Evolutionary Algorithms for Dynamic Environments: Prediction using Linear Regression and Markov Chains

Anabela Simões[1,2] Ernesto Costa[2]

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

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

CISUC TECHNICAL REPORT TR 2008/01 - ISSN 0874-338X MARCH 2008

Abstract. In this work we investigate the use of prediction mechanisms in Evolutionary Algorithms for dynamic environments. These mechanisms, 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. Our study is made using environments where some pattern can be observed. Environments can change in two ways: periodically, following a fixed change period, or according to a repeated pattern of change. Markov Chains are applied to model the characteristics of the environment and we use that model to predict about possible future states. To forecast when a change will probably arise, we employ a linear regression predictor. Both predictions modules use information from the past to estimate the future. Knowing *à priori* when a change will take place and which state(s) will appear next, we can introduce useful information in the population *before* change happens, avoiding the performance's decrease usually observed with standard evolutionary algorithms. The techniques are applied to several instances of the dynamic bit matching problem and the obtained results prove the effectiveness of the proposed mechanisms.

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

1 Introduction

Evolutionary algorithms (EAs) have been applied successfully to a great variety of stationary optimization problems. However, most real-world applications change over time and some modifications have been introduced in EAs in order to deal with this kind of problems: the use of memory [1, 10, 13, 14, 20, 21], the maintenance of population's diversity [8, 9, 13] or the use of several populations [3,19].

When the environment is dynamic, in some cases, a certain repeated functioning can be observed. For instance, the environment can change in a cyclic manner or repeating a certain pattern. In environments with these characteristics, we can try to predict the *moment* and the *pattern* of the change. Predicting modifications allows anticipating the sudden decrease in performance of an evolutionary algorithm and improve its adaptability.

The idea of anticipating the change in dynamic environments has already been explored. A brief description of these works will be done in next section. In our case, our method involves the use of a **memory** of good past individuals, besides the normal population. That memory interplays with two other modules: one based on linear regression, and the other supported by a Markov chain. Linear regression is used to estimate ***when*** the next change in the environment will happen; Markov chains are used to model what is known about all possible environments and the transitions among them. The Markov chain is used to predict ***which*** new environments will most probably appear in the future.

The output of the linear regression module is based on the time of past changes. Once that moment is defined we use the Markov chain model to predict how the new possible environments will look like. **Before** the predicted moment of change we seek from memory good individuals for these new situations and inject them in the normal population.

The main goal of this paper is to investigate the effectiveness of using predictors based on linear regression and Markov chains. 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. Section 3 presents the basics of Markov. Prediction based on linear regression method is explained in section 4. In section 5 we explain the overall architecture of an EA that utilizes the Markov chain prediction and the linear regression module. In section 6 we present the experimental setup used to test the proposed ideas. Experimental results are summarized in section 7. 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 behaviour in the present. Since information about the future typically is not available, it is attained through learning from past situations.

Branke et al. [2] 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 [17] 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. [18] 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 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 Ŏsmera's function. Results showed that future prediction was useful in the Ŏsmera'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 [4, 5, 6, 7] proposed in the last years several approaches focused on the importance of using learning and anticipation in online dynamic optimization. These works analyse 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.

The use of linear regression to predict the moment of next change was initially proposed by Simões and Costa [16]. The idea was tested with different dynamic optimization problems, using a variable-size memory EA. Several issues were analysed 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.

3 Markov Chains

There are random processes that do not keep memory of the whole past. They are called Markov processes. If a process can assume only a finite or countable set of states we talk of Markov chains. 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, ... P is a matrix representing transition probabilities between states and λ is the initial probability distribution for all the states in S.

Markov chains are memoryless, meaning that the present state is enough to predict future states, i.e.:

$$Pr(X_{n+1}= x \mid X_n=x_n, \ldots, X_1=x_1) = Pr(X_{n+1} = x \mid X_n=x_n)$$

A Markov chain is fully specified by [12]:

- The probability of initial states: $\lambda=\{p_0, p_1, p_2, \ldots\}$ with p_i, the probability of starting at state *i*.
- The transition probabilities matrix: $P = \begin{bmatrix} p_{00} & p_{01} & p_{02} & \cdots & p_{0n} \\ p_{10} & p_{11} & p_{12} & \cdots & p_{1n} \\ p_{20} & p_{21} & p_{22} & \cdots & p_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ p_{m0} & p_{m1} & \cdots & \cdots & p_{mn} \end{bmatrix}$

where p_{ij} is the probability from going from state *i* (line) to state *j* (row).

4 Linear Regression

Simple linear regression studies 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 about 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*. The equation of that line is:

$$\mu_y = \beta_0 + \beta_1 * x \quad (1)$$

with intercept β_0 and slope β_1. This is the linear regression line and describes how the mean response changes with *x*. The observed *y* values will vary around the mean and it's assumed that this variation, measured by the standard deviation, is the same for all the values of *x* [11].

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. Three inferences are possible: (1) estimate the slope β_1 and the intercept β_0 of the regression line; (2) estimate the mean response μ_y, for a given value of x; (3) predict a future response *y* for a given value of *x*.

In general, the goal of linear regression is to find the line that best predicts *y* from *x*. Linear regression does this by finding the line that minimizes the sum of the squares of the vertical distances of the points from the line.

