Prediction in Evolutionary Algorithms for Dynamic Environments using Markov Chains and Nonlinear Regression

Anabela Simões, Ernesto Costa

CISCUC
Centro de Informática e Sistemas da Universidade de Coimbra
Prediction in Evolutionary Algorithms for Dynamic Environments using Markov Chains and Nonlinear Regression

Anabela Simões 1,2  Ernesto Costa 2

1Dept. of Informatics and Systems Engineering
ISEC - Coimbra Polytechnic
Rua Pedro Nunes - Quinta da Nora
3030-199 Coimbra – Portugal

2Centre for Informatics and Systems of the University of Coimbra
Polo II – Pinhal de Marrocos
3030 - 290 Coimbra – Portugal
abs@iseec.pt, ernesto@dei.uc.pt

CISUC TECHNICAL REPORT TR 2009/01 - ISSN 0874-338X  JANUARY 2009

Abstract. The inclusion of prediction mechanisms in Evolutionary Algorithms (EAs) used to solve dynamic environments allows forecasting the future preparing the algorithm to the changes. Prediction is a difficult task, but if some recurrence is present in the environment, it is possible to apply statistical methods which use information from the past to estimate the future. In this work we enhance a previously proposed computational architecture, incorporating a new predictor based on nonlinear regression. The system uses a memory-based EA to evolve the best solution and a predictor module based on Markov chains to estimate which possible environments will appear in the next change. Another prediction module is responsible to estimate when next change will happen. In this work important enhancements are introduced in this module, replacing the linear predictor by a nonlinear one. The performance of the EA is compared using no prediction, using predictions supplied by linear regression and by nonlinear regression. The results show that this new module is very robust allowing to accurately predicting when next change will occur in different types of change periods.

Keywords: Evolutionary Algorithms, Dynamic Environments, Prediction, Markov chains, Nonlinear Regression

1 Introduction
Evolutionary Algorithms (EA) have been successfully used to solve different types of optimization problems in static environments. When the environment changes along time, the optimum we are looking for will, in general, be different. Due to its characteristics the algorithm will have difficulties to redirect the search in order to find the new optimum, or near optimum, in acceptable time. That is why some modifications were proposed to cope with that problem: the use of memory to keep track of past good solution's candidates, techniques to maintain a high level of population's diversity or mechanisms relying on different populations, each one tuned for a particular environment (see [29] for a review on those alternatives). In certain cases, the dynamics of the environment's changes are not chaotic but follow instead a certain repeating pattern and, in those cases, we may expect to be able to predict both the time and the trend of the modification. If we succeed on that endeavor we may avoid the sudden decrease in performance that typically results from the change in the environment improving thus the algorithm's adaptability. In this paper we extend some ideas we develop recently and which are based on two predictors: one, based on regression, to estimate when the next change will take place and, the other one, supported by a Markov chain model, to predict how the environment will be different. The main goal of this paper will be to evaluate experimentally a predictor based on nonlinear regression and to compare its performance with one supported by a linear regression mechanism. They will be combined with the predictor involving a Markov Chain and will be tested in connection with different types of dynamic environments.

The remaining text is organized as follows: section 2 describes related work concerning prediction and anticipation used by EAs in the context of dynamic environments. In section 3 we explain the overall architecture of an EA that utilizes the two prediction modules. The nonlinear regression predictor is explained in section 4. In section 5 we present the experimental
setup used to test the investigated ideas. Experimental results are summarized in section 6. We conclude with some remarks and ideas for future work.

2 Related Work

Recently, several studies concerning anticipation in changing environments using EAs have been proposed. The main goal of these approaches is to estimate future situations and so decide the algorithm’s behavior in the present. Since information about the future typically is not available, it is attained through learning from past situations.

Branke et al. [3] try to understand how the decisions made at one stage influence the problems encountered in the future. Future changes are anticipated by searching not only for good solutions but also for solutions that additionally influence the state of the problem in a positive way. These so-called flexible solutions are easily adjustable to changes in the environment. Studies on the tardiness job-shop problem, with jobs arriving on-deterministically over time, showed that avoiding early idle times increases flexibility, and thus the inclusion of an early idle time penalty as secondary objective into the scheduling algorithm can significantly enhance the system’s performance.

Stroud [25] used a Kalman-Extended Genetic Algorithm (KGA) in which a Kalman filter is applied to the fitness values associated with the individuals that make up the population. This is used 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 information contained within the population of solutions allows efficient discovery of better-adapted solutions. The ability of the KGA to continually find near-optimal solutions is demonstrated at several levels of process and observation noise.

Van Hemert et al. [26] introduced an EA with a meta-learner to estimate at time t how the environment will be at time t+1. 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 about the future is made based on observations from the past using two types of predictors: a perfect predictor and a noisy predictor. In reality they should not be called predictors. Concerning the former, the correct optimal value at the future time step is given to the solver, and for the latter the noisy predictor just provides the system noisy values as the optimal solution for the next step. The idea was tested with two benchmark problems: the knapsack problem and the Osmera’s function. Results showed that future prediction was useful in the Osmera’s function and more experimentation is claimed necessary to assess about the usefulness of prediction in the knapsack problem. As far as we know this work was not further explored.

Bosman [5, 6, 7, 8] proposed in the last years several approaches focused on the importance of using learning and anticipation in online dynamic optimization. These works analyze the influence of time-linkage present in problems such as scheduling and vehicle routing. The presence of time-linkage in this kind of problems 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 propose 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 of its parameters.

