferred to the next population to preserve the best solution found. The desired precision and the parameters' domains are initialized at the beginning, and when the GA is running, in order to provide faster and correct estimations using the known data, an alteration can be made on these initial domains. This task is controlled by two parameters t_d and s_d. The value of t_d is randomly chosen and can have four different values: 1 for a left translation of the domain, 2 for a right translation of the domain, 3 for an increase of the domain and 4 for a decrease of the initial domain. The size of the translation, increase or decrease is given by the parameter s_d ($s_d \in [0.0, 1.0]$). These two parameters are changed during the run, and applied to the best individual of the population. If the individual's fitness is improved using the new domain size, the domains' size is adjusted for all individuals of the next generation.

A predictor based on nonlinear regression is tested to estimate when the next change will happen. While the linear predictor can provide predictions by calculating the slope and the intercept, the nonlinear predictor needs at least one function, in order to give future estimations. Four different functions are incorporated in the nonlinear predictor. The four functions are defined by the Eqs. 2 through 5.

$$f_1 = \theta_1 + \theta_2 x + \theta_3 x^2 \quad (2)$$

$$f_2 = \frac{\theta_1 x}{\theta_2 + \theta_3 x} \quad (3)$$

$$f_3 = \frac{\theta_1}{1 + e^{(\theta_2 - \theta_3 x)}} \quad (4)$$

$$f_4 = \frac{(\theta_1 x)^{\theta_4} + \theta_2 \theta_3}{\theta_3 + x^{\theta_4}} \quad (5)$$

Each one of these functions, using different values for the vector θ can model a wide set of data. For example, function f_1 creates a rapid change period, i.e, with few generations between two changes. Using f_2 the change period is slower; with function f_3 the change period initially changes very quickly but slows down as time proceeds. Function f_4 is used to create a very rapid change period, becoming slower at the end. In the proposed nonlinear regression predictor, a set of n functions $f_1, f_2, \ldots, f_n$, can be used to give predictions. At time t, only one function is active, selected using Eq. 1. As we said, the vector parameter θ is estimated using a standard GA. Every time a change occurs in the environment and additional information is available, the GA is executed to find the vector θ that better fits the data. Thus, the vector θ is estimated using only the *known data*. Using these estimated parameters and the selected function, the predictor indicates *when* the next change will occur. After the k th change has occurred, the prediction error is computed. The prediction error is the difference between the predicted value and the generation when the change actually occurred. If this error is greater than an established threshold α_p, the module analyzes all available functions to assess if another function can provide better results. In this step all functions are re-evaluated using Eq. 1 with the data obtained since the $(k-1)$ th change. The function which minimizes the sum of squared errors is selected to provide the next predictions. If more than one function have equal errors, this choice is made at random. Figure 1 shows how the proposed module works.

Simões and Costa (2009a,b), this predictor was tested using cyclic, patterned and nonlinear change periods. The provided predictions were accurate for all types of change periods. However, the nonlinear change periods for testing this module were created using the functions incorporated in the module. Therefore, the prediction accuracy depended on the quality of the parameters' estimation made by the GA. In this paper we intend to test the accuracy of this module using different nonlinear change periods obtained by other functions, than the four ones incorporated in the predictor. The goal is to see how the predictor chooses the function to approximate the known data and to measure the quality of the predictions provided by it. We also use a change period obtained by a combination of four different nonlinear func-

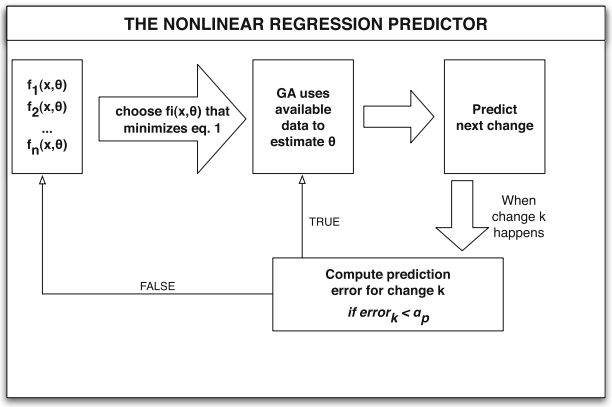

Fig. 1 The nonlinear regression predictor (adapted from Simões and Costa 2009b)

tions, known and unknown to the module. Section 6 details this information.

5.2 Predicting *how*

The prediction of how the environment will be modified in the next change, combined with the prediction about the generation where that change will happen, makes possible the preparation of the population *before* the change. If we know how the next environment will look we can introduce the appropriate individuals into the population. This way, when the change effectively happens, the EA will be prepared to face the new environmental conditions.

To gather information about the characteristics of the known environments we use a Markov chain. The information stored in the Markov chain is also used to estimate which environment(s) can appear in the next change.

5.2.1 Markov chain predictor

A Markov chain can be defined as a sequence of random variables X_1, X_2, X_3, ... that do not keep memory of the whole past (Norris 1997). In fact, Markov chains are memoryless, meaning that the present state is enough to predict future states, i.e.: $Pr(X_{n+1} = x|X_n = x_n, \ldots, X_1 = x_1) = Pr(X_{n+1} = x|X_n = x_n)$.

A discrete Markov chain model can be defined by the tuple (S, P, λ). S is the state space, a finite or countable infinite set of possible values for a sequence of random variables $X_1, X_2, X_3, \ldots$, P is a matrix representing transition probabilities between states. In the matrix P, the element p_{ij} is the probability of going from state X_i to state X_j. λ is the initial probability distribution for all the states in S. $\lambda = p_0, p_1, p_2, \ldots$ with p_i, the probability of starting at state X_i.

Markov chains are often described by a directed graph, where the nodes are the states and the edges are labeled by the probabilities of going from one state to another state.

The proposed approach uses two Markov chains. One is called system Markov chain (*SMC*) and another is called Algorithm Markov chain (*AMC*). The *SMC* is created at the beginning with all the possible states and probabilities transitions. This component is used to decide how the environment changes, but all the information contained in it is *unknown* to the evolutionary algorithm. There is no restriction to the maximum number of states, since this parameter is not related to the memory size: one individual in memory can be connected to more than one state. The *AMC* is created empty at the beginning, and is updated on-line as new information is gathered from the evolutionary process. If a new environment appears, a new state is added to the *AMC* and the corresponding matrix P is updated. The prediction about which environment(s) may appear in the future is given according to the information contained in the *AMC*. In a perfect scenario, at the end of the run, the *AMC* is equal to the *SMC*.

Example

The following example shows how the *AMC* evolves for a maximum number of states equal to 5 using the following *SMC* defined *a priori*: $\lambda = 1.0, 0.0, 0.0, 0.0, 0.0$ and the transition matrix:

$$P_{SMC} = \begin{pmatrix} 0.00 & 0.50 & 0.00 & 0.50 & 0.00 \\ 0.00 & 0.00 & 0.50 & 0.25 & 0.75 \\ 0.00 & 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.25 & 0.00 & 0.75 \\ 1.00 & 0.00 & 0.00 & 0.00 & 0.00 \end{pmatrix}$$

Again, all this information is unknown to the EA and is used to decide how the environment will change.

The *AMC* starts without any information ($t = 0$).

$$t = 0$$

$$P_{AMC} = \begin{pmatrix} 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \end{pmatrix}$$

The first state is chosen using λ, in this example is state 1. Using P_{SMC} the next states can be states 2 or 4. Assuming that the state 2 is chosen at random, the *AMC* transition matrix is updated ($t = 1$).

$$t = 1$$

$$P_{AMC} = \begin{pmatrix} 0.00 & \textit{1.00} & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \end{pmatrix}$$

At this point no predictions are performed, since at least two environmental changes are required to provide predictions. The next state is randomly chosen from the three possibilities (states 3, 4 or 5). Once more, assuming that state 5 is chosen, the *AMC* transition matrix is updated ($t = 2$). From state 5, P_{SMC} indicates that the next state will be state 1, so at $t = 3$ the *AMC* transition matrix is updated.

Since state 1 is already present in P_{AMC}, the Markov model can make a prediction (state 2). Using the P_{SMC}, the next states can be states 2 or 4. If the state 4 is chosen at random, the prediction fails, but this new transition is used to update the *AMC* transition matrix ($t = 4$). Assuming that from state 4, a transition to state 5 is randomly chosen, the P_{AMC} is updated ($t = 5$).

$t = 2$

$$P_{AMC} = \begin{pmatrix} 0.00 & 1.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & \mathit{1.00} \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \end{pmatrix}$$

$t = 3$

$$P_{AMC} = \begin{pmatrix} 0.00 & 1.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 1.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ \mathit{1.00} & 0.00 & 0.00 & 0.00 & 0.00 \end{pmatrix}$$

$t = 4$

$$P_{AMC} = \begin{pmatrix} 0.00 & \mathit{0.50} & 0.00 & \mathit{0.50} & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 1.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 1.00 & 0.00 & 0.00 & 0.00 & 0.00 \end{pmatrix}$$

$t = 5$

$$P_{AMC} = \begin{pmatrix} 0.00 & 0.50 & 0.00 & 0.50 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 1.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & \mathit{1.00} \\ 1.00 & 0.00 & 0.00 & 0.00 & 0.00 \end{pmatrix}$$

State 5 is already present in *AMC*, so the module gives state 1 as prediction. This state is in fact the state that will be chosen at $t = 6$, where P_{AMC} is updated (no changes are made). From state 1, *AMC* gives as predictions states 2 or 4, that are the two real possibilities, and, once more, at $t = 7$ no changes are made to P_{AMC}. At $t = 6$ and $t = 7$ the predictions given by the Markov model are 100 % precise.

End of example

The use of a Markov model assumes that the environmental information can be measured and stored in a specific state of the chain. In this work the Markov model, each state has an integer index and explicitly stores information about the fitness function. This means that each state s ($s = 1, 2, \ldots$) of the *SMC* corresponds to a different instance of the problem to be optimized (capacity of the knapsack, binary template, system variables, etc). When a transition occurs, the *AMC* doesn't know when it occurred (it is detected by the algorithm), but after being detected, the new environment is automatically identified by the *AMC* by observing which instance of the problem is now being optimized, for instance retrieving the new capacity of the knapsack in the new environment.

5.3 Anticipation

The efficacy of the two predictors described before is achieved if they are used at the right moment. In order to prepare the population before the changes happen, an anticipation mechanism must be robust and capable of dealing with erroneous predictions. Moreover, the prediction errors should be used to improve the next predictions' values.