The estimated values for β_0 and β_1 called b_0 and b_1 are obtained using previous observations through equations (2) and (3). The intercept b_0 is given by:

$$b_0 = \bar{y} - b_1 * \bar{x} \quad (2)$$

The slope b_1 is given by:

$$b_1 = cr * \frac{s_y}{s_x} \quad (3)$$

Where $\bar{y}$ is the mean of the observed values of y, $\bar{x}$ is the mean of the observed values of x, *cr* the correlation between *x* and *y* given by equation (4), s_x and s_y the standard deviations of the observed *x* and *y*, respectively, given by equation (5).

$$cr = \frac{1}{n-1}\sum\left(\frac{x_i - \bar{x}}{s_x}\right)\left(\frac{y_i - \bar{y}}{s_y}\right) \tag{4}$$

$$s_x = \sqrt{\frac{1}{n-1}\sum(x_i - \bar{x})^2} \qquad s_y = \sqrt{\frac{1}{n-1}\sum(y_i - \bar{y})^2} \tag{5}$$

with *n*, the number of previous observations from *x* and *y*.

Once estimated the slope and intercept of the regression line, we can predict the value of *y* (called by $\hat{y}$) from a given *x* using the regression line equation:

$$\hat{y} = b_0 + b_1 * x \tag{6}$$

Notice that linear regression does not test if the data are linear. It assumes that the data are linear, and finds the slope and intercept that make a straight line best fit the known data. If the error in measuring *x* is large, other inference methods are needed.

5 System Overview

In this section we will detail each component of the proposed system, called PredEA. Figure 1 shows the overview of the implemented architecture using the Markov chain and the linear regression modules to enhance the EA with prediction capabilities. The major components of the complete architecture are the following:

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

(b) Memory: the memorized individuals are associated with the state index where they were the best solution. It is updated from time to time with the current best individual of population.

(c) *Markov chain module*: consists in set of states (templates), a matrix of state transition probabilities and the initial probability vector. 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 states appear.

(d) *Linear regression module*: based on previous information, this component is used to predict when next change may occur.

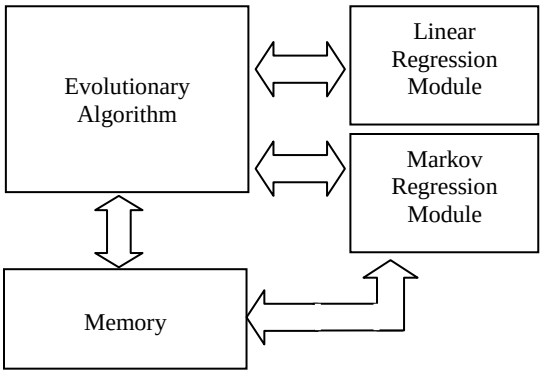


Figure 1. Computational platform of the ***PredEA***

The dynamics of the environment is defined off-line: the number of different states, the possible environments, the sequence of environments to use during the simulation and the initial state. We emphasize that all this information is **unknown** by the EA and that the Markov model is constructed during the simulation.

As the predictions made by the linear regression module may not be exact, we use a parameter, called Δ, to control the maximum estimated error, measured in terms of generations before the actual occurrence of the change. More explicitly, if the predicted value returned by the linear regression model is generation g, at generation $g - \Delta$, the Mrakov model is issued to predict possible future states. At that time, individuals from the memory are retrieved and introduced in the population, replacing the worst ones. The selected memory individuals are those who were good solutions in the state(s) that are considered to be the next possible ones by the Markov chain model.

Every time a change actually happens, the probabilities of the transition matrix of the Markov chain are updated accordingly. This includes the case when a new state appears. In that case, the new state is included in the model and, again, the transition matrix is updated. Notice that, in the earlier stages of the simulation, prediction is difficult, since the algorithm needs to experience a learning phase to set up the values of the transition matrix.

As we will see, the anticipation based on the introduction of useful information from memory, avoids the decrease of the algorithm's performance.

Each component will be explained in detail in next sections and the pseudocode can be found in section 5.5.

5.1 Evolutionary Algorithm

It's a standard memory-based EA. One population of individuals evolve by means of selection, crossover and mutation and is used to find the best solution for the current environment. The memory population is used 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 readapt to the new environment.

5.2 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 [20]: $T_M(0)$=rand(5,10) and $T_M(t) = T_M(t-1)$ + 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 previosly 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 [1]. 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 [15] and proved to be very efficient in memory-based EAs for changing environments.

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.

5.3 Markov Chain Module

In our approach, each state of the Markov chain corresponds to a template that represents the global optimum for a certain environment. Initially, the maximum number of different states is defined as well as the possible sequence of states that may occur during the algorithmic process. 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.

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.) This sequence has a corresponding succession of binary templates that the EA must optimize (Figure 2, right).

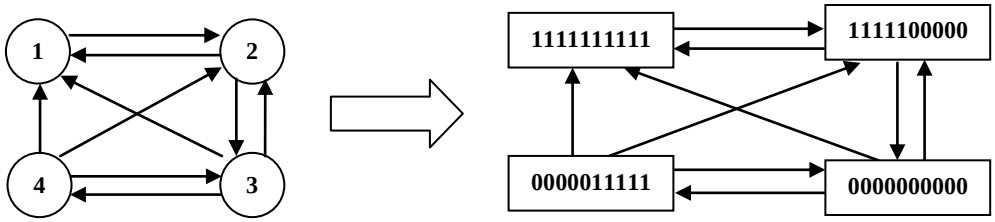


Figure 2. An example of environmental changes

The transition matrix starts with all positions filled with zero, and when a transition 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 thee introduction of important information before change, allowing the continuous adaptation of the EA to the new conditions.