Rossi et al [17] compare different techniques to improve the search for tracking a moving optimum using the information provided by a predictor mechanism using Kalman filters. The used predictor assumes that the changes in the environment are not random and can be learned, helping the EA to keep track of the current optimum.

The use of linear regression to predict the moment of next change was initially proposed by Simões and Costa [21]. The idea was tested with different dynamic optimization problems, using a variable-size memory EA [20]. Several issues were analyzed such as the speed or the severity of change. The results showed that, if some pattern can be found in the changes of the environment, the predictor gives accurate estimations that can be used to enhance the EA’s adaptability to future situations [21]. Later, in [23] a predictor based on Markov chain was added and used to predict which environments may appear in the future.

3 Using Prediction in the Evolutionary Algorithm

Simões and Costa [23] proposed a computational model called PredEA to deal with dynamic. The proposed architecture uses a traditional EA that evolves a population of individuals that aim to optimize the current fitness function. A memory is used to store useful information from the past that is used in future changes. The traditional memory-based EA was extended with two prediction modules. The first module uses information about when previous changes have occurred to estimate the generation when the next change will be observed. In previous work, predictions provided by this module were
made by a linear regression predictor [23]. The second, using Markov chains, keeps track of previous environments and provides predictions about which environments will appear in the future. A different module (anticipation) uses the information provided by the previous two modules and prepares the EA for the next change.

Figure 1 illustrates the proposed architecture.

![Computational architecture of the PredEA](image)

Here is a brief description about each one of the components of Figure 1:

- **Evolutionary Algorithm**: standard evolutionary algorithm which evolves a population of individuals through the application of selection, crossover and mutation;

- **Memory**: stores the best individual of the population in a certain moment. The memorized individuals are associated with the environments where they were the best solution. The memory is updated from time to time as suggested by [0];

- **Predictor 1 (P1)**: this module saves the generations where different changes have occurred and uses this information to foresee when next change may take place. The predictor uses a standard linear regression technique;

- **Predictor 2 (P2)**: every time a different environment appears this module stores the environmental information. It consists in a set of states (each state corresponds to a different environment), a matrix of state transition probabilities and the initial probability vector. Each state corresponds to a different environment. The initial probability vector is initialized choosing randomly the initial state. The state transition probability matrix starts filled with zeros and is updated on-the-fly when different environments appear. When this module is called, it uses all the available information to estimate to which environment(s) the system will change;

- **Anticipation module (A)**: this module manages all the information provided by the two predictors and decides when to activate the mechanisms to prepare the EA to the next change. At that time, information from memory is retrieved and inserted into the population. This information corresponds to those individuals that can be useful to the next predicted environments.

Next sections detail each one of these modules.

### 3.1 Evolutionary Algorithm

It's a standard memory-based EA. A main population of individuals evolve 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. When a change happens or is predicted, the information stored in memory is retrieved and used to help the EA to re-adapt to the new environment.
3.2 Memory

Memory is used to store best individuals of the current population. It starts empty and has a limited size (20 individuals). The update time, $T_M(t)$, is computed as suggested in [28]: $T_M(0)=\text{rand}(5,10)$ and $T_M(t)=T_M(t-1) + \text{rand}(5,10)$.

An individual is stored in two situations:

- if the environment changed in the meantime and no individual related to this environment was previously stored.
- if an individual already exists in memory for the current environment, but it is worst than current best, the latter individual replaces the former in memory.

If memory is full we replace the most similar individual, in terms of Hamming distance by the current best if it is better [0].

This way we maximize the capacity of the memory to keep an individual for each different environment.

This scheme, called generational replacing strategy, was proposed in [22] and proved to be very efficient in memory-based EAs for changing environments. This scheme optimizes the capacity of the memory storing the best individual found so far for the present environment. When the memory becomes full, the similar individual is replaced by the best individual in the main population.

Memory is also used to detect changes in the environment: a change occurs if at least one individual of the memory has its fitness changed.

3.3 Predictor 1 module (P1)

This predictor uses information about when previous changes were observed to estimate when next change will occur. Previous work used a linear regression predictor to perform this task. The first two changes of the environment are stored after they happen (no prediction can be made yet). Based on these two values, a first approximation of the regression line can be built and the regression module starts providing the predictions about the next possible moment of change. Then, each time a change occurs the regression line is updated.

If we assume that the changes in the environment are periodic or following a certain pattern that is repeatedly observed. In these situations it makes sense to use linear regression to predict the generation when next change will take place, based on previous observations.

If the changes in the environment occur in generations which follow a nonlinear function, the predictions made by P1 are incorrect. In this work a nonlinear predictor will be used to deal with both situations.

3.4 Predictor 2 module (P2)

This module consists in two parts. One is hidden for the system (unknown to the algorithm) and is built offline with the following information: the maximum number of different states that may appear and a probability matrix of transitions that is used to model how the environments will change.

The second part is built on-line by the algorithm and is used to make predictions. It keeps track about the different environments and estimates which environments may appear in the next change. These 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 another. Associated to each transition it’s 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 along time.

The information that is stored about each environment is problem dependent. For instance if the dynamic bitmatching problem is being used, each state corresponds to a different template.