The "right moment" to start preparing the population is decided using the values predicted either by the linear predictor or by the nonlinear predictor. Both methods estimate the generation when the next change will be observed. Knowing this value, the system starts the preparation for the change *some generations before*. If the prediction mechanisms are accurate and the correct information is introduced into the main population *before* the change, the EA's performance is not affected by the changes in the environment. When the prediction mechanisms fail and no anticipation is made, when a change occurs, the EA's performance suffers from a sudden decrease and the EA takes some time to readapt to the new environment.

A parameter called Δ is used to decide how many generations before the predicted moment of change the anticipation starts. The prediction error at time t (e_t) is given by $e_t = g - g'$, where g is the predicted generation for the occurrence of next change and g' is the generation where the change actually happens. Those prediction errors can be positive or negative. A negative error indicates that the predicted value for the next change (g) is smaller than the real value (g'), i.e, $g < g'$ and thus the anticipation of the change is successfully made. A positive error means the opposite. In this situation, if the value of Δ is greater than this error, the anticipation is made before the change; otherwise, the change is detected only when it occurs and no effective anticipation is executed.

The value of Δ must be chosen in order to cover the prediction error and to guarantee that the preparation for the next change is made before it happens. More explicitly, if the next change is estimated to happen at generation g, at generation $g - \Delta$, the Markov model is used to predict the set of possible future environments. At that time, individuals from the memory are retrieved and introduced into the population,

replacing the worst ones. In order to be effective, the Markov model must act before the change. Therefore, if the change is observed at generation g', the value of Δ must assure that the condition $\Delta > |g - g'|$ is observed. In addition, smaller values of Δ are better in the sense that the anticipation starts as close to the change as possible. If the environment changes fast and Δ has a large value, the anticipation can be started in the wrong moment and is ineffective. Thus, the choice of the best value of Δ assumes soaring importance. Different approaches to assign a value to Δ were tested in Simões and Costa (2009a): Δ constant and Δ adjustable. The auto-adjusting methods for Δ used the previous errors to decide the new value. Different approaches to adjust the value of Δ were tested:

- using the maximum prediction error
- using the average of the positive prediction errors
- using the average of all the prediction errors (absolute value)
- using the maximum and the average of the positive prediction errors

The results obtained showed that all the methods for adjusting Δ were very effective. In this paper we use the method that obtained the best results, which is the method that changes the value of Δ using the maximum and the average of the positive prediction errors (Simões and Costa 2009a). In this method, the value of Δ is updated as follows: in the first two changes $\Delta = 5$, in the next k changes ($k > 2$) the highest prediction error and the average of the positive errors are used to update Δ:

$$\Delta(k) = \begin{cases} 5 & \text{if } k = 1, 2 \\ max\left\{2, \frac{\Delta_m(k)+\Delta_{av}(k)}{2}\right\} & \text{if } k > 2 \end{cases} \quad (6)$$

$\Delta_m(k)$ computes a value for Δ using the highest prediction errors:

$$\Delta_m(k) = \begin{cases} 5 & \text{if } k = 1, 2 \\ max\{2, e_1, e_2, \ldots, e_k\} & \text{if } k > 2 \end{cases} \quad (7)$$

and $\Delta_{av}(k)$ is calculated using the average of the positive prediction errors:

$$\Delta_{av}(k) = \begin{cases} 5 & \text{if } k = 1, 2 \\ max\{2, \frac{\sum_{i=1}^{k} e_i}{k}\} & \text{if } k > 2 \text{ and } e_i > 0 \end{cases} \quad (8)$$

where e_i is the observed error at the i th change.

5.4 Putting it all together: prediction in the EA

The proposed computational model, consisting of MEGA enhanced with two predictors and one anticipation modules, is called *PredEA* (see Fig. 2). This algorithm is a traditional *EA* that evolves a population of individuals aiming

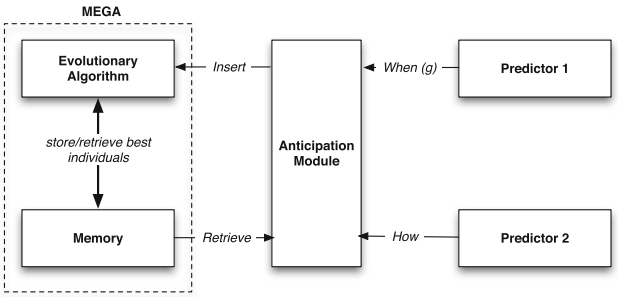

Fig. 2 Architecture of PredEA (adapted from Simões and Costa 2009a)

to optimize the current fitness function. A memory is used to store useful information from the past that is used in future changes. The first prediction module uses information about *when* the previous changes occurred to estimate the generation when the next change will be observed. The predictions are given by one of the techniques previously discussed, based on linear or on nonlinear regression. The second module uses a Markov chain to keep track of previous environments and provides predictions on *how* the environment will look like in the next change step. The two predictor modules are managed by a third component, the anticipation module, that uses the information provided by the previous two modules and prepares the EA for the next change.

Memory-based evolutionary algorithm

In this algorithm, a main population of individuals evolves by means of selection, crossover and mutation and is used to find the best solution for the current environment. Another population is used as memory to store the best current individual from time to time. It starts empty and has a limited size (20 % of the global number of individuals). When a change happens (or it is predicted) the information stored in the memory is retrieved and used to help the EA readapt to the new environment. When the memory is full, the replacement of the memorized individuals is made using the *generational* replacing strategy proposed by Simões and Costa (2007a). This method replaces a memory individual of the same period by the best individual of the population, if it is a better solution. If there are more different environments than the capacity of the memory, the replacement is made using the *similar* strategy, which replaces the most genetically similar individual found in memory. Instead of updating memory in a fixed time interval, we create a stochastic time pattern using a random value between 5 and 10 ($t_m = t + rand(5, 10)$). The memory is also used to *detect changes* in the environment: a change occurs when at least one individual in the memory has its fitness changed. When the memory is updated, the best individual of the population is stored and the current state of the Markov model is added as a reference to that individual. If the memory capacity is attained and a solution is replaced,

the new solution keeps the links to the Markov model states that the replaced individual had and a new link to a different state can be added. So, a memory solution can be linked to more than one state.

Predictor 1 module (P1)
This predictor uses information about when previous changes were observed to estimate when the next change would occur. Two methods are tested for this module: (1) a linear regression predictor which is tested using all available observations to estimate the next change or using only a fraction of those observations (time-window); (2) a nonlinear regression predictor is also investigated. This module is activated every time a change is detected, i.e, at change k the module provides an estimation for the generation where the change $k+1$ may occur.

Predictor 2 module (P2)
This module keeps track of the different environments and estimates which environments may appear in the next change. Those predictions are made using only the information known so far about the previous environmental changes. Each state of the Markov chain corresponds to a different environment. If two states are linked, it means that a change happened from one state to the other. Associated to each transition is a probability value which is updated every time a change is detected. The initial state is randomly chosen among the existing states. Again we stress that this information is unknown to the algorithm and the model is updated throughout time. The information that is stored about each environment is problem dependent.

Anticipation module (A) This module receives the information provided by the two predictors and decides when to start the preparation of the EA for the next change. This activation must be done at the correct time in order to prepare the population to the next environment(s) predicted by the $P2$ module. The $P1$ module estimates the generation when the next change will be observed and the A module starts the anticipation some generations before. The Δ parameter described previously is used by this module to decide how many generations before the predicted moment of change (g) the anticipation starts. The value of this parameter is also used to cover the prediction errors associated with the $P1$'s estimations. The anticipation process consists of retrieving from memory individuals that were good solutions in the environments that the $P2$ module indicates as the next to appear. These individuals are inserted into the main population at generation $g-\Delta$, replacing the same number of worst individuals, so the population size is kept constant. If the $P2$ module don't provide any prediction, five random individuals from memory are inserted into the population, replacing five randomly selected individuals. These retrieved individuals remain unchanged (i.e., they are not affected by crossover or mutation) until the change happens. Algorithm 1 shows the pseudo code of the *PredEA* algorithm.

```
Function PredEA
    t_m: time to update memory
    max: maximum number of states
    markov: System Markov chain
    initial_state: initial state

1   t = 0
2   k = 0
3   t_m = rand(5, 10)
4   P(t) = initialize population randomly
5   M(t) = initialize memory randomly
6   Δ(k) = 5
7   Initialize the AMC information
8   repeat
9   evaluate memory M(t)
10  evaluate population P(t)
11  preserve best individual from P(t − 1)
12  if a change k is detected
13      Store performance measures
14      Activate the P1 module:
15          (i) Update P1 information
16          (ii) Predict next change g
17      Update Δ(k)
18      k = k + 1
19      Update AMC
20      if prediction fails
21          P'(t) = Select best from P(t) + M(t)
22  else P'(t) = P(t)
23  if t = g − Δ(k)
24      Activate the P2 module:
25          (i) next_s = Predict next state(s)
26          (ii) if number of next_s > 0
27              Activate the A module:
28              P'(t) = Insert solutions from M(t)
29           else
30              P'(t) = Insert 5 random inds.
31  else P'(t) = P(t)
32  if t = t_m or change detected
33      Update memory
34      t_m = t + rand(5, 10)
35  P''(t) = Selection(P'(t))
36  Crossover(P''(t))
37  Mutation(P''(t))
38  P(t + 1) = P''(t)
39  t = t + 1
40  until stop_condition is 7r7e
```

Algorithm 1 Pseudocode of PredEA

6 Experiments

This section introduces the experiments that were performed to study, compare and validate the proposed prediction mechanisms in different dynamic problems. It provides a description of the problems used as benchmarks, details the different types of dynamic environments, specifies the parameter settings used in the experiments and finally describes how the evaluation and validation of the results was made.

6.1 Problems

In this paper two benchmark problems are used to test the proposed methods: the dynamic knapsack and the dynamic bit-matching problems. Benchmark problems should have a set of good characteristics like being simple to implement, analyze or evaluate, computational efficient, flexible and allow conjectures about real-world problems (see Nguyen et al. 2012). The dynamic knapsack and the dynamic bit-matching belong to this set of benchmarks that have these good characteristics (see Cruz et al. 2011). For the purpose of this investigation, the use of these benchmarks is preferable to the use of artificial general-purpose problems generators (see Rohlfshagen and Yao 2009, Ben-Romdhane et al. 2013). The knapsack problem is an NP-complete combinatorial optimization problem often used as benchmark. It consists of selecting a number of items to a knapsack with limited capacity. Each item has a value and a weight and the objective is to choose the items that maximize the total value, without exceeding the capacity of the bag. In this paper, the knapsack uses 100 items, the initial weights are randomly created in the range [1, 50] and the values are obtained by adding the corresponding weight to a random number obtained in the range [1, 5]. The capacity of the knapsack is set to 60 % of the total weight of all items. The fitness of an individual using binary representation is equal to the sum of the values of the selected items, if the weight limit is not reached. If too many items are selected, then the fitness is set to the difference between the total weight of all items and the weight of the selected items, multiplied by 10^{-10}, in order to ensure that invalid individuals are distinguished from the valid ones. The problem becomes dynamic by changing the capacity of the knapsack.