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

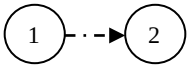
$$\lambda = \{1.0, 0.0, 0.0, 0.0\} \qquad P = \begin{pmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.5 & 0.0 \\ 0.3 & 0.4 & 0.0 & 0.3 \\ 0.2 & 0.2 & 0.6 & 0.0 \end{pmatrix}$$

The algorithm builds the following model step by step:

Step 1: initial state (defined in vector **λ**)

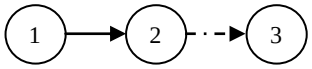
$$P'_{(0)} = \begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

Step 2: go to state 2 (according to matrix P, probability p_{12})



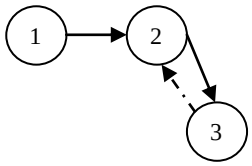
$$P'_{(1)} = \begin{pmatrix} 0.0 & \mathbf{1.0} & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

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



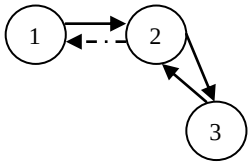
$$P'_{(2)} = \begin{pmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & \mathbf{1.0} & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

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



$$P_{(3)} = \begin{pmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & \mathbf{1.0} & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

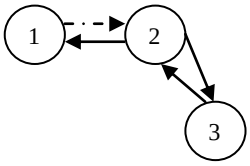
Step 5: probabilistically chooses to go to state 3 or 1 (1 was assumed as the chosen state)
At this step, since state 2 already appeared in the past, the Markov chain module gives as prediction a possible transition to state 3 (known information until the present moment)



$$P'_{(4)} = \begin{pmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ \mathbf{0.5} & 0.0 & \mathbf{0.5} & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

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)



$$P'_{(5)} = \begin{pmatrix} 0.0 & \mathbf{1.0} & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

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

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

5.4 Linear Regression Module

Knowing the best moment to start using the predicted information provided by the Markov chain module can improve the adaptation's capabilities of our EA. This moment is computed by calling the Linear Regression Module. The method is simple: 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 new values of b_0 and b_1 are computed and, using Equation 2, the regression line is updated.

We will 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.

Example

For example, suppose that three observations were already made (n = 3):

Obs	x	y
1	1	50
2	2	100
3	3	150

The first change at generation 50, the second change at generation 100 and the third change at generation 150. Can we predict when will occur the fourth change?

Using equations (2) and (3), the estimated values for the slope and intercept of the regression line are:

$$b_1 = 1 * \frac{50}{1} = 50 \text{ and } b_0 = 100 - b_1 * 2 = 100 - 50 * 2 = 0$$

Using equation (6) we can predict when the fourth change will occur (x = 4):

$$next_{gen} = \hat{y} = b_0 + b_1 * x = 0 + 50 * 4 = 200$$

In this case, if the change is determined with a fixed change period of size r = 50, the prediction is exact and there is no related error.

This component was incorporated in the EA and, every time a change happens, the moment when next change will probably occur is predicted. This prediction is made based on previous changes and uses that information to estimate the slope β_1 and the intercept β_0 of the regression line. Based on this line, the generation where next change will occur is estimated.

5.5 *PredEA* Pseudocode

Now that we have described the different components we can present the pseudocode of ***PredEA***. The global system can be described as follows:

```
PredEA(max, markov, initial-state)
1.  Randomly create initial population
2.  Create empty memory
3.  Create the transition matrix with max sates filled with zeros
4.  repeat
5.      Evaluate population
6.      Evaluate memory
7.      if is time to update memory then
8.          Store best individual
9.          Set next time to update memory
10.     if an environmental change happens then
11.         Store performance measures
            %Prediction modules
12.         Update linear regression line
13.         Predict g (next_change) (Linear Regression Module)
14.         Update the algorithm's Markov transition matrix
15.     if g (next_change) is close (as defined by g - Δ) then
16.         Predict next state(s) (using EA's Markov model)
17.         Search memory for best individual(s) for that(ese) state(s)
18.         Introduce the selected individual(s) into population
        %Standard EA steps
19.     Perform Selection, crossover and mutation
20.     Define next population
21. until stop_condition
```

Figure 3. Pseudocode for the ***PredEA***

max is the maximum number of states of the Markov chain, *markov* is the Markov model defined off-line and *initial-state* is the randomly chosen initial state for the Markov model. The action described in 12 will be explained in section 6 (Equation 7).

6 Experimental Design

Experiments were carried out to compare the ***PredEA*** with a similar algorithm without prediction capabilities (we will refer this second algorithm as ***noPredEA***). The latter algorithm is an EA with the direct memory scheme proposed by [21]. Memory is updated using the same time method, but instead of the similar replacing strategy used by [21], a generational scheme is used instead. After a change is detected, population and memory are merged and the best **N - M** individuals are selected as a temporary population to go through crossover and mutation, while the memory remains unaffected (**N** is the size of population, **M** is the size of memory).