Example

In Figure 2 we show an example of a possible situation where there are four different states and transitions, together with the corresponding templates. Assume that the different states will appear following the sequence illustrated in figure 2 (left). If the problem to solve is the dynamic bitmatching problem, this sequence has a corresponding succession of binary templates that the EA must optimize (Figure 2, right). Each transition has an associated probability that is used to probabilistic decide how the environment will change. This is the first part (hidden) of the P2 module.
The second part of the P2 module, which is built on-line, uses a transition matrix that starts with all positions filled with zero, and when a change is detected, the probability value is updated according to the number of transitions that already exist from that state. The sum of the probabilities in the same row of the matrix must be 1. Each element $p_{ij}$ of the matrix $P$ is calculated as follows:

$$p_{ij} = \frac{C(i, j)}{\sum_j C(i, j)}$$

Where $C(i, j)$ is the number of times that state $j$ follows state $i$.

Ideally, after some generations and environmental changes our algorithm will construct a Markov model identical to the hidden one. From then on, the next state(s) can be correctly predicted making possible the introduction of important information before change, allowing the continuous adaptation of the EA to the new conditions.

For example, assume that 4 different environments are supposed to appear according to the following initial and transition probabilities matrices (this is the hidden information):

$$\lambda = \{1.0, 0.0, 0.0, 0.0, 0.0\}$$

$$P = \begin{pmatrix}
0.0 & 1.0 & 0.0 & 0.0 \\
0.5 & 0.0 & 0.5 & 0.0 \\
0.3 & 0.4 & 0.0 & 0.3 \\
0.2 & 0.2 & 0.6 & 0.0
\end{pmatrix}$$

Let’s see how the algorithm is updated step by step (the $P'$ matrix is updated every time a change is detected) and how the predictions about which environments may appear in the future are made:

**Step 1:** initial state is 1 (defined in vector $\lambda$)

$$P^{(0)} = \begin{pmatrix}
0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0
\end{pmatrix}$$

**Step 2:** go to state 2 (according to matrix $P$, probability $p_{1,2}$)

$$P^{(1)} = \begin{pmatrix}
0.0 & 1.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0
\end{pmatrix}$$
Step 3: probabilistically chooses to go to state 3 or 1 (3 was assumed as the chosen state)

\[
P_{23} = \begin{pmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}
\]

Step 4: probabilistically chooses to go to state 1, 2 or 3 (2 was assumed as the chosen state)

\[
P_{32} = \begin{pmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}
\]

Step 5: probabilistically chooses to go to state 3 or 1 (1 was assumed as the chosen state)
At this step, since state 2 already appeared in the past, the Markov chain module gives as prediction a possible transition to state 3 (known information until the present moment)

\[
P_{23} = \begin{pmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}
\]

Step 6: go to state 2
At this step, since state 1 already appeared in the past, the Markov chain module gives as prediction a possible transition to state 2 (known information until the present moment)

\[
P_{32} = \begin{pmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}
\]

Continuing this reasoning, the matrix \(P'\) will evolve towards the values of \(P\) defined offline at the beginning (unknown information to the EA):

\[
P = \begin{pmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.5 & 0.0 \\ 0.3 & 0.4 & 0.0 & 0.3 \\ 0.2 & 0.2 & 0.6 & 0.0 \end{pmatrix}
\]

3.5 Anticipation Module
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 P1 module. This "good moment" is provided by the P2 module which estimates the generation when next change will be observed. Knowing this value, the system starts preparing the change some generations before. If the prediction mechanisms are accurate and the correct information is introduced in the main population before the change, the EA's performance is not affected by the changes in the environment and it continues evolving completely
readapted to the new conditions. 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 change is actually detected if at least one individual in the memory changes its fitness (as it is suggested in [29]).

A parameter, called $\Delta$, is used to decide how many generations before the predicted moment of change (i.e., the $A$ module must be activated. The value of this parameter is also used to cover minor prediction errors associated to the $P_1$'s estimations. The value of $\Delta$ must be chosen in order to assure that the $A$ module is activated to prepare the population before the change.

When used with a linear regression predictor the module $P_1$ works correctly if the change period follows a linear (or close to linear) trend [23]. For instance, if the environment changes in a periodic manner, every $r$ generations, then the predicted values are precise. If a different pattern is observed in the change period, there will be an error associated to the values provided by the $P_1$ module. Initially the value of $\Delta$ was chosen off-line and kept constant during the entire run. This method was a limitation to the predictor's efficacy and different methods of adjusting the value of $\Delta$ during the run were proposed and successfully tested [24].

The anticipation process consists in retrieving from memory individuals that were good solutions in the environments that the $P_2$ module indicates as the next to appear. These individuals are inserted into the main population at the generation $g-\Delta$, replacing the worst individuals. If the $P_2$ module doesn't provide any prediction, five random individuals from memory are inserted into the population, replacing five randomly selected.

### 3.6 PredEA Pseudocode

The Pseudocode of the PredEA is detailed in Figure 3.

```plaintext
PredEA_improved(max, markov, initial-state)
1. Randomly create initial population
2. Create empty memory
3. Initialise $\Delta = 5$
4. Create the transition matrix with max states filled with zeros
5. repeat
6. Evaluate population
7. Evaluate memory
8. If time to update memory then
9. Store best individual
10. Set next time to update memory
11. If an environmental change happens then
12. Store performance measures
13. Activate the $P_1$ module
   i. Update $P_1$ information
   ii. Predict when next change will occur ($g$)
14. Update the value of $\Delta$
15. Update the algorithms Markov transition matrix ($P^*$)
16. If $g$ (next_change) is close (as defined by $g - \beta$) then
17. Activate the $P_2$ module
   i. Predict next state(s)
18. Activate the $A$ module
   i. Search memory for best individual(s)
   ii. Introduce those individuals into population
19. Perform selection, crossover and mutation
20. Define next population
21. until stop condition
```