The bit-matching problem is a unimodal problem whose goal is to find a solution that matches a given template. Changing the template from time to time turns this problem dynamic. The severity of the change is defined by the number of bits that change in the template. The difficulty of the problem can be increased using templates with larger dimensions.

The maximum number of different environments (max) is set to $max = 3, 5, 10, 20$ or 50. Depending on the problem, max different capacities (for the dynamic knapsack problem) or max different templates (for the dynamic bit-matching problem) are created. The different capacities are generated from the initial capacity of the knapsack and making variations of 20 % (severity of the change), following the next equation (m is the number of items, w_i is the weight of the $i\ th$ item, and C is the capacity of the knapsack):

$$C(t) = \begin{cases} 0.6 \times \sum_{i=1}^{m} w_i, \texttt{if} & t = 0 \\ C(t-1) - 0.2 \times C(t-1), \texttt{if} & t \texttt{ is odd} \\ C(t-1) + 0.2 \times C(t-1), \texttt{if} & t \texttt{ is even} \end{cases}$$

The different templates are created using an initial template $T(0) = \{0\}$. The following templates are obtained by changing $\frac{l}{max}\%$ (severity of the change) of the bits from the previous template (l is the chromosome length). Both problems use binary representation with chromosomes of length 100.

6.2 Dynamic environments

According to the classification provided in Sect. 3, three different types of change period are used: (1) *periodic* (or linear): every r generations, with $r = 10$, $r = 50$, $r = 100$ and $r = 200$; (2) *patterned*: the moments of change are decided by repeating an established pattern. Four different patterns are included in the study: 5-10-5, 10-20-10, 50-60-70 and 100-150-100. The generations when the environment changes are calculated using $change(i) = change(i-1) + pattern(i-1 \bmod K)$, where mod is a modula operation, K is the total number of components in a pattern (in this paper $K = 3$) and i is the change index ($i = 1, 2, \ldots$). For the first change ($i = 1$) we assume, $change(i-1) = 0$; (3) *nonlinear*: the change period is defined by a nonlinear function. In order to generate different nonlinear change periods, seven different types of nonlinear functions are used. These functions are the four ones incorporated in the module (Eqs. 2 through 5), and three additional ones, never used before, defined by Eqs. 9 through 11. The goal is to see how the nonlinear predictor approximates the data generated with functions that are not included in the module and how the linear predictor performs for these cases. The three new functions are defined by the following equations:

$$f_5 = \frac{\theta_1 x}{1 + e^{\theta_2 \theta_3 x}} \tag{9}$$

$$f_6 = \frac{\theta_1 x^2 + \theta_2 x}{e^{\theta_3 x} + \theta_4} \tag{10}$$

$$f_7 = \begin{cases} f_4, \texttt{if} & 1 < x \leq 100 \\ f_6, \texttt{if} & 100 < x \leq 200 \\ f_5, \texttt{if} & 200 < x \leq 300 \\ f_1, \texttt{if} & 300 < x \leq 500 \end{cases} \tag{11}$$

For generating the seven nonlinear change periods, the parameter vector θ is used with the values of Table 1.

For each of the change period types, two different types of environmental changes are defined (cyclic and probabilistic) and the environments change between max different states. For the probabilistic type, the probabilities associated to each different state are set at the beginning of the run and correspond to the system Markov model (*SMC*) transition matrix. The probabilistic transitions are applied only in certain states, chosen at random. So, for 50 states, only a fraction of them have probabilistic transitions. The number of transitions is also chosen at random, between 2 and 5. The severity of

Table 1 Values for θ_i used in the experiments

Function		θ_1	θ_2	θ_3	θ_4	Change period
f_1		60.00	2.00	0.10	–	$NL1$
f_2		62.67	0.627	0.047	–	$NL2$
f_3		15,000.00	5.00	0.05	–	$NL3$
f_4		10,000.00	100.00	100.00	1.50	$NL4$
f_5		75.00	0.050	0.04	–	$NL5$
f_6		50.00	15,000.00	0.045	1,000.00	$NL6$
f_7	f_4	7,000.00	1.00	200.00	1.30	$NL7$
	f_6	950.00	500.00	0.01	2,000.00	
	f_5	245.00	0.055	0.040	–	
	f_1	20.00	50.00	0.112	–	

the change is kept constant according to the description in the previous section. If the Markov model provides accurate predictions the severity of the change is not relevant for the performance of the algorithm, since that the stored information allows the reintroduction of the best individuals for the next change. The severity of the change can affect the performance of the algorithm if the prediction fails, but this experimentation is not included in this study.

6.3 Settings

Four different versions of PredEA are compared with the standard MEGA, that will be called noPredEA. PredEA is tested with the linear regression predictor using all known observations (PredEA-LR), using a time-window of size two (PredEA-LR2) and using a time-window of size ten (PredEA-LR10). Moreover, PredEA is also tested using the nonlinear predictor (PredEA-NLR). A time-window of size 2 or 10 means that the next prediction is made using the 2 or the 10 previous observations, respectively. For the nonlinear regression predictor, the estimation of the parameters is done using all available observations, except when a new function is selected. In this case, at k th change, the predictor considers only the observations measured after change $(k-1)$ th. All those techniques are used combined with the Markov chain predictor. In all the experiments, the information concerning the type of change period, the type of environmental change, the change period size, and the number of different states are *unknown* to the *EA*. A total of 30 runs are performed with each technique for each problem. Binary representation is used for all the studied problems. The global number of individuals (n) is set to 100. The memory size m is set to 20 % of n. The EA is allowed to evolve for as many generations as necessary so as to result in 500 environmental changes. The detection of a change is made using the memory—a change occurs when at least one individual in the memory has its fitness changed. This method is not enough to detect changes for the Knapsack problem, so an additional mechanism is incorporated: the expected knapsack capacities are compared with the capacities generated by the algorithm. When a difference is observed, a change in the environment is assumed. The uniform crossover operator is applied with a probability of 70 % and flip mutation is used with 1 % rate. The initial population and memory are randomly created and the selection of parents is made using the binary tournament selection method. The next population is formed using the generated offspring, through recombination and mutation, and the best individual (the elite) of previous population is preserved. For the $P2$ module, the threshold α_p is set to 10.

The parameters of the GA used to find the best parameters to the nonlinear regression predictor are the following: population of size 50, uniform crossover rate of 75 % and mutation rate of 1 %. The GA is stopped in one of three cases: if no better solution is found for 10 generations, if the sum of the squared errors is equal to zero, if a maximum of 1,000 generations is attained. The domain of the parameters, and consequently the chromosome size, depends on the nonlinear function as shown in Table 2. The precision used for each parameter θ is six places after the decimal point. The initial parameters' domains are chosen in order to enclose a wide set of nonlinear curves but can change as described in Sect. 5.

6.4 Evaluation and validation

To evaluate the different algorithms, the offline performance measure (Branke 2002) is used, which is defined as follows:

$$offline(t) = \frac{1}{t}\sum_{i=1}^{t} best'_t$$

where t is the actual generation and $best'_t$ is the maximum observed fitness since the last time step at which a change in the environment occurred. This measure includes the re-evaluation of the memory individual's introduced after the change is detected. The values presented in the next section refer to the average of the offline performance obtained at the end of the 30 runs. This work compares 6 different algorithms, applied to 15 different types of change periods, using 2 different dynamics (cyclic and probabilistic) for 5 different

Table 2 Upper and lower limits for the parameters' domain and number of genes used to encode each parameter

f	θ_i	Lower limit	Upper limit	Num of genes for θ_i	Chromosome length
f_1	θ_1	−10	65	27	80
	θ_2	0	200	28	
	θ_3	0	5	25	
f_2	θ_1	0	65	26	65
	θ_2	0	1	20	
	θ_3	0	0.5	19	
f_3	θ_1	14,000	15,000	34	76
	θ_2	0	5	23	
	θ_3	0	0.5	19	
f_4	θ_1	9,000	10,000	30	103
	θ_2	50	100	26	
	θ_3	50	100	26	
	θ_4	0	2	21	

settings of the maximum number of different states, in two benchmark problems, resulting in a total of 1800 different tested situations. All the results obtained are statistically validated using the nonparametric Friedman test at a 0.01 level of significance. After this test, if a significance is found, the multiple pair wised comparisons are performed using the Nemenyi procedure. For multiple comparisons, the p-value (0.01) used in the Nemenyi test is adjusted using the Bonferroni correction method. In the statistical tables presented in the next section, each line compares a pair of algorithms using the notation "++" or "−−", when the first algorithm is significantly better than, or significantly worse than the second one, respectively. The use of "∼" indicates that there is no statistical significance in the results obtained by the two algorithms.

7 Results

This section presents the results of all the experiments. First, the prediction accuracy of the proposed methods is analyzed and second, the performance of the different algorithms is compared.

7.1 Prediction accuracy

Prediction accuracy consists of the frequency of correct outcomes reached by the proposed predictors. First we analyze the accuracy for the Markov model predictor, and then for the linear and nonlinear regression predictors.

7.1.1 Accuracy of the Markov model predictor

For the Markov model predictor, a predicted value is considered correct when the module provides the correct value for the next environmental transition.

Table 3 Accuracy of the Markov model predictions

Number of different environments	Type of dynamics	Accuracy (%)
3	Cyclic	99.40
3	Probabilistic	98.80
5	Cyclic	99.00
5	Probabilistic	98.00
10	Cyclic	98.00
10	Probabilistic	96.20
20	Cyclic	96.00
20	Probabilistic	93.60
50	Cyclic	90.00
50	Probabilistic	87.00

Table 3 shows the prediction accuracy of the Markov model based on 500 environmental changes, in each run. As the number of different environments increases, the prediction accuracy expectedly decreases, but the attained results are still very good. Moreover, for probabilistic dynamics the prediction accuracy is worse. This happens because the Markov model uses data collected from previous transitions to estimate the future. As the number of environments increases or the transitions are probabilistic, the Markov model needs more time to learn the entire behavior of the environment. So, the proposed predictor based on a Markov model provides excellent predictions, well above 90 %, in almost all situations.