6.1 Dynamic Bit Matching Problem

The benchmark used to test the investigated ideas was the well-known dynamic bit matching problem: 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.

6.2 Experimental Setup

6.2.1 EA's parameters

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 mutation applied with probability p_m=0.01. Binary representation was used with chromosomes of size 100 (size of the binary templates). Population of 80 individuals and a memory of 20 individuals were used. The value of the Δ constant referred above was 5 generations. Other experiments were made changing this parameter Δ, and some preliminary results are shown in section 7.

Table 1 summarizes the EA's settings:

EA parameters	**value**
Individual's representation	binary
initialization	uniform randomly created
population size	80
memory size	20
crossover	uniform, probability 70%
mutation	flip, probability 1%
parent's selection	tournament, size 2
survivors' selection	generational with elitism of size one
stop criterion	the number of generations necessary for 200 environmental changes
goal	maximize matching with template
Δ	{5, 10, 25}

Table 1. Evolutionary Algorithm's parameters' settings

6.2.2 Performance Measure

For each experiment, 30 runs were executed and the number of generations was computed based in 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:

$$Foverall = \frac{1}{G}\sum_{i=1}^{G}\left[\frac{1}{R}\sum_{j=1}^{R}F_{bestij}\right] \qquad (7)$$

G=number of generations, R=number of runs.

Usually, in papers related with the algorithms' performance on changing environments (e.g. [1, 13, 21]), 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.

6.2.3 Number and transition between states

The number of different states (templates) 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 (this kind of dynamics will be denoted by P_{ij}=1) or *probabilistic*, where, in certain states, the transition can be made to different states (this kind of dynamics will be called by P_{ij}<>1). See section 6.2.5 for more details.

In the case of deterministic transitions we have a situation as the one illustrated in figure 4.

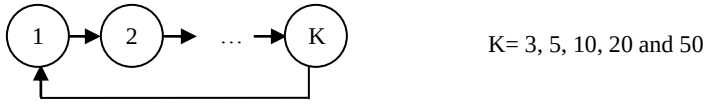


Figure 4. Deterministic changing environments

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 5 shows an example with 5 states.

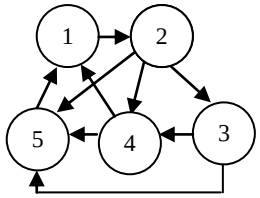


Figure 5. Probabilistic changing environments with 5 states

6.2.4 When does the environment change?

The period was changed in two different ways: periodically, every **r** generations or according to a fixed pattern.

In periodic environments the parameter ***r*** was used with four different values: 10, 50, 100 and 200 generations between changes. This type of changes will be denoted as *cyclic-periodic* environments.

In the second case, a pattern was set and the moments of change were 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 follow some pattern making prediction possible.

In the first case (pattern 5-10-5), changes will occur at generations 5, 15, 20, 25, 35, 40, 45, 55, 60, .., i.e, the pattern provides the gap between the changes. This type of change will be referred as *cyclic-pattern* environments. This means that 80 different situations were analyzed. Table 2 summarizes the different scenarios tested in this work.

Type of change	Change period	Number of states	Type of states dynamics	
Cyclic-periodic	10	3	Deterministic (P_{ij}=1)	Probabilistic (P_{ij}<>1)
		5		
		10		
		20		
		50		
	50	3	Deterministic(P_{ij}=1)	Probabilistic (P_{ij}<>1)
		5		
		10		
		20		
		50		
	100	3	Deterministic(P_{ij}=1)	Probabilistic (P_{ij}<>1)
		5		
		10		
		20		
		50		
	200	3	Deterministic(P_{ij}=1)	Probabilistic (P_{ij}<>1)
		5		
		10		
		20		
		50		
Cyclic-pattern	5-10-5	3	Deterministic(P_{ij}=1)	Probabilistic (P_{ij}<>1)
		5		
		10		
		20		
		50		
	10-20-10	3	Deterministic(P_{ij}=1)	Probabilistic (P_{ij}<>1)
		5		
		10		
		20		
		50		
	50-60-70	3	Deterministic(P_{ij}=1)	Probabilistic (P_{ij}<>1)
		5		
		10		
		20		
		50		
	100-150-100	3	Deterministic(P_{ij}=1)	Probabilistic (P_{ij}<>1)
		5		
		10		
		20		
		50		

Table 2. Different environmental dynamics tested

7 Results

In this section obtained results are analysed. First we show the accuracy of the linear regression predictor and then the ***PredEA***'s performance is compared with a similar EA without the prediction capabilities (***noPredEA***). Statistical information will also be provided.

7.1 Accuracy of Predicted Values using Linear Regression

When changes are defined by the parameter **r** and they occur every **r** generations, the statistical model using linear regression gives correct predictions, since all observed values are on the regression line. Figure 6 shows the predicted values using periodic changes of size 10 and 50 (**r=10**, **r = 50**).