**Figure 3. PredEA pseudocode**

The efficacy of the linear predictor can be compromised if the environments changes following a nonlinear trend. In this work we propose the replacement of the predictor by an equivalent based on nonlinear regression. This predictor will be able to make predictions in environments changing periodically or following linear and nonlinear patterns.
4 Nonlinear Regression

Usually, linear regression is used to model relationships between variables that follow a linear correlation. Some nonlinear functions can be modeled using linear regression (e.g., polynomial regression) but nonlinear regression is often used in these cases because it allows modeling a wide range of functions. In next sections we present the techniques for modeling data that displays nonlinear behaviors and use functions that are nonlinear in the model parameters.

4.1 The Nonlinear Regression Basics

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 $x$. Nonlinear regression is characterized by the fact that the prediction equation depends nonlinearly on one or more unknown parameters. The basic form for a nonlinear model between the response $y$ and a predictor $x$ is given as:

$$ y_i = f(x_i, \theta) + e_i $$

(1)

where $y_i$ and $x_i$ are the data, $f$ is a nonlinear function involving the predictor and the parameter vector $\theta$ and $e_i$ a random error [18].

For instance, let's assume the asymptotic regression model:

$$ f(x) = \theta_i - \theta_i x, \text{ where } 0 < \theta_i < 1 $$

(2)

Figure 4 shows to examples of curves using different values for the parameters $\theta_i$.

![Asymptotic regression model](image)

Figure 4. The asymptotic regression model

Although nonlinear regression is less intuitive and more complicated to use than linear regression, it is more powerful because it allows predictions in both linear and nonlinear data and are more suitable to model information of real world which is in general nonlinear. The difficult task in nonlinear regression is to estimate the correct values for the parameter vector $\theta$. Once estimated the parameters, predictions can be performed using the nonlinear function. Next section briefly explores the techniques for estimating the nonlinear parameters.

4.2 Parameter Estimation in Nonlinear regression

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 [18].

A nonlinear parameter estimation problem is an optimization problem which goal is to minimize the sum of squared errors given by eq. 3:
\[ \text{Sum}_{i=1}^{n} (y_i - f(x_i, \theta))^2 \]  

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 [18, 15].

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.

4.3 Nonlinear Regression in PredEA

In this work we propose a new method, which works with nonlinear regression, to use in the P1 module showed in Figure 1. Using a nonlinear based predictor we achieved more robust and accurate predictions for the moment of next change in a wider range of situations. The P1 module has a set of functions \( f_1, f_2, \ldots, f_n \) that can be used to give the predictions. At time \( t \), only one function is active. The choice of the active function is made measuring the prediction errors of all functions. The function with lower error is the selected one. The vector parameter \( \theta \) is estimated using a standard GA as proposed by [16]. This GA uses the available information from the past to find the values for \( \theta \), which minimize the errors of equation 3. Every time a change occurs in the environment and additional information is available, the GA is executed to find a vector \( \theta \) that better fits the data. The GA evolves a population of binary strings which correspond to different values for \( \theta \). The fitness function that the GA has to minimize is the least squares error function (eq. 3). Because individuals with higher fitness are selected more often, after some generations the best individual represents the optimal solution for \( \theta \). The vector \( \theta \) is estimated using only the known data. Using these estimated parameters and the selected function, the P1 module predicts when next change will occur.

After the real change happens, the prediction error is computed. If the error is superior to an established threshold \( \alpha \), the P1 module analyzes all the available functions to see if this error can be reduced. If so, a different function is used for future predictions. These actions correspond to the step 13f of the algorithm given in Figure 2. Figure 5 shows how the proposed module works. At the beginning, the first function is selected randomly.

![Diagram](attachment:image.png)

**Figure 5. The Nonlinear Regression Module**

5 Experimental Design

The algorithm described before was tested and compared with similar algorithms: an EA without any prediction mechanism, and the PredEA using linear regression in the P1 module.

The identification of these three algorithms will be:
- **PredEA-NLR**: memory-based evolutionary algorithm using predictors based on Markov chains and nonlinear regression;
- **PredEA-LR**: memory-based evolutionary algorithm using predictors based on Markov chains and linear regression;
- **NoPredEA**: memory-based evolutionary algorithm without prediction mechanisms.

### 5.1 Benchmark Problems

The benchmarks used to test the investigated ideas were the **dynamic bitmatching** problem and the **dynamic knapsack problem**.

The first problem can be described as: given a binary template, the individual’s fitness is the number of bits matching the specified template. A set of different binary templates is generated at the beginning of the run. When a change happens, a different template is chosen from that set. In our experiments we used templates of length 100.