Figure 3 shows how the EA behaves over time using prediction and without prediction. The example illustrates a typical result for the first 50 environmental changes in the dynamic bit matching problem with ten different states. We can see that the EA using prediction goes through a *learning* phase—where the Markov model acquires the history of possible environmental transitions—and an *equilibrium*

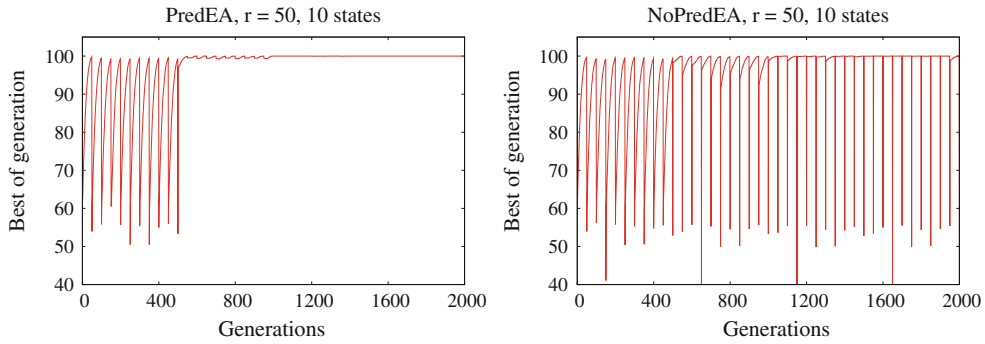

Fig. 3 Best of generation for PredEA-LR and noPredEA, bit matching problem

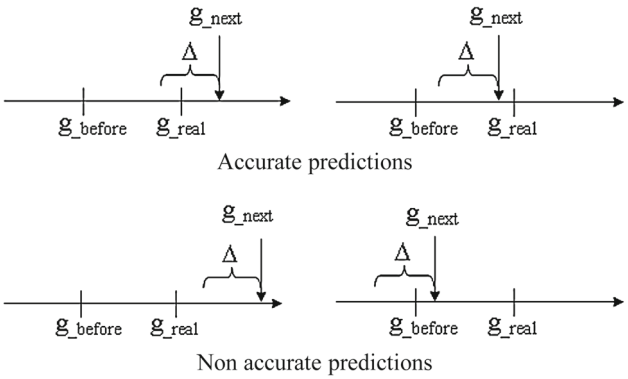

Fig. 4 Examples of good and bad predictions for the Linear and non-linear predictors

phase—where the Markov model provides the correct predictions. During the equilibrium phase no fitness decrease is observed. On the other hand, the EA without prediction experiences a decrease on its performance every time a change happens. In this case, the recuperation is achieved only after the change, when the information from memory is introduced into the population.

7.1.2 Accuracy of the linear and nonlinear predictors

For the linear and nonlinear regression predictors a value is considered accurate if, using the value of Δ, the anticipation is made before the real change occurs. If a previous change occurs at generation g_{before} and the predicted value for next change is g_{next}, this value is considered accurate if $g_{next} - \Delta < g_{real}$ and $g_{next} - \Delta \geq g_{before}$ (where g_{real} corresponds to the generation when the change effectively happens and the accuracy of the predicted value is measured). Figure 4 shows different cases of good and bad predictions.

The results of Tables 4 and 5 show that the three linear prediction techniques, in periodic change periods, obtain 100 % accuracy. This is expected, since data follows a straight line, correctly estimated by these methods. For PredEA-LR, when the change period is patterned, the prediction accuracy is always above 99 %, and the prediction errors are negative,

Table 4 Accuracy of PredEA-LR

PredEA-LR				
Type of change period	Change period	Accuracy %	Error	Δ
Periodic	$r = 10$	100.00	0.00	2.01
	$r = 50$	100.00	0.00	2.02
	$r = 100$	100.00	0.00	2.02
	$r = 200$	100.00	0.00	2.03
Patterned	5-10-5	99.87	−0.32	3.01
	10-20-10	99.73	−0.29	6.06
	50-60-70	99.66	−0.29	8.06
	100-150-100	99.66	−0.02	31.95
Nonlinear	$NL1$	0.00	−1,885.52	5.00
	$NL2$	0.25	778.56	592.53
	$NL3$	7.07	−485.08	208.22
	$NL4$	5.56	1,240.24	1,066.77
	$NL5$	2.90	441.40	157.05
	$NL6$	7.63	−651.28	221.87
	$NL7$	6.24 %	−1,125.53	780.00

indicating that the predicted values correspond to a generation before the real change. Nevertheless, the values of Δ are able to cover those errors, leading to the high prediction accuracies presented in Table 4. Even when Δ presents larger values, for the 50-60-70 and the 100-150-100 patterned change periods, the predictions are correct. These larger values are due to the predictions errors used for adjusting the value of Δ. For the situations where the change period follows a nonlinear trend, the PredEA-LR technique fails. The prediction accuracies for those cases are very weak. This happens because the straight line obtained by the linear regression method uses all the observations which do not fit the data obtained by the nonlinear functions. The enormous prediction errors confirm that evidence.

The linear regression is also tested using two different sizes for the time-window (PredEA-LR2 and PredEA-LR10). The results for these two methods are in Table 5 and show

Table 5 Accuracy of PredEA-LR2 and PredEA-LR10

Type of change period	Change period	PredEA-LR2			PredEA-LR10		
		Accuracy (%)	Error	Δ	Accuracy (%)	Error	Δ
Periodic	$r = 10$	100.00	0.00	2.00	100.00	0.00	2.00
	$r = 50$	100.00	0.00	2.01	100.00	0.00	2.01
	$r = 100$	100.00	0.00	2.01	100.00	0.00	2.01
	$r = 200$	100.00	0.00	2.02	100.00	0.00	2.02
Patterned	5-10-5	99.87	0.01	4.9	99.87	0.01	2.01
	10-20-10	99.73	0.03	9.94	99.73	0.03	4.03
	50-60-70	99.66	−0.03	19.77	99.66	0.01	5.36
	100-150-100	99.66	0.17	49.53	99.66	0.23	20.29
Nonlinear	$NL1$	100.00	−0.20	2.01	99.83	−2.57	2.01
	$NL2$	100.00	0.23	2.01	93.97	1.98	4.50
	$NL3$	90.91	−0.01	2.43	68.18	−0.51	15.75
	$NL4$	66.16	0.86	6.26	20.71	11.45	39.68
	$NL5$	100.00	0.07	2.01	100.00	0.22	2.01
	$NL6$	100.00	−0.10	2.01	100.00	−1.51	2.01
	$NL7$	97.38	−0.22	3.79	93.36	−2.71	13.24

Table 6 Accuracy of PredEA-NLR

PredEA-NLR					
Type of change period	Change period	Accuracy (%)	Error	Δ	Chosen function
Periodic	$r = 10$	100.00	0.00	2.01	f_1
	$r = 50$	100.00	0.00	2.02	f_2
	$r = 100$	100.00	0.00	2.02	f_1
	$r = 200$	100.00	0.00	2.03	f_2
Patterned	5-10-5	100.00	−0.36	2.02	f_1
	10-20-10	66.76	−0.30	3.03	f_1
	50-60-70	99.66	−0.28	5.12	f_1
	100-150-100	100.00	−0.27	27.77	f_1
Nonlinear	$NL1$	97.58	−0.85	5.00	f_1
	$NL2$	100.00	−0.39	2.07	f_2
	$NL3$	100.00	−1.56	2.02	f_3
	$NL4$	100.00	−0.72	2.02	f_4
	$NL5$	13.46	−8.66	25.98	f_1
	$NL6$	58.35	5.95	5.36	f_1
	$NL7$	62.98	−3.36	58.85	f_1 and f_4

that the chosen size for the time-window influences the obtained prediction, especially in nonlinear change periods. The PredEA-LR2 technique obtains high prediction accuracy in all types of change periods. The lowest value (66.16 %) is obtained in the $NL4$ change period. Although the value of Δ is enough to cover the prediction errors, due to the characteristics of $NL4$ curve, several erroneous anticipations are performed. The PredEA-LR10 obtains similar results for the periodic and patterned change periods but the prediction accuracy is slightly worst for the nonlinear change periods.

The results shown on Table 6 refer to the prediction accuracy of PredEA-NLR. The last column of this table indicates which function, incorporated in the module, is chosen to provide the predictions. The nonlinear predictor is 100 % accurate in the periodic change periods and obtains near 100 % of accuracy in the patterned change periods. An exception occurs for the change period 10-20-10 (66.76 %). For periodic and patterned change periods the functions f_1 or f_2 are selected to estimate the time of the next change. The values of the prediction error show that this selection is appropri-

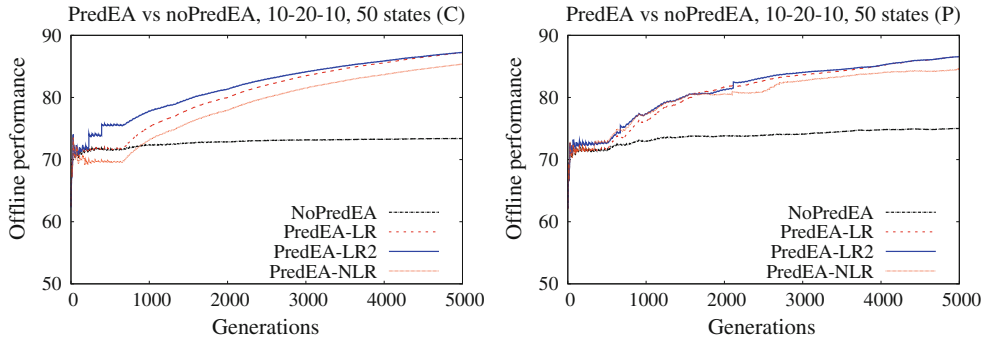

Fig. 5 Offline performance over time, 10-20-10, 50 states, cyclic (C) and probabilistic (P) transitions, bit matching problem

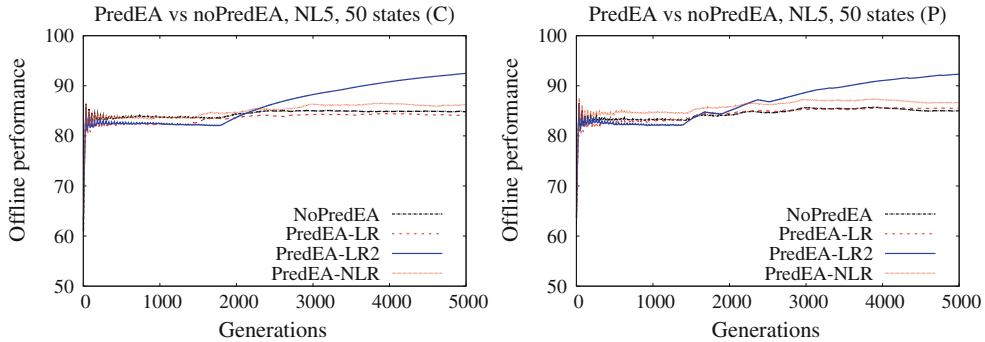

Fig. 6 Offline performance over time, $NL5$, 50 states, cyclic (C) and probabilistic (P) transitions, bit matching problem

ate. For the nonlinear situations, PredEA-NLR has a higher prediction accuracy for $NL1$ through $NL4$ change periods, since those functions are incorporated in the module and are correctly selected to provide the estimated values. For $NL5$ through $NL7$ the prediction is made using the functions f_1 and f_4 and the accuracy is lower. Despite the worst accuracy obtained by PredEA-NLR, the prediction errors show that the module is able to select the best function available. For the change period $NL7$, the function f_4 is selected firstly and, at generation 102, the function f_1 is activated and used through the rest of the run.