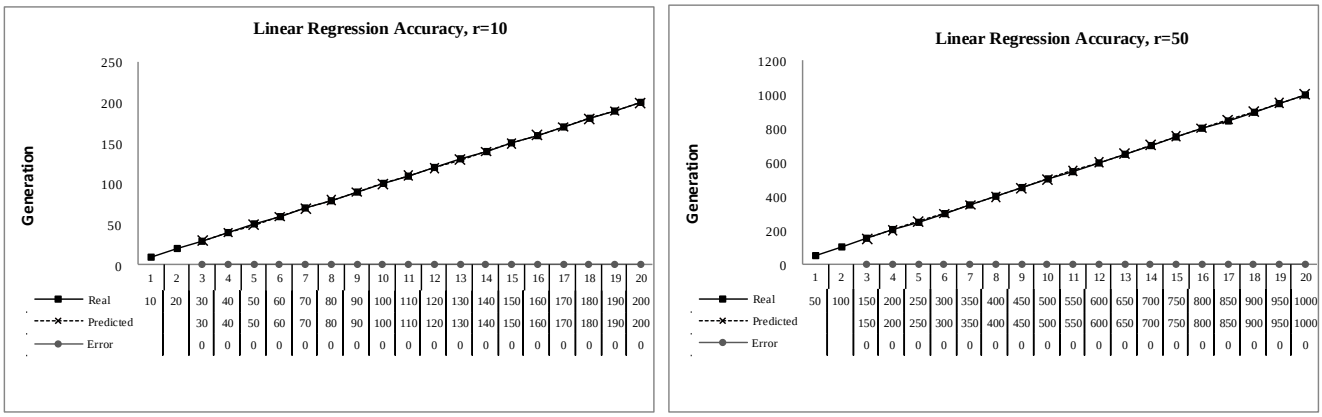


Figure 6. Prediction accuracy in periodic environments with a fixed period (r = 10 and r = 50)

Using a pattern to generate the periodicity of the change, we may have situations where the predicted values are not precise. In these cases, there is an associated error that decreases over time. Nevertheless this decreasing is very slow. For the 20 first changes observed using the patterns 5-10-5 and 10-20-10 and plotted in Figures 7 and 8, we can see that the prediction model was not exact. In this case, the Δ constant assumed in our implementation (5 generations), in general, was enough to cover that error, and in this case, the insertion of individuals in the population was always made *before* the change occurs.

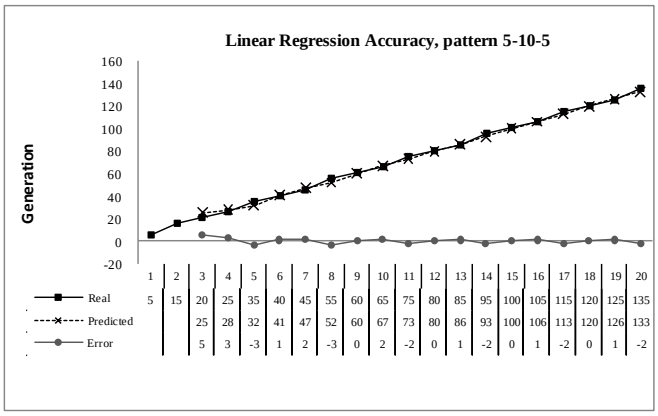


Figure 7. Prediction accuracy in periodic environments with a pattern change (5-10-5)

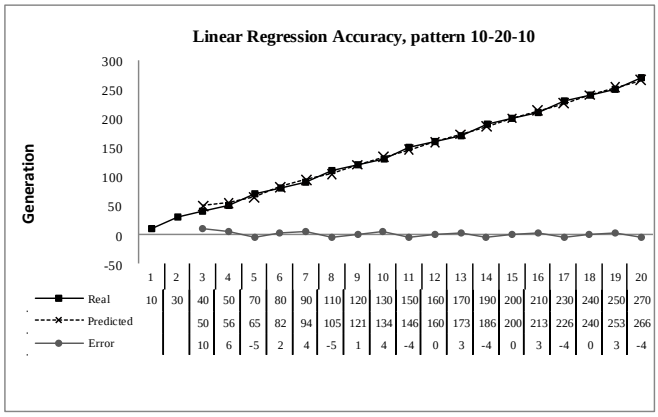


Figure 8. Prediction accuracy n periodic environments with a pattern change (10-20-10)

In the case of pattern 50-60-70, in some cases, the *Δ* constant of 5 wasn't enough to ensure that some action could be taken before change. As we will see in next section, this fact leads to a poor performance of the algorithm. Figure 9 shows the predicted values using this pattern's change.

Using a different pattern, 100-150-100 the used prediction model had worse predicted values. The errors associated to the estimated values were in most cases superior to 5, and our module using an interval margin of 5 generations didn't deal with these changes before they happen. Figure 10 shows the expected values and the resultant error. Once again we observe a decrease of **PredEA**'s performance.