The knapsack problem is a NP-complete combinatorial optimization problem often used as benchmark. It consists in selecting a number of items to a knapsack with limited capacity. Each item has a value ($v_i$) and a weight ($w_i$) and the objective is to choose the items that maximize the total value, without exceeding the capacity of the bag:

$$
\max v(x) = \sum_{i=1}^{n} v_i x_i
$$

subject to the weight constraint:

$$
\sum_{i=1}^{n} w_i x_i \leq C
$$

We used a knapsack problem with 100 items using strongly correlated sets of randomly generated data constructed in the following way [12, 28]:

$$
w_i = \text{uniformly random integer} [1, 50] \quad (6)
$$

$$
v_i = w_i + \text{uniformly random integer} [1, 5] \quad (7)
$$

$$
C = 0.6 \times \sum_{i=1}^{100} w_i \quad (8)
$$

The fitness of an individual 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 penalized in order to ensure that invalid individuals are distinguished from the valid ones. The fitness function is defined as follows:

$$
f(x) = \begin{cases} 
\sum_{i=1}^{100} v_i x_i, & \text{if } \sum_{i=1}^{100} w_i x_i \leq C \\
10^{-10} \times \left( \sum_{i=1}^{100} w_i - \sum_{i=1}^{100} w_i x_i \right), & \text{else}
\end{cases} \quad (9)
$$

Different capacities are created starting from the first capacity $C$ provided by equation (8). Each different created capacity value differs from the previous one from plus 10% or less 10%. The different capacities are generated at the beginning of the run and, when a change happens, a different value is chosen from that set.

### 5.3 Experimental Setup

#### 5.3.1 Parameters of the Evolutionary Algorithm

The EA’s parameters were set as follows: generational replacement with elitism of size one, tournament selection with tournament of size two, uniform crossover with probability $p_c=0.7$ and flip mutation applied with probability $p_m=0.01$. Binary representation was used with chromosomes of size 100 (size of the binary templates and number of items for the
knapsack problem). Population of 80 individuals and a memory of 20 individuals were used. The value of the \( \Delta \) constant referred above was 5 generations. Table 1 summarizes the EA’s settings:

<table>
<thead>
<tr>
<th>EA parameters</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual’s</td>
<td>binary</td>
</tr>
<tr>
<td>representation</td>
<td>uniformly randomly created</td>
</tr>
<tr>
<td>initialization</td>
<td>80</td>
</tr>
<tr>
<td>population size</td>
<td>20</td>
</tr>
<tr>
<td>memory size</td>
<td>uniform, probability 70%</td>
</tr>
<tr>
<td>crossover</td>
<td>flip, probability 1%</td>
</tr>
<tr>
<td>mutation</td>
<td>tournament, size 2</td>
</tr>
<tr>
<td>parent’s selection</td>
<td>generational with elitism of size one</td>
</tr>
<tr>
<td>survivors’ selection</td>
<td>the number of generations necessary for 200 environmental changes</td>
</tr>
<tr>
<td>stop criterion</td>
<td>maximize matching with template, maximize knapsack profit</td>
</tr>
<tr>
<td>goal</td>
<td>Constant = {5, 10, 25}, Adaptable = {Max_Err, Av_Err+, Av_Err(all), Max_Av_Err}</td>
</tr>
</tbody>
</table>

Table 1. Evolutionary Algorithm’s parameters’ settings

5.3.2 Parameters of the GA for nonlinear regression parameter’s estimation

The GA used to estimate the nonlinear regression parameters was run for 100 generations, using standard parameter settings: population of 30 individuals, generational replacement with elitism of size one, tournament selection with tournament of size two, one-point crossover with probability \( p_c=0.75 \) and flip mutation applied with probability \( p_m=0.05 \) (as suggested in [16]).

5.3.3 Parameters of the P1 module

The nonlinear predictor was used with two functions which can model both linear and nonlinear patterns.

\[
f_1(x) = \theta_1 x^2 + \theta_2 x + \theta_3 \tag{10}
\]

\[
f_2(x) = \frac{\theta_1 x}{\theta_1 + \theta_2 x} \tag{11}
\]

The results presented in section 6 were obtained using the \( \Delta \) adjusting method called Max_Av_Err ([24]) either in PredEA-NLR and PredEA-LR. This method updates the value of \( \Delta \) using the maximum prediction error and the average of the positive prediction errors and was the approach that allowed the EA to obtain the best results.

The parameter \( \alpha \) used to choose which function will make the predictions was set to 10. If the predicted error is superior to \( \alpha \), a new evaluation of the model is done and the module can change the active function if its application reduces the prediction error.

5.3.4 Parameters of the P2 module

The number of different states (templates or capacities) used in the experimentation were 3, 5, 10, 20 and 50. The environmental transitions were of two kinds: deterministic, i.e. the probability to change to the next state is always 1 or probabilistic, where, in certain states, the transition can be made to different states. The different states corresponding to the different templates for the dynamic bitmatching problem or to the different capacities of the knapsack problem were generated off-line and randomly selected every time a change in the environment happens. More details can be found in [23].

In the case of deterministic transitions we have a situation as the one illustrated in Figure 6.

[Diagram of deterministic changing environments]

\( K = 3, 5, 10, 20 \) and 50
With probabilistic transitions we may have many different possibilities. In our work we considered Markov chains with 3, 5, 10, 20 and 50 states. The transitions and the corresponding probabilities were defined off-line. The Figure 7 shows an example with 5 states.

![Figure 7](image-url) Probabilistic changing environments with 5 states

### 5.3.5 Types of change period

Three different types of change period were used: periodic, patterned and nonlinear.

**Periodic (linear)**

This type of change period follows a linear trend. If the change period is set to $r$, the environment changes every $r$ generations. We used $r = 10$, $r = 50$, $r = 100$ and $r = 200$.

**Patterned (close to linear)**

In this case, the change period is generated through the repetition of a determined pattern. A pattern is set and the moments of change are calculated based on that pattern. Four different patterns were investigated: 5-10-5, 10-20-10 (fast), 50-60-70 (medium) and 100-150-100 (slow). This way the intervals between changes are not always the same, but the global behavior is close to linear. For instance if the pattern 5-10-5 was used, the first change occurs at generation 5, the second at generation 5 + 10, the third at generation 15 + 5. The first 10 generations of change computed using the four patterns are available in Table 2.