The results obtained reveal some limitations of PredEA-NLR, namely the need to have a 'good' function to provide valid estimations. Moreover, for PredEA-LR, the use of all available information, besides the computational cost, leads to poor prediction accuracies in nonlinear change periods. The use of a time-window in the linear regression predictor (PredEA-LR2, PredEA-LR10) allows to obtain the best prediction accuracies in all the studied situations. The size of the time-window is an important choice that should be further analyzed.

7.2 Algorithms' performance

This section sets forth the performance obtained by the different methods. The results refer to PredEA-LR, PredEA-LR2 and PredEA-NLR and are compared with noPredEA. The results of PredEA-LR10, for lack of space, are not reported, but are similar to PredEA-LR2's results.

Figures 5 and 6 show the evolution of the algorithms' performance over time. These two figures refer to the bit matching problem, for the 10-20-10 and the $NL5$ change periods, for 50 different states and are representative of the remaining cases. Table 7 shows the results for the dynamic bit matching problem, and Table 8 contains the scores concerning the dynamic knapsack problem. The statistical results, obtained using the statistical tests, are on Tables 9 and 10 for the dynamic bit matching problem and the dynamic knapsack problem, respectively. In all tables, adjacent to the number of states is the type of environment: C for cyclic and P for probabilistic.

From the analysis of the results it is evident that, in general, all the prediction techniques allow the EA to preform significantly better than the same algorithm without prediction. The only situation where this observation is not so consistent is for the PredEA-LR in the nonlinear change periods, because of the poor accuracy of PredEA-LR on these cases (from 0.00 to 7.63 %—see Table 4). In fact, the results show that a relation between the prediction accuracy and the performance obtained by the algorithms can be found. Since the prediction accuracy for the Markov-based model is the same for all the algorithms, the different performances are related to the performance of the linear/nonlinear predictors. For the periodic change periods, all the methods are very accurate (100 %) and

Table 7 *PredEA* and *noPredEA* results: dynamic bit matching for periodic, patterned and nonlinear change periods and for cyclic (C) and probabilistic (P) transitions between 3, 5, 10, 20 and 50 states

Bit matching		Number of states									
Period	Algorithm	3		5		10		20		50	
		C	P	C	P	C	P	C	P	C	P
$r = 10$	noPredEA	89.68	90.63	85.99	86.53	80.74	81.97	75.27	76.53	69.74	70.89
	PredEA-LR	98.14	97.32	96.86	96.24	93.72	93.39	90.17	90.14	84.86	85.62
	PredEA-LR2	98.75	97.93	97.53	96.83	94.91	94.09	91.23	90.68	85.32	85.83
	PredEA-NLR	98.24	97.37	96.89	96.27	93.76	93.49	90.18	90.18	84.89	85.64
$r = 50$	noPredEA	99.01	99.01	98.92	98.92	98.68	98.72	97.26	97.65	89.11	90.74
	PredEA-LR	99.86	99.85	99.79	99.80	99.64	99.58	99.27	99.22	98.20	98.03
	PredEA-LR2	99.90	99.89	99.83	99.80	99.66	99.64	99.31	99.25	98.26	98.07
	PredEA-NLR	99.91	99.89	99.83	99.81	99.66	99.63	99.32	99.25	98.26	98.05
$r = 100$	noPredEA	99.50	99.50	99.44	99.45	99.31	99.33	99.12	98.82	94.44	95.42
	PredEA-LR	99.93	99.92	99.88	99.87	99.77	99.75	99.54	99.51	98.84	98.71
	PredEA-LR2	99.94	99.93	99.89	99.88	99.78	99.76	99.55	99.52	98.87	98.75
	PredEA-NLR	99.94	99.93	99.89	99.88	99.78	99.76	99.55	99.52	98.86	98.72
$r = 200$	noPredEA	99.73	99.73	99.70	99.70	99.60	99.61	99.46	99.31	97.17	97.49
	PredEA-LR	99.94	99.94	99.91	99.90	99.82	99.81	99.65	99.63	99.13	99.06
	PredEA-LR2	99.95	99.95	99.92	99.91	99.83	99.82	99.66	99.64	99.14	99.07
	PredEA-NLR	99.95	99.95	99.92	99.91	99.83	99.82	99.66	99.64	99.14	99.07
5–10–5	noPredEA	85.56	89.94	85.59	86.53	80.97	81.48	74.05	75.73	67.99	69.04
	PredEA-LR	96.57	96.45	94.74	94.02	90.52	90.34	86.88	87.10	84.23	81.53
	PredEA-LR2	96.86	95.76	94.67	93.24	90.49	90.15	86.83	86.44	84.28	81.60
	PredEA-NLR	96.89	96.76	94.90	94.04	90.58	90.45	86.89	87.26	84.59	81.85
10–20–10	noPredEA	91.43	94.68	92.42	92.47	87.69	88.42	81.00	82.87	73.40	75.00
	PredEA-LR	97.79	98.31	97.67	96.40	95.14	94.22	92.80	92.32	87.22	86.52
	PredEA-LR2	98.25	98.56	97.60	96.69	95.58	95.18	93.13	92.31	87.26	86.58
	PredEA-NLR	95.51	96.42	94.56	93.28	93.92	92.27	91.16	90.83	85.36	84.62
50-60-70	noPredEA	99.15	99.15	99.06	99.07	98.83	98.86	98.15	97.88	90.78	92.34
	PredEA-LR	99.89	99.86	99.80	99.79	99.62	99.60	99.23	99.18	98.07	97.86
	PredEA-LR2	99.87	99.87	99.82	99.79	99.62	99.60	99.23	99.17	98.04	97.85
	PredEA-NLR	99.89	99.87	99.82	99.80	99.63	99.61	99.25	99.19	98.09	97.89
100–150–100	noPredEA	99.57	99.57	99.52	99.52	99.41	99.42	99.25	98.98	95.22	96.17
	PredEA-LR	99.92	99.92	99.91	99.90	99.81	99.80	99.62	99.58	99.02	98.91
	PredEA-LR2	99.93	99.94	99.93	99.91	99.84	99.83	99.65	99.57	99.03	98.92
	PredEA-NLR	99.94	99.94	99.92	99.91	99.83	99.82	99.64	99.59	99.05	98.92
$NL1$	noPredEA	98.95	98.94	98.73	98.77	98.38	98.49	97.02	97.41	91.60	92.83
	PredEA-LR	98.94	98.67	99.00	98.67	98.40	98.58	97.12	97.75	91.76	93.46
	PredEA-LR2	99.78	99.75	99.68	99.66	99.50	99.51	99.26	99.24	99.59	98.52
	PredEA-NLR	99.74	99.68	99.64	99.61	99.48	99.48	99.25	99.22	99.58	98.53
$NL2$	noPredEA	98.11	98.14	97.96	97.97	97.58	97.35	97.01	94.40	84.08	85.36
	PredEA-LR	98.12	98.15	97.96	97.97	97.57	97.35	97.01	94.38	84.10	85.43
	PredEA-LR2	99.79	99.78	99.62	99.60	99.30	99.23	98.63	98.49	96.54	96.16
	PredEA-NLR	99.80	99.79	99.67	99.65	99.33	99.26	98.66	98.53	96.55	96.19
$NL3$	noPredEA	99.06	99.07	98.84	98.88	98.48	98.55	97.12	97.47	93.42	94.05
	PredEA-LR	99.34	99.03	99.29	99.17	99.11	99.01	98.51	98.42	94.01	94.65
	PredEA-LR2	99.57	99.55	99.44	99.42	99.52	99.46	98.95	98.67	97.52	97.37
	PredEA-NLR	99.98	99.92	99.85	99.83	99.60	99.57	99.13	99.07	97.71	97.58

Table 7 continued

Bit matching		Number of states									
Period	Algorithm	3		5		10		20		50	
		C	P	C	P	C	P	C	P	C	P
*NL*4	noPredEA	99.02	99.06	98.87	98.88	98.52	98.52	97.90	97.44	92.92	93.48
	PredEA-LR	99.07	99.09	98.90	98.91	98.53	98.53	97.91	97.41	92.94	93.57
	PredEA-LR2	99.73	99.73	99.60	99.58	99.27	99.23	98.65	98.54	96.54	96.56
	PredEA-NLR	99.96	99.94	99.83	99.80	99.49	99.45	98.83	98.73	96.73	96.72
*NL*5	noPredEA	97.58	97.68	97.43	97.41	96.94	96.89	91.43	91.42	80.31	81.69
	PredEA-LR	97.73	97.84	97.53	97.73	96.77	96.96	92.78	92.35	81.08	82.57
	PredEA-LR2	99.94	99.93	99.90	99.87	99.79	99.76	99.58	99.51	96.09	95.67
	PredEA-NLR	98.22	98.22	98.11	98.06	97.75	97.62	94.32	94.38	82.97	83.32
*NL*6	noPredEA	98.54	98.56	98.12	98.14	97.68	97.74	94.22	96.00	87.21	88.58
	PredEA-LR	99.18	99.17	98.89	98.70	98.05	98.03	95.29	96.12	88.92	90.15
	PredEA-LR2	99.87	99.85	99.79	99.78	99.62	99.61	99.34	99.30	98.61	98.52
	PredEA-NLR	99.54	99.36	99.28	99.24	99.15	99.10	98.94	98.23	97.72	97.41
*NL*7	noPredEA	99.53	99.55	99.51	99.52	99.43	99.46	99.16	98.94	95.14	95.79
	PredEA-LR	99.77	99.75	99.75	99.74	99.65	99.64	99.45	99.27	96.71	96.86
	PredEA-LR2	99.93	99.92	99.89	99.88	99.79	99.78	99.65	99.62	98.91	98.84
	PredEA-NLR	99.91	99.90	99.88	99.86	99.75	99.74	99.63	99.59	98.62	98.77