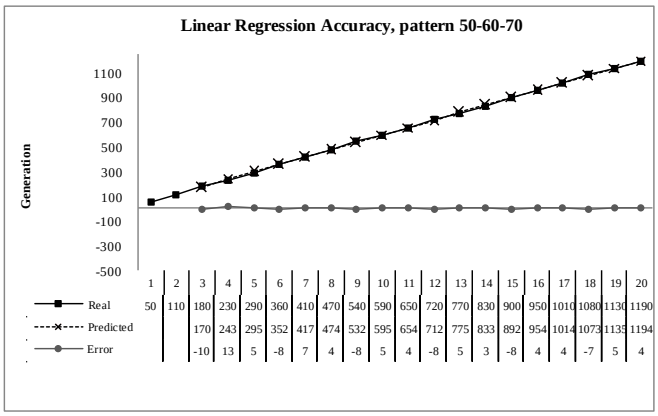


Figure 9. Prediction accuracy in periodic environments with a pattern change (50 – 60 – 70)

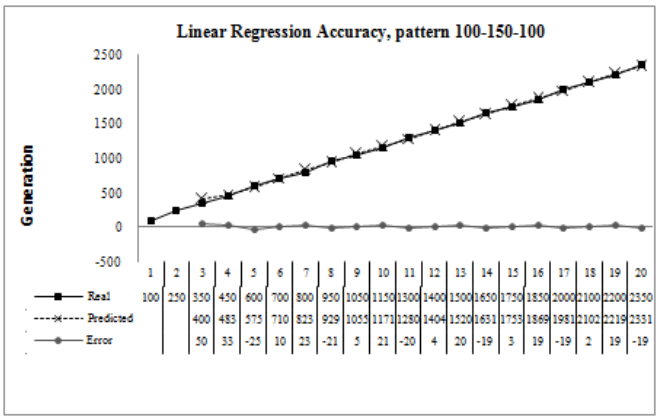


Figure 10. Prediction accuracy in periodic environments with a pattern change (100 – 150 – 100)

7.2 PredEA *versus* noPredEA

7.2.1 Results on cyclic-periodic environments

Results obtained for *cyclic-periodic* environments (changing every ***r*** generations) are given in table 3. The best scores are marked in bold.

The statistical results of comparing the two EAs are given in table 4. We used a paired one-tailed t-test at a 0.01 level of significance. The notation used in tables 4 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.

		Number of states				
r	Algorithm	3	5	10	20	50
10	Pred-EA (Pij = 1)	**98.24**	**97.92**	**97.87**	**97.33**	**94.42**
	Pred-EA (Pij<>1)	**98.10**	**97.78**	**97.25**	**96.55**	**93.55**
	NoPred-EA (Pij = 1)	89.41	84.90	80.04	74.87	69.69
	NoPred-EA (Pij<> 1)	89.64	85.41	80.58	75.40	70.38
50	Pred-EA (Pij = 1)	**99.39**	**99.04**	**98.08**	**96.46**	**91.31**
	Pred-EA (Pij<>1)	**98.90**	**98.39**	**98.69**	**96.45**	**90.19**
	NoPred-EA (Pij = 1)	98.72	98.39	97.66	95.40	88.29
	NoPred-EA (Pij<> 1)	98.71	98.39	97.65	95.76	89.28
100	Pred-EA (Pij = 1)	**99.69**	**99.51**	**99.01**	**98.55**	**95.48**
	Pred-EA (Pij<>1)	**99.43**	**99.67**	**99.29**	**99.14**	**95.10**
	NoPred-EA (Pij = 1)	99.38	99.24	98.90	98.29	94.11
	NoPred-EA (Pij<> 1)	99.37	99.24	98.91	98.29	94.70
200	Pred-EA (Pij = 1)	**99.84**	**99.75**	**99.50**	**99.37**	**97.74**
	Pred-EA (Pij<>1)	**99.72**	**99.79**	**99.64**	**99.56**	**97.75**
	NoPred-EA (Pij = 1)	99.69	99.62	99.45	99.15	97.04
	NoPred-EA (Pij<> 1)	99.69	99.62	99.46	99.15	97.34

Table 3. Global results on cyclic-periodic environments, with change period ***r***

T-test results	*r*	Nº of states				
		3	5	10	20	50
PredEA -- noPredEA (P_{ij}= 1)	10	++	++	++	++	++
PredEA -- noPredEA (P_{ij}<> 1)		++	++	++	++	++
PredEA -- noPredEA (P_{ij} = 1)	50	++	++	++	++	++
PredEA -- noPredEA (P_{ij}<> 1)		++	+	++	++	++
PredEA -- noPredEA (P_{ij} = 1)	100	++	++	++	++	++
PredEA -- noPredEA (P_{ij}<> 1)		++	++	++	++	++
PredEA -- noPredEA (P_{ij} = 1)	200	++	++	++	++	++
PredEA -- noPredEA (P_{ij}<> 1)		++	++	++	++	++

Table 4. Statistical significance of comparing ***PredEA*** with ***noPredEA*** (ttest) in cyclic-periodic environments (r=10,50, 100 and 200)

The results obtained with ***PredEA*** were always significant better than the ***noPredEA***. Using prediction to insert information **before** change happens actually improves the EA's performance. In rapidly changing environments (r = 10), the improvements introduced with the anticipation of change are clearly positive. Besides, as the number of different states increases, the ***noPredEA***'s performance decreases faster than the ***PredEA***'s. Using 50 states the results were inferior since, in some cases, the algorithm has not enough time to complete the "learning phase". In these cases, more time of evolution is necessary. Figure 11 plots the achieved results.