<table>
<thead>
<tr>
<th>Change number</th>
<th>5-10-5</th>
<th>10-20-10</th>
<th>50-60-70</th>
<th>100-150-100</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>10</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>30</td>
<td>110</td>
<td>250</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>40</td>
<td>180</td>
<td>350</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>50</td>
<td>230</td>
<td>450</td>
</tr>
<tr>
<td>5</td>
<td>35</td>
<td>70</td>
<td>290</td>
<td>600</td>
</tr>
<tr>
<td>6</td>
<td>40</td>
<td>80</td>
<td>360</td>
<td>700</td>
</tr>
<tr>
<td>7</td>
<td>45</td>
<td>90</td>
<td>410</td>
<td>800</td>
</tr>
<tr>
<td>8</td>
<td>55</td>
<td>110</td>
<td>470</td>
<td>950</td>
</tr>
<tr>
<td>9</td>
<td>60</td>
<td>120</td>
<td>540</td>
<td>1050</td>
</tr>
<tr>
<td>10</td>
<td>65</td>
<td>130</td>
<td>590</td>
<td>1150</td>
</tr>
</tbody>
</table>

Table 2. The change periods generated through the repetition of a pattern

**Nonlinear**

The two nonlinear change periods were generated using $f_1$ and $f_2$ (eq. 10 and eq. 11) with the following parameters: $\theta_1 = 60$, $\theta_2 = 2$ and $\theta_3 = 0.1$ for $f_1$ and $\theta_1 = 62.67$, $\theta_2 = 0.627$ and $\theta_3 = 0.0047$ for $f_2$. This information is unknown to the prediction module P1, which evolves the values of $\theta_i$ through a GA using only the known data. These two functions allow to model different behaviors for the change period. In the first ($f_1$), the environment changes slower at the beginning and it become faster along time. Using $f_2$ the changes in the environment occur faster. These types of change period will be referred as NL1 and NL2, respectively.

Figure 8 shows the generations where the environment changes using $f_1$ and $f_2$. 
5.3.6 Performance Measures

For each experiment, 30 runs were executed and the number of generations was computed based on 500 environmental changes. The overall performance used to compare the algorithms was the best-of-generation fitness averaged over 30 independent runs, executed with the same random seeds:

\[
overall = \frac{1}{G} \sum_{i=1}^{G} \left[ \frac{1}{R} \sum_{j=1}^{R} F_{best_i,j} \right]
\]  

(12)

where G is the total number of generations and R is the number of runs.

To evaluate the algorithm’s performance along time we used the offline performance. Offline performance is calculated as the average of the best values found so far at each time step [2]. Only the individuals evaluated since the last change are considered:

\[
offline = \frac{1}{G} \sum_{i=1}^{G} e'(t), e' = \max\{e_t, e_{t+1}, \ldots, e_{\tau}\}
\]  

(13)

where G is the total number of generations and \(\tau\) is the last time step before t at which a change in the environment occurred.

Usually, in papers related with the algorithms’ performance on changing environments (e.g. [0, 19, 29]), the measures are saved only after the change is detected and some actions had been taken (as the introduction of information from memory). This way, we don’t know what really happened to the EA’s performance instantly after the change.

In this work, the performance measure is saved immediately after a change is detected. This way we can see if the information introduced before the change, based on given predictions, is really useful to the algorithm’s adaptability.

6 Results

In this section obtained results are analyzed. First we focus how the proposed methods for adjusting \(\Delta\) influence the accuracy of the linear prediction module. Then the algorithm’s performance is analyzed and compared using the different proposed approaches. Finally we present the statistical analysis of the obtained results.
6.1 Prediction Efficacy

The efficacy of prediction was measured using the total number of changes observed and the response of the algorithm to those changes. If the algorithm was able to provide useful information before the next change effectively happens and after the previous change, then we consider that the prediction was accurate, otherwise it is considered inaccurate.

Table 3 shows the accuracy of the P1 using linear regression and the proposed nonlinear regression predictor. The average of the prediction errors and the size of $\Delta$ are also shown.

As we can see in Table 3 the proposed predictor based on nonlinear regression was able to provide accurate predictions in all types of change periods. For the situations where the changes occurred in a nonlinear trend and the linear predictor failed, the nonlinear predictor successfully estimated the next change. As you can see, in the NL1 and NL2 situations the improvements introduced by the nonlinear predictor in the P1 module were noteworthy.

The values reported in Table 3 also show that the prediction error was smaller using the nonlinear-based predictor. This means that the estimated values were equal or better than the values provided by the linear regression predictor. The value of $\Delta$ was also improved. The smaller values for $\Delta$ obtained by the nonlinear regression predictor mean that the algorithm was able to react closer to the change in the environment, saving computational effort.