Table 8 *PredEA* and *noPredEA* results: dynamic knapsack for periodic, patterned and nonlinear change periods and for cyclic (C) and probabilistic (P) transitions between 3, 5, 10, 20 and 50 states

Knapsack		Number of states									
Period	Algorithm	3		5		10		20		50	
		C	P	C	P	C	P	C	P	C	P
$r = 10$	noPredEA	1,849.57	1,853.31	1,849.11	1,854.60	1,840.03	1,839.48	1,793.66	1,799.04	1,669.13	1,683.74
	PredEA-LR	1,859.57	1,863.93	1,861.35	1,867.86	1,854.73	1,855.57	1,808.89	1,814.83	1,683.73	1,699.14
	PredEA-LR2	1,860.81	1,864.91	1,861.65	1,868.83	1,855.56	1,855.96	1,809.42	1,815.18	1,685.07	1,699.16
	PredEA-NLR	1,859.94	1,864.75	1,861.52	1,868.70	1,855.24	1,856.36	1,809.11	1,815.12	1,683.92	1,699.28
$r = 50$	noPredEA	1,860.59	1,865.40	1,860.78	1,867.73	1,851.61	1,851.68	1,804.76	1,817.04	1,680.30	1,687.44
	PredEA-LR	1,864.37	1,870.19	1,865.64	1,873.56	1,858.86	1,860.07	1,813.40	1,825.64	1,688.24	1,695.47
	PredEA-LR2	1,864.36	1,870.26	1,866.72	1,874.17	1,859.04	1,860.16	1,813.60	1,825.59	1,688.30	1,695.80
	PredEA-NLR	1,864.77	1,870.24	1,866.22	1,873.78	1,858.98	1,860.35	1,813.61	1,825.76	1,688.29	1,695.52
$r = 100$	noPredEA	1,863.16	1,868.04	1,864.16	1,871.73	1,855.27	1,853.87	1,808.21	1,819.53	1,683.73	1,692.90
	PredEA-LR	1,865.51	1,870.31	1,866.95	1,875.17	1,859.85	1,859.26	1,814.54	1,825.57	1,689.64	1,698.64
	PredEA-LR2	1,865.98	1,869.74	1,866.95	1,875.19	1,860.13	1,859.42	1,814.67	1,825.55	1,689.71	1,698.99
	PredEA-NLR	1,865.82	1,868.54	1,867.20	1,875.57	1,860.34	1,859.33	1,814.83	1,825.58	1,689.65	1,698.78
$r = 200$	noPredEA	1,864.06	1,867.03	1,866.00	1,873.43	1,858.00	1,859.45	1,811.18	1,825.43	1,686.79	1,700.56
	PredEA-LR	1,865.57	1,868.47	1,867.84	1,875.88	1,860.81	1,862.25	1,815.27	1,828.88	1,690.74	1,704.27
	PredEA-LR2	1,965.64	1,868.56	1,867.97	1,875.92	1,861.62	1,862.26	1,815.67	1,829.18	1,690.92	1,704.76
	PredEA-NLR	1,865.58	1,868.54	1,867.95	1,875.91	1,861.08	1,862.26	1,815.48	1,829.28	1,690.80	1,704.55
5–10–5	noPredEA	1,833.57	1,846.72	1,846.18	1,850.01	1,836.80	1,838.11	1,790.34	1,796.52	1,665.79	1,681.70
	PredEA-LR	1,844.16	1,859.65	1,859.60	1,864.39	1,852.41	1,853.08	1,806.60	1,812.45	1,681.49	1,697.05
	PredEA-LR2	1,844.35	1,859.55	1,860.31	1,864.57	1,852.21	1,852.48	1,806.55	1,812.21	1,681.74	16,97.21
	PredEA-NLR	1,844.55	1,860.40	1,860.87	1,865.03	1,852.55	1,853.74	1,806.86	1,813.29	1,681.72	1,697.84

Table 8 continued

Knapsack		Number of states									
Period	Algorithm	3		5		10		20		50	
		C	P	C	P	C	P	C	P	C	P
10–20–10	noPredEA	1,837.92	1,853.56	1,850.43	1,855.41	1,840.53	1,841.94	1,794.42	1,808.12	1,664.74	1,679.17
	PredEA-LR	1,844.89	1,862.31	1,859.08	1,866.10	1,852.67	1,853.58	1,806.49	1,819.42	1,674.35	1,689.53
	PredEA-LR2	1,845.15	1,862.06	1,859.41	1,866.65	1,853.39	1,853.36	1,806.11	1,819.69	1,674.78	1,689.96
	PredEA-NLR	1,842.43	1,860.58	1,856.80	1,864.02	1,851.36	1,851.43	1,803.08	1,815.19	1,670.41	1,685.16
50–60–70	noPredEA	1,871.61	1,865.02	1,861.30	1,870.12	1,851.74	1,850.70	1,805.38	1,815.17	1,681.20	1,688.86
	PredEA-LR	1,874.44	1,868.35	1,865.55	1,875.33	1,858.45	1,858.15	1,812.47	1,822.49	1,687.39	1,695.53
	PredEA-LR2	1,874.50	1,868.63	1,865.82	1,875.58	1,858.65	1,858.24	1,812.26	1,822.54	1,687.90	1,695.57
	PredEA-NLR	1,874.54	1,868.74	1,865.65	1,875.55	1,858.94	1,858.21	1,812.74	1,822.91	1,687.65	1,695.86
100–150–100	noPredEA	1,856.57	1,867.50	1,863.97	1,871.71	1,856.16	1,855.20	1,809.02	1,819.75	1,684.55	1,693.54
	PredEA-LR	1,858.19	1,870.10	1,867.29	1,875.43	1,860.85	1,860.09	1,815.30	1,826.1,8	1,690.59	1,699.55
	PredEA-LR2	1,858.50	1,870.51	1,867.64	1,875.87	1,860.62	1,860.38	1,815.68	1,826.50	1,690.97	1,699.92
	PredEA-NLR	1,858.72	1,870.99	1,867.42	1,875.66	1,861.61	1,860.16	1,815.97	1,826.94	1,690.90	1,699.73
$NL1$	noPredEA	1,861.10	1,862.12	1,860.54	1,865.47	1,851.44	1,851.85	1,804.82	1,808.31	1,678.06	1,697.48
	PredEA-LR	1,861.74	1,862.58	1,861.04	1,865.50	1,852.61	1,852.20	1,805.95	1,809.62	1,679.88	1,699.66
	PredEA-LR2	1,866.16	1,868.09	1,867.90	1,874.67	1,861.64	1,862.74	1,816.58	1,819.01	1,687.57	1,707.09
	PredEA-NLR	1,866.05	1,867.93	1,867.68	1,874.38	1,861.57	1,862.84	1,816.24	1,819.65	1,687.50	1,707.01
$NL2$	noPredEA	1,857.70	1,861.36	1,858.66	1,863.41	1,850.50	1,850.46	1,801.34	1,815.55	1,670.50	1,696.64
	PredEA-LR	1,857.87	1,860.98	1,858.50	1,864.31	1,849.87	1,850.46	1,801.46	1,815.90	1,670.05	1,696.36
	PredEA-LR2	1,860.92	1,864.72	1,863.48	1,867.86	1,855.92	1,856.15	1,808.52	1,822.83	1,677.09	1,702.83
	PredEA-NLR	1,860.73	1,864.76	1,862.58	1,867.70	1,855.77	1,856.32	1,808.55	1,822.80	1,677.07	1,702.87
$NL3$	noPredEA	1,859.73	1,863.16	1,858.37	1,867.64	1,851.47	1,855.39	1,803.53	1,826.09	1,680.72	1,702.13
	PredEA-LR	1,859.44	1,863.93	1,858.61	1,869.44	1,853.03	1,857.05	1,806.57	1,830.53	1,682.81	1,704.85
	PredEA-LR2	1,862.21	1,866.43	1,862.95	1,872.96	1,856.30	1,861.04	1,808.05	1,832.12	1,684.35	1,706.25
	PredEA-NLR	1,863.31	1,867.82	1,863.68	1,873.22	1,857.80	1,861.93	1,809.82	1,832.89	1,684.44	1,706.71
$NL4$	noPredEA	1,861.56	1,861.67	1,858.09	1,864.67	1,856.50	1,850.37	1,799.61	1,816.59	1,650.37	1,708.75
	PredEA-LR	1,863.19	1,861.97	1,857.97	1,864.39	1,856.52	1,850.23	1,799.81	1,816.22	1,650.39	1,708.48
	PredEA-LR2	1,863.24	1,862.47	1,859.80	1,865.36	1,858.44	1,851.78	1,801.79	1,81,8.76	1,652.73	1,710.81
	PredEA-NLR	1,863.91	1,863.17	1,859.98	1,866.44	1,859.02	1,852.23	1,802.30	1,819.31	1,652.66	1,711.08
$NL5$	noPredEA	1,856.16	1,859.92	1,855.88	1,863.15	1,847.14	1,848.65	1,800.23	1,811.92	1,672.76	1,687.04
	PredEA-LR	1,855.89	1,859.95	1,855.20	1,864.30	1,847.89	1,847.56	1,799.90	1,811.13	1,672.64	1,687.24
	PredEA-LR2	1,861.02	1,873.78	1,865.08	1,876.06	1,856.13	1,858.54	1,817.39	1,823.36	1,688.68	1,696.88
	PredEA-NLR	1,857.13	1,860.34	1,856.21	1,865.81	1,849.07	1,850.22	1,800.92	1,812.13	1,673.35	1,688.92
$NL6$	noPredEA	1,858.68	1,863.10	1,857.65	1,864.21	1,849.06	1,848.64	1,801.98	1,808.94	1,678.03	1,688.56
	PredEA-LR	1,860.78	1,864.75	1,858.63	1,865.45	1,850.98	1,850.60	1,802.92	1,809.52	1,679.42	1,690.53
	PredEA-LR2	1,869.93	1,877.82	1,868.01	1,880.88	1,858.1,8	1,868.47	1,813.60	1,825.99	1,696.40	1,706.31
	PredEA-NLR	1,866.17	1,875.93	1,866.74	1,874.77	1,856.64	1,862.97	1,810.11	1,816.94	1,694.75	1,699.61
$NLR7$	noPredEA	1,865.08	1,868.65	1,865.01	1,872.72	1,856.64	1,855.36	1,808.31	1,812.43	1,684.40	1,695.10
	PredEA-LR	1,866.85	1,869.22	1,866.22	1,872.90	1,857.31	1,855.51	1,809.29	1,813.12	1,687.98	1,698.20
	PredEA-LR2	1,872.34	1,875.99	1,874.92	1,883.51	1,870.18	1,869.39	1,822.96	1,821.89	1,699.14	1,706.90
	PredEA-NLR	1,870.21	1,871.29	1,872.23	1,880.89	1,867.73	1,867.47	1,819.59	1,820.55	1,695.20	1,704.94

the performances obtained are, in the majority of the cases, equivalent. Tables 9 and 10 show that, in periodic change periods, the comparisons of the performances obtained by the three versions of PredEA are, in general, statistically equivalent. For the patterned change periods, the three proposed methods obtain similar performances, except for the 10-20-