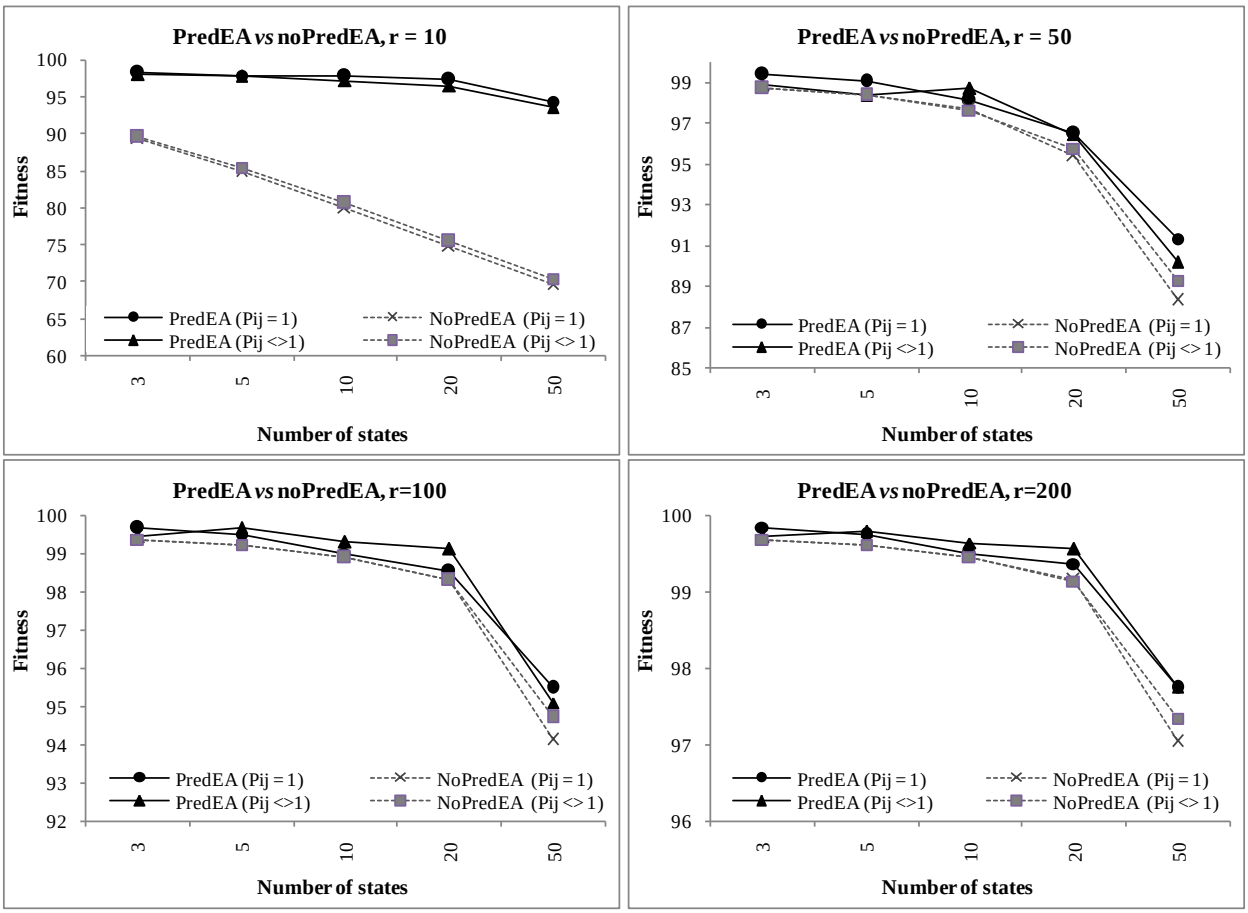


Figure 11. Global results for ***PredEA*** and ***noPredEA*** in cyclic-periodic environments (r = 10, 50, 100 and 200)

In figures 12 to 15 we can see the behaviour of the algorithms in the first 5000 generations, using 10 different states with *cyclic* (P_{ij}=1) dynamics. ***PredEA*** has a starting phase where the performance is very instable and abruptly decreases every time a change is detected. This is the "learning phase" where the algorithm builds the Markov chain model and its transition matrix. After that, and since repeated environments reappear, the predictions are correctly made by the two predictor modules and the ***PredEA's*** performance reaches an "equilibrium phase". Using r=10, this equilibrium is not reached in the

first 5000 generations, since the gap between changes is small and the algorithm can't find the global optimum. In this case, the equilibrium phase is completely attained after a larger number of generations.
The behaviour of ***noPredEA*** is very unstable. After a change, we observe a decrease on its fitness and only after retrieving information from memory, which is made immediately after a change happens, the EA recovers.