<table>
<thead>
<tr>
<th>Change period</th>
<th>Algorithm</th>
<th>Act before change</th>
<th>Average of prediction error</th>
<th>Average of prediction error (abs)</th>
<th>Average of $\Delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periodic</td>
<td>PredEA-LR</td>
<td>100.00%</td>
<td>0.00</td>
<td>0.00</td>
<td>5.00</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR</td>
<td>100.00%</td>
<td>0.00</td>
<td>0.00</td>
<td>5.00</td>
</tr>
<tr>
<td></td>
<td>PredEA-LR</td>
<td>97.87%</td>
<td>-0.32</td>
<td>1.01</td>
<td>3.01</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR</td>
<td>100.00%</td>
<td>-0.36</td>
<td>1.01</td>
<td>2.02</td>
</tr>
<tr>
<td></td>
<td>PredEA-LR</td>
<td>99.73%</td>
<td>-0.29</td>
<td>2.37</td>
<td>6.06</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR</td>
<td>99.73%</td>
<td>-0.30</td>
<td>2.36</td>
<td>4.04</td>
</tr>
<tr>
<td></td>
<td>PredEA-LR</td>
<td>99.66%</td>
<td>-0.29</td>
<td>4.43</td>
<td>8.06</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR</td>
<td>99.66%</td>
<td>-0.28</td>
<td>4.38</td>
<td>5.12</td>
</tr>
<tr>
<td></td>
<td>PredEA-LR</td>
<td>96.66%</td>
<td>-0.02</td>
<td>11.53</td>
<td>31.95</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR</td>
<td>100.00%</td>
<td>-0.27</td>
<td>10.77</td>
<td>27.77</td>
</tr>
<tr>
<td></td>
<td>PredEA-LR</td>
<td>0.00%</td>
<td>-1885.52</td>
<td>1885.52</td>
<td>5.00</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR</td>
<td>98.45%</td>
<td>-0.85</td>
<td>0.85</td>
<td>5.00</td>
</tr>
<tr>
<td></td>
<td>PredEA-LR</td>
<td>0.25%</td>
<td>778.56</td>
<td>778.56</td>
<td>592.53</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR</td>
<td>100.00%</td>
<td>-0.39</td>
<td>0.62</td>
<td>2.07</td>
</tr>
</tbody>
</table>

Table 3. Prediction errors for the different types of change periods

6.2 Algorithm’s Performance

Tables 4 and 5 show the global performance of the algorithms solving the DBM and the DKP problems, respectively. The reported results concern the different types of change periods and the different number of environments (states). The best scores are marked with bold.

The results show that the algorithm using the nonlinear predictor (PredEA-NLR) obtained the best results. In the change periods following a nonlinear trend the results were significantly improved when compared with the PredEA-LR algorithm. This is more visible in the NL2 type. In this case, the environmental changes occur faster and the EA without prediction may have some difficulties in readapting after the change. Also, this was the case where the linear regression predictions lead to an untimely use of the information given by the P2 module. In this case, the information was always introduced into the population after the change happens. The new method avoids this error and the PredEA’s performance was improved. These observations were consistent in both studied problems.
### Table 4. Global results for the dynamic bitmatching problem

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Algorithm</th>
<th>Number of states</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>r=10</td>
<td>noPredEA</td>
<td>89.68 90.63</td>
</tr>
<tr>
<td></td>
<td>PredEA-LR</td>
<td>98.14 97.32</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR</td>
<td>98.24 97.37</td>
</tr>
<tr>
<td>r=50</td>
<td>noPredEA</td>
<td>99.01 99.01</td>
</tr>
<tr>
<td></td>
<td>PredEA-LR</td>
<td>99.86 99.85</td>
</tr>
<tr>
<td>r=100</td>
<td>noPredEA</td>
<td>99.50 99.50</td>
</tr>
<tr>
<td></td>
<td>PredEA-LR</td>
<td>99.93 99.92</td>
</tr>
<tr>
<td>r=200</td>
<td>noPredEA</td>
<td>99.73 99.73</td>
</tr>
<tr>
<td>s=5-10</td>
<td>noPredEA</td>
<td>85.56 89.94</td>
</tr>
<tr>
<td></td>
<td>PredEA-LR</td>
<td>96.57 96.45</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR</td>
<td>96.89 96.76</td>
</tr>
<tr>
<td>s=10-20</td>
<td>noPredEA</td>
<td>91.43 94.68</td>
</tr>
<tr>
<td></td>
<td>PredEA-LR</td>
<td>97.79 98.31</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR</td>
<td>98.10 98.62</td>
</tr>
<tr>
<td>s=50-70</td>
<td>noPredEA</td>
<td>99.15 99.15</td>
</tr>
<tr>
<td>s=100-150</td>
<td>noPredEA</td>
<td>99.57 99.57</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR</td>
<td>99.94 99.94</td>
</tr>
<tr>
<td>s=150-200</td>
<td>noPredEA</td>
<td>98.95 98.94</td>
</tr>
<tr>
<td></td>
<td>PredEA-LR</td>
<td>99.19 98.68</td>
</tr>
<tr>
<td>s=200-250</td>
<td>noPredEA</td>
<td>98.11 98.14</td>
</tr>
<tr>
<td></td>
<td>PredEA-LR</td>
<td>98.12 98.15</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR</td>
<td>99.80 99.79</td>
</tr>
</tbody>
</table>
### Table 5. Global results for the dynamic knapsack problem

Figures 9 to 18 show the evolution of the algorithms’ performance along time. The plots show the off-line performance obtained by the EA with and without prediction solving the dynamic bitmatching problem. Figures 19 to 28 refer to the knapsack problem.