Table 9 Statistical results: dynamic knapsack for periodic, patterned and nonlinear change periods and for cyclic (C) and probabilistic (P) transitions between 3, 5, 10, 20 and 50 states

Bit matching		Number of states									
Period	Pair of algorithms	3		5		10		20		50	
		C	P	C	P	C	P	C	P	C	P
$r = 10$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-NLR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
$r = 50$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	∼	∼	∼	∼	∼	∼	∼	∼	∼	++
	PredEA-NLR – PredEA-LR2	∼	∼	∼	−−	−−	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
$r = 100$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	∼	∼	∼	∼	++	∼	∼	∼	∼	+
	PredEA-NLR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
$r = 200$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	∼	++	∼	∼	++	∼	∼	∼	∼	∼
	PredEA-NLR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	∼	∼	∼	∼	−−	∼	∼	∼	∼	∼
5–10–5	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	++	∼	++	∼	∼	∼	∼	∼	∼	++
	PredEA-NLR – PredEA-LR2	∼	∼	∼	++	∼	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
10–20–10	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
	PredEA-NLR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
	PredEA-LR – PredEA-LR2	−−	∼	∼	∼	∼	∼	∼	∼	∼	∼
50–60–70	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-NLR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼

Table 9 continued

Bit matching		Number of states									
Period	Pair of algorithms	3		5		10		20		50	
		C	P	C	P	C	P	C	P	C	P
100–150–100	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	++	++	∼	∼	++	∼	∼	++	∼	∼
	PredEA-NLR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
$NL1$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	∼	∼	∼	∼	++	∼	∼	∼	∼	∼
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
$NL2$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	∼	++	∼	++	∼	∼	∼	∼	∼	∼
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
$NL3$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	∼	∼	∼	∼	++	++	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
$NL4$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	++	∼	++	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR2	∼	++	∼	++	++	∼	++	++	∼	∼
	PredEA-LR – PredEA-LR2	∼	−−	−−	−−	−−	−−	−−	−−	−−	−−
$NL5$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
	PredEA-LR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
$NL6$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	++	∼	∼	∼	∼	++	∼	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
	PredEA-LR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−

Table 9 continued

Bit matching		Number of states									
Period	Pair of algorithms	3		5		10		20		50	
		C	P	C	P	C	P	C	P	C	P
NL7	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	∼	∼	∼	∼	∼	∼	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
	PredEA-LR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−

Table 10 Statistical results: dynamic bit matching for periodic, patterned and nonlinear change periods and for cyclic (C) and probabilistic (P) transitions between 3, 5, 10, 20 and 50 states

Bit matching		Number of states									
Period	Pair of Algorithms	3		5		10		20		50	
		C	P	C	P	C	P	C	P	C	P
$r = 10$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-NLR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
$r = 50$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	++	++	∼	∼	++	++	++	++	++	∼
	PredEA-NLR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
$r = 100$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-NLR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
$r = 200$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-NLR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
5–10–5	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	++	++	++	∼	∼	++	∼	∼	++	++
	PredEA-NLR – PredEA-LR2	∼	∼	++	∼	∼	++	∼	++	∼	∼
	PredEA-LR – PredEA-LR2	−−	∼	∼	++	∼	∼	∼	∼	∼	∼

Table 10 continued

Bit matching		Number of states									
Period	Pair of Algorithms	3		5		10		20		50	
		C	P	C	P	C	P	C	P	C	P
10–20–10	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
	PredEA-NLR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
	PredEA-LR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
50–60–70	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	∼	∼	∼	∼	∼	∼	∼	∼	∼	++
	PredEA-NLR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	++
	PredEA-LR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
100–150–100	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-NLR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
$NL1$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR2	∼	−−	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
$NL2$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	∼	∼	∼	∼	∼	∼	∼	∼	∼	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
$NL3$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	∼	∼	∼	∼	++	∼	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
$NL4$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	∼	∼	∼	++	∼	∼	∼	∼	∼	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR2	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼
	PredEA-LR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−

Table 10 continued

Bit matching		Number of states									
Period	Pair of Algorithms	3		5		10		20		50	
		C	P	C	P	C	P	C	P	C	P
$NL5$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	∼	∼	∼	∼	∼	∼	++	++	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	++	++	∼	∼	++	∼	++	++	∼	∼
	PredEA-NLR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
	PredEA-LR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
$NL6$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	∼	∼	∼	∼	++	++	∼	∼	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
	PredEA-LR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
$NL7$	PredEA-NLR – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-LR – noPredEA	∼	∼	∼	∼	∼	∼	∼	∼	++	++
	PredEA-LR2 – noPredEA	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR	++	++	++	++	++	++	++	++	++	++
	PredEA-NLR – PredEA-LR2	−−	−−	−−	−−	−−	−−	−−	−−	−−	−−
	PredEA-LR – PredEA-LR2	−−	−−	−−	−−	−−	−−	∼	−−	−−	−−

10 change period. In this case, PredEA-NLR obtains the lowest prediction accuracy (66.76 %) when compared with the other two approaches (99.73 %). Consequently, as the results show, PredEA-NLR achieves significantly worse fitness than PredEA-LR and PredEA-LR2. Figure 5 shows the offline performance measured over time. While the performance of noPredEA remains nearly constant during the entire process, the other algorithms improve their fitness as time passes. Furthermore, PredEA-LR2 is more effective at the beginning of the process than the other algorithms. Figure 5 also shows that PredEA-NLR performs worse than PredEA-LR and PredEA-LR2. For the remaining patterned change periods, the three methods obtain similar prediction accuracies (above 99 %) and the performances obtained are, in general, equivalent. Finally, for the nonlinear change periods, as stated before, the fitness obtained by PredEA-LR is statistically comparable to the scores achieved by noPredEA. The reason for this is the weak prediction accuracy obtained by the linear predictor in the nonlinear change periods. The use of all information in the linear regression method is unsuitable to fit the data generated by the nonlinear functions. The performances obtained by PredEA-NLR are always statistically better than PredEA-LR's results. Although PredEA-LR fails in the nonlinear change periods, PredEA-LR2 obtains the best performances among all the techniques. The update of the regression line using only the two previous observations is enough to continuously fit the points of the nonlinear curves. This method performs evenly to PredEA-NLR for the $NL1$ through $NL4$ change periods, and is significantly superior in the remaining nonlinear change periods. Notice that the $NL1$ through $NL4$ change periods are generated using the nonlinear functions incorporated in the nonlinear regression predictor, benefiting PredEA-NLR. Figure 6 shows the evolution of the offline performance over time for the $NL5$ change period. It is obvious the superiority of PredEA-LR2 over the remaining methods. While PredEA-LR and PredEA-NLR are not able to evolve, PredEA-LR2 continuously improves its performance. Analyzing the plot, we can see that, around generation 2000, the offline measure of PredEA-LR2 for the cyclic case strongly increases. This happens because near generation 2000 the Markov model completes the learn of the dynamics of the environment, which is associated to the high prediction accuracy of PredEA-LR2 and leads to these superior results. For the cyclic case, the Markov model takes more time to learn the entire dynamics of the environment, so the performance increases slightly slower. For the nonlinear change periods, not only PredEA-LR2 obtains the highest results, but also provides a significantly faster performance of the EA. Since the predictions are provided using only two observations, the computational effort is considerably less than PredEA-LR that uses all the observations. Concerning the computational times PredEA-NLR is the most expen-

sive method since the parameter's estimation made by the GA is time consuming and is made in every environmental change.

As global conclusions we stress that the proposed prediction techniques significantly improve the performance of the EA in the studied dynamic environments. In fact, when the appropriate information is retrieved from the memory and inserted into the population *before* the change, the performance of the EA is significantly increased. Furthermore, the use of an appropriate time-window in the linear regression predictor resulted as the most robust and the most efficient method with the lowest computational effort.

8 Conclusions

When in the presence of environments that change following a repeated behavior, the use of prediction mechanisms is highly beneficial to the performance of memory-based EAs. Unlike other methods investigated so far, the proposed approaches are able to predict both *when* the next change will happen and *how* the environment will look like. By using past data, accurate predictions can be made and the algorithm can anticipate the change by introducing useful information into the population before such change takes place. Two different predictors were used to estimate when the next change would occur: one using linear regression, and another using nonlinear regression. The linear predictor was tested using all collected data and using only a part of the observations, enclosed by a time-window. All the proposed methods performed better than the algorithm without prediction. Moreover, the linear regression predictor using all the collected data, performed well for cyclic and patterned change periods, but failed in the presence of nonlinear change periods. On the other hand, the nonlinear predictor provided accurate predictions in all types of environments analyzed, except for the nonlinear change periods obtained by functions unknown to the module. The use of a time-window in the linear predictor was the most robust and efficient method: it allows the EA to obtain the best performances in all types of change periods, with the minimum computational cost. For the linear and patterned change periods, PredEA-LR2 performed equivalently to PredEA-LR and PredEA-LR10. For the nonlinear change periods, PredEA-LR2 outperformed the remaining techniques. The most relevant results of this method were obtained for the nonlinear change periods. While the other linear regression techniques failed and the nonlinear method found some difficulties, the use of a small time-window allowed to divide the nonlinear change period in small sets, which were correctly fitted by the linear regression method.

The second predictor was implemented using a Markov chain and estimated how the environment would change. The Markov chain stored information about the environments and the transitions among them. After that, this information was used to predict which possible environment(s) would appear in the next change. This predictor performed very well in the situations analyzed: it started by learning the dynamics of the environmental changes and, after that phase, the predictions provided were, for the most part, correct. When the number of different environments increased, the predictor needed more time to acquire all the necessary information in order to make valid predictions.

The investigated methods are advantageous for those problems where a repeated behavior can be found. In these cases, the investigated prediction techniques can improve the performance of the problem solver. In other type of problems the proposed approach can be used as a complement to the problem solver, working in the background, trying to identify possible patterns. So, even if a certain problem is mainly non-periodic, if in a certain moment some trend is observed, it will be detected and the benefits of the presented methods can be used. The fact that the new individuals chosen from the memory to be inserted into the population do not participate in the evolutionary process until the change effectively occurs, make them harmless if things go differently than predicted. Moreover, the prediction techniques are not yet studied for problems that have a stochastic nature or present time-linkage dependences. Examples of real world problems that present a repeated behavior and where our model could be tested, are the class of dynamic shortest path routing problems, e.g., in mobile ad hoc networks or dealing with traffic jams in cities.

The proposed methods, although conferring superior performances to the EA for the studied situations, present several limitations and need to be improved and further tested. So, as future work, it is crucial to improve the Markov chain module, removing the problem dependence. We are exploring different approaches to measure the environment and store the appropriate information in the Markov model. This information will be used to implicitly identify an environment as a new one or as a known one. Other required improvement concerns the choice for the size of the time-window. In fact, PredEA-LR2 and PredEA-LR10 worked well for the selected functions, but it is not certain that these time-windows are the best choices if different functions are used to generate the change periods. If these two issues are improved the proposed method can be used and tested in different problems, using different representations, such as real-valued. Moreover, the proposed techniques will be used in other types of algorithms besides memory-based, such as multi-population approaches and will be tested in a real word application.

References

Barlow GJ, Smith SF (2008) A memory enhanced evolutionary algorithm for dynamic scheduling problems. In: Applications of evolutionary computing. Lecture notes in computer science, vol. 4974. Springer, Berlin, pp 606–615

Ben-Romdhane H, Alba E, Krichen S (2013) Best practices in measuring algorithm performance for dynamic optimization problems. Soft Comput 17(6):1005–1017

Bosman PAN, La Poutré H (2006) Computationally intelligent online dynamic vehicle routing by explicit load prediction in evolutionary algorithm. In: Proceedings of parallel problem solving from nature. Lecture notes in computer science, vol 4193. Springer, Berlin, pp 312–321

Bosman PAN, La Poutré H (2007) Inventory management and the impact of anticipation in evolutionary stochastic online dynamic optimization. In: Proceedings of the IEEE congress on evolutionary computation. IEEE Press, New York, pp 268–275

Branke J (2002) Evolutionary optimization in dynamic environments. Kluwer Academic Publishers, Dordrecht

Cobb HG, Grefenstette JJ (1993) Genetic algorithms for tracking changing environments. In: Proceedings of the fifth international conference on genetic algorithms. Morgan Kaufmann, Menlo Park, pp 523–530

Cruz C, Gonzlez J, Pelta D (2011) Optimization in dynamic environments: a survey on problems, methods and measures. Soft Comput 15(7):1427–1448

De Jong K (2006) Evolutionary computation: a unified approach. MIT Press, Massachusetts

Goldberg DE, Smith RE (1987) Nonstationary function optimization using genetic algorithms with dominance and diploidy. In: Grefenstette JJ (ed) Proceedings of the second international conference on genetic algorithms. Lawrence Erlbaum Associates, UK, pp 59–68

Grefenstette JJ (1992) Genetic algorithms for changing environments. In: Männer R, Manderick B (eds) Parallel problem solving from nature (PPSN II)

Hatzakis I, Wallace D (2001) Dynamic multi-objective optimization with evolutionary algorithms: a forward-looking approach. In: Proceedings of the genetic and evolutionary computation conference. ACM Press, New York, pp 1201–1208

Karaman A, Uyar S, Eryigit G (2005) The memory indexing evolutionary algorithm for dynamic environments. In: Applications of evolutionary computing. Lecture notes in computer science, vol 3449. Springer, Berlin, pp 563–573

Li C, Yang S (2012) A general framework of multipopulation methods with clustering in undetectable dynamic environments. IEEE Trans Evol Comput 99:1

McCabe GP, Moore DS (2003) Introduction to the practice of statistics. Freeman and Company, Ohio

Michalewicz Z, Schmidt M, Michalewicz M, Chiriac C (2007) Adaptive business intelligence: three case studies. In: Yang S, Ong Y-S, Jin Y (eds) Evolutionary computation in dynamic and uncertain environments. Studies in computational intelligence, vol 51. Springer, Berlin, pp 179–196

Nash JC, Walker-Smith M (1987) Nonlinear parameter estimation: an integrated system in BASIC. Marcel Dekker Inc., New York

Ng P, Wong KC (1995) A new diploid scheme and dominance change mechanism for nonstationary function optimization. In: Proceedings of the Sixth International Conference on Genetic Algorithms. Morgan Kaufmann, Menlo Park, pp 159–166

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

Norris JR (1997) Markov chains. In: Cambridge series in statistical and probabilistic mathematics. Cambridge University Press, Cambridge

Pan Z, Chen Y, Kan L, Zhang Y (1995) Parameter estimation by genetic algorithms for nonlinear regression. In: Proceedings of the international conference on optimization techniques and applications. World Scientific, Singapore, pp 946–953

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

Rohlfshagen P, Yao X (2009) The dynamic knapsack problem revisited: a new benchmark problem for dynamic combinatorial optimization. In: Applications of evolutionary computing. Lecture notes of computer science, vol 5484. Springer, Berlin

Rossi C, Abderrahim M, Daz JC (2008) Tracking moving optima using Kalman-based predictions. Evol Comput 16(1):1–30

Schmidt M, Michalewicz Z, Michalewicz M, Chiriac C (2005) Prediction and optimization in a dynamic environment: a case study. In: Proceedings of the IEEE congress on evolutionary computation, vol 1. IEEE Press, New York, pp 781–788

Simões A, Costa E (2007) Improving memory's usage in evolutionary algorithms for changing environments. In: Proceedings of the IEEE congress on evolutionary computation. IEEE Press, New York, pp 276–283

Simões A, Costa E (2007) Variable-size memory evolutionary algorithm to deal with dynamic environments. In: Applications of evolutionary computing. Lecture notes in computer science, vol 4448. Springer, Berlin, pp 617–626

Simões A, Costa E (2008) Evolutionary algorithms for dynamic environments: prediction using linear regression and markov chains. In: Parallel problem solving from nature (PPSN X). Lecture notes on computer science, vol 5199. Springer, Berlin, pp 306–315

Simões A, Costa E (2009) Improving prediction in evolutionary algorithms for dynamic environments. In: Proceedings of the genetic and evolutionary computation conference. ACM Press, New York, pp 875–882

Simões A, Costa E (2009) Prediction in evolutionary algorithms for dynamic environments using markov chains and nonlinear regression. In: Proceedings of the genetic and evolutionary computation conference. ACM Press, New York, pp 883–890

Simões A, Costa E (2012) Virtual loser genetic algorithm for dynamic environments. In: Di Chio C, et al. (eds) Applications of evolutionary computing. Lecture notes on computer science, vol 7248. Springer, Berlin, pp 539–548

Stroud PD (2001) Kalman-extended genetic algorithm for search in nonstationary environments with noisy fitness evaluations. IEEE Trans. Evol. Comput. 5(1):66–77

Ursem RK (2000) Multimodal optimization techniques in dynamic environments. In: Whitley D et al (eds) Proceedings of the genetic and evolutionary computation conference. Morgan Kaufmann, Menlo Park, pp 19–26

Uyar AS, Harmanci AE (2002) Preserving diversity in changing environments through diploidy with adaptive dominance. In: Langdon WB et al (eds) Proceedings of the genetic and evolutionary computation conference. Morgan Kaufmann, Menlo Park, p 679

Uyar AS, Harmanci AE (2005) A new population based adaptive dominance change mechanism for diploid genetic algorithms in dynamic environments. Soft Comput 9(11):803–814

van Hemert J, Van Hoyweghen C, Lukshandl E, Verbeeck K (2001) A futurist approach to dynamic environments. In: Genetic and evolutionary computation conference, EvoDOP workshop, pp 35–38

Weicker K (2003) Evolutionary algorithms and dynamic optimization problems. Der Andere Verlag, Munchen

Yang S (2005) Memory-based immigrants for genetic algorithms in dynamic environments. In: Beyer H.-G. (ed) Proceedings of the genetic and evolutionary computation conference, vol 2. ACM Press, New York, pp 1115–1122

Yang S.: A comparative study of immune system based genetic algorithms in dynamic environments. In: Proceedings of the genetic and evolutionary computation conference. ACM Press, New York, pp 1377–1384

Yang, S (2006) Dominance learning in diploid genetic algorithms for dynamic optimization problems. In: Proceedings of the genetic and evolutionary computation conference. ACM Press, New York, pp 1435–1436

Yang S (2007) Explicit memory schemes for evolutionary algorithms in dynamic environments. In: Yang S, Ong Y-S, Jin Y (eds) Evolutionary computation in dynamic and uncertain environments. Studies in computational intelligence, vol 51. Springer, Berlin, pp 3–28

Yang S (2008) Genetic algorithms with memory- and elitism-based immigrants in dynamic environments. Evol Comput 3(16):385–416

Yang S, Yao X (2008) Population-based incremental learning with associative memory for dynamic environments. IEEE Trans Evol Comput 5(12):542–561

Younes A, Basir O, Calamai P (2006) A hybrid evolutionary approach for combinatorial problems in dynamic environments. In: Proceedings of the Canadian conference on electrical and computer engineering. IEEE Press, New York, pp 1595–1600

Zhou A, Jin Y, Zhang Q, Sendhoff B, Tsang E (2007) Prediction-based population re-initialization for evolutionary dynamic multi-objective optimization. In: Evolutionary multi-criterion optimization. Lecture notes in computer science, vol 4403. Springer, Berlin, pp 832–846