We can see that the PredEA using the new method based on nonlinear regression (PredEA-NLR) was able to improve the algorithm's performance. By the reasons stated before, this enhancement was more considerable in the NLR2 change period. All the results support that the proposed predictor is efficient and robust and improves the algorithm's performance. It is capable of accurately predict when next change will occur in environments changing periodically, following a repeated pattern or a nonlinear function. This way the anticipation of the change and the introduction of useful individuals in the population allows the continuous adaptation of the EA.
Figure 9. Comparing offline performance for the bitmatching problem: 10 states, periodic $r = 10$

Figure 10. Comparing offline performance for the bitmatching problem: 10 states, periodic $r = 100$

Figure 11. Comparing offline performance for the bitmatching problem: 10 states, pattern 10-20-10
Figure 12. Comparing offline performance for the bitmatching problem: 10 states, pattern 100-150-100

Figure 13. Comparing offline performance for the bitmatching problem: 10 states, NL2 change period

Figure 14. Comparing offline performance for the bitmatching problem: 50 states, periodic $r = 10$
Figure 15. Comparing offline performance for the knapsack problem: 50 states, periodic $r = 100$

Figure 16. Comparing offline performance for the knapsack problem: 50 states, pattern 10-20-10

Figure 17. Comparing offline performance for the knapsack problem: 50 states, pattern 100-150-100
Figure 18. Comparing offline performance for the knapsack problem: 50 states, NL2 change period

Figure 19. Comparing offline performance for the knapsack problem: 10 states, periodic \( r = 10 \)

Figure 20. Comparing offline performance for the knapsack problem: 10 states, periodic \( r = 100 \)
Figure 21. Comparing offline performance for the knapsack problem: 10 states, pattern 10-20-10

Figure 22. Comparing offline performance for the knapsack problem: 10 states, pattern 100-150-100

Figure 23. Comparing offline performance for the knapsack problem: 10 states, NL2 change period
Figure 24. Comparing offline performance for the knapsack problem: 50 states, periodic $r = 10$

Figure 25. Comparing offline performance for the knapsack problem: 50 states, periodic $r = 100$

Figure 26. Comparing offline performance for the knapsack problem: 50 states, pattern 10-20-10
Figure 27. Comparing offline performance for the knapsack problem: 50 states, pattern 100-150-100

Figure 28. Comparing offline performance for the knapsack problem: 50 states, NL.2 change period

6.3 Statistical Analysis

The major statistical results of comparing the different methods are in Tables 6 and 7. We used paired one-tailed t-test at a 0.01 level of significance [13]. The notations used in Tables 6 and 7 to compare each pair of algorithms is "−", “+”, “++” or “−−”, when the first algorithm is better than, worse than, significantly better than, or significantly worse than the second algorithm. Table 6 refers to the t-test results for the dynamic bitmatching problem and Table 7 shows the statistical results for the dynamic knapsack problem.

In both problems the statistical results corroborate that the proposed predictor based on nonlinear regression introduced significant improvements in the PredEA’s performance. Tables 6 and 7 show that, for the nonlinear change periods, the new predictor was always significantly better than the linear regression predictor, which failed on those situations. In the remaining cases the proposed method was also accurate, improving the results in some situations. Those improvements were not so noteworthy because, in the periodic or patterned change periods, choosing correctly the value of Δ, the predictions provided by linear regression module were already precise.
### Bitmatching Problem

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Algorithm</th>
<th>Number of states</th>
<th>3</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>D</td>
<td>P</td>
<td>D</td>
<td>P</td>
<td>D</td>
<td>P</td>
</tr>
<tr>
<td>r=10</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>r=50</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>r=100</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>r=200</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>5-10-5</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>10-20-10</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>50-60-70</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>100-150</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>NL 1</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>NL 2</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>

Table 6. Statistical results for the bitmatching problem

### Knapsack Problem

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Algorithm</th>
<th>Number of states</th>
<th>3</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>D</td>
<td>P</td>
<td>D</td>
<td>P</td>
<td>D</td>
<td>P</td>
</tr>
<tr>
<td>r=10</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>r=50</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>r=100</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>r=200</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>5-10-5</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>10-20-10</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>50-60-70</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>100-150</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>NL 1</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>NL 2</td>
<td>PredEA-NLR -- noPredEA</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>PredEA-NLR -- PredEA-LR</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>

Table 7. Statistical results for the knapsack problem
7 Conclusions and Future Work

In this paper we propose a new, precise and robust prediction method to use in EA dealing with dynamic environments. This predictor, which uses nonlinear regression, is responsible to estimate when the next change in the environment will happen. In previous work this task was made by a predictor based on linear regression which was limitative in situations where the change period was not periodic or changes follow a nonlinear pattern. The proposed method overcomes these limitations and proved to be robust and efficient, allowing making accurate predictions in a wider range of situations. Analyzing the obtained results, some conclusions can be stated.

First, the nonlinear regression predictor was able to accurately predict when next change will take place in all types of change periods studied. Second, the precision of the predictions was improved, especially in the situations where the linear regression predictor failed. Third, better predictions lead to a better adjustment of the value of $\delta$ avoiding unnecessary and expensive computational efforts.

The major limitation of the proposed algorithm is the number of functions used in the predictor module $SP15$. The efficacy of the EA can be compromised if the environment changes following a pattern that cannot be modeled by any of the existing functions. Some improvements are being introduced and tested to overcome this limitation, namely the incorporation of more functions and the use of a GP to evolve the function that best fits the known data.

8 Acknowledgments

The work of the first author described in this report was partially financed by the PhD Grant BD/39293/2006 of the Foundation for Science and Technology of the Portuguese Ministry of Science and Technology and High Education.

9 References