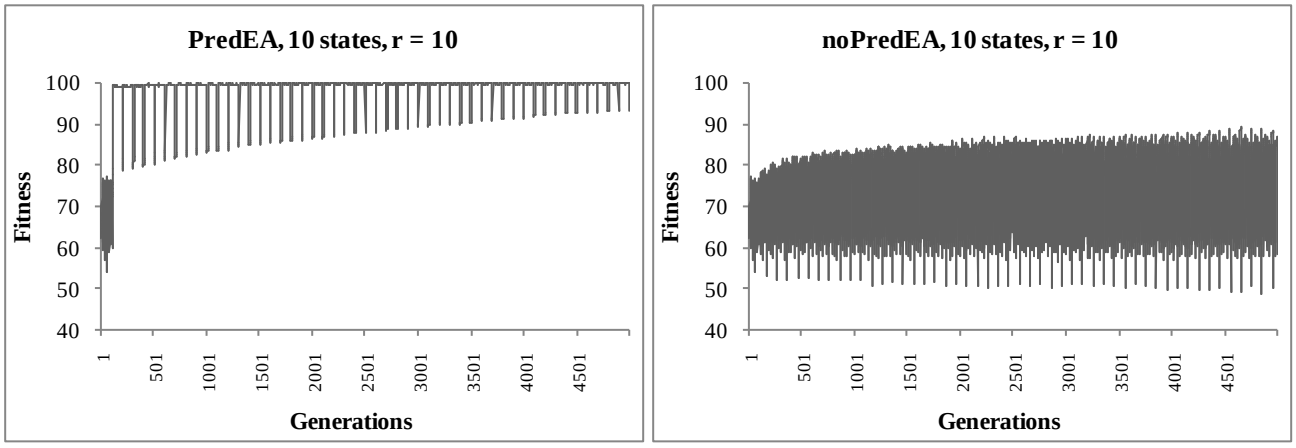


Figure 12. ***PredEA*** versus ***noPredEA***, change period r = 10, 10 states, dynamics of type P_{ij}=1

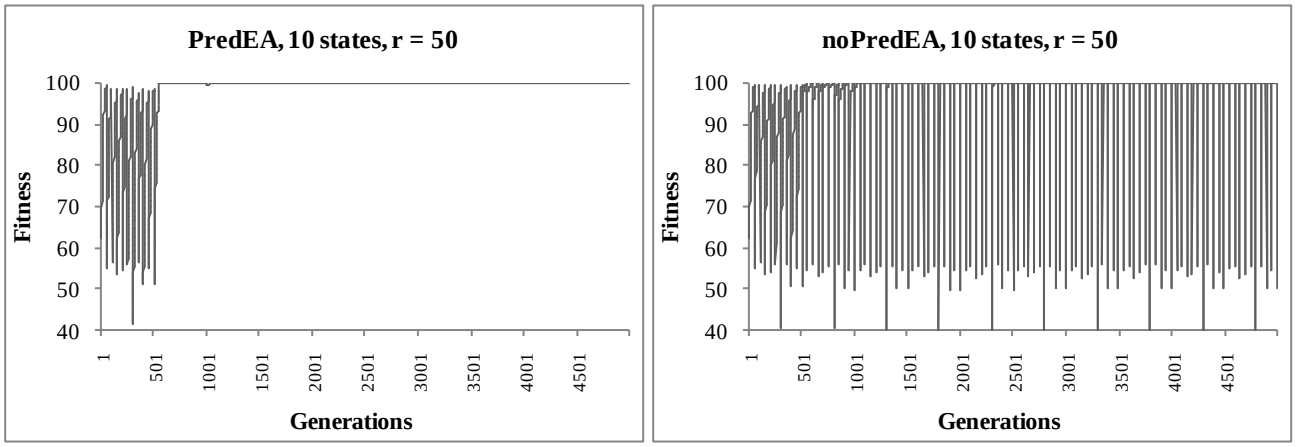


Figure 13. ***PredEA*** versus ***noPredEA***, change period r = 50, 10 states, dynamics of type P_{ij}=1

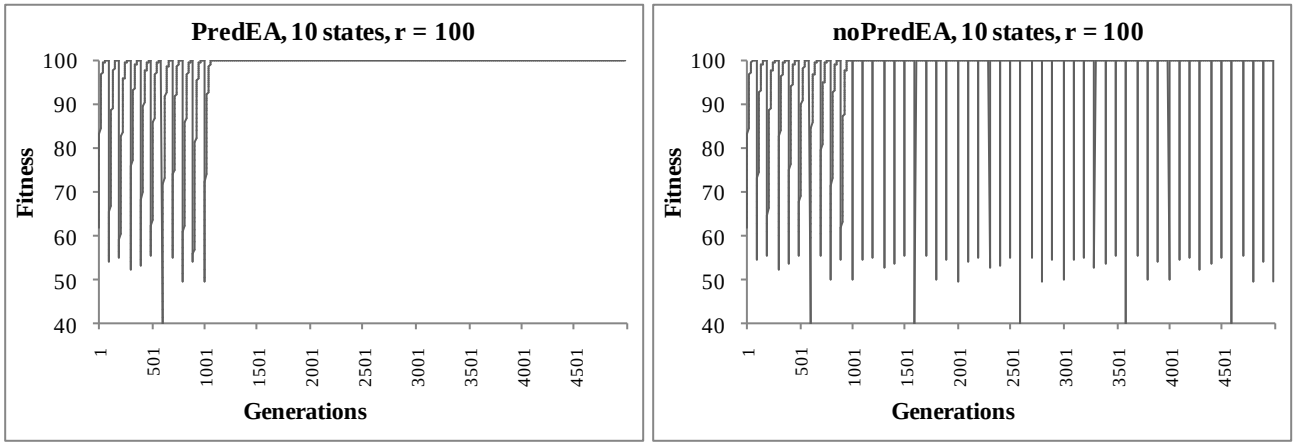


Figure 14. ***PredEA*** versus ***noPredEA***, change period r = 100, 10 states, dynamics of type P_{ij}=1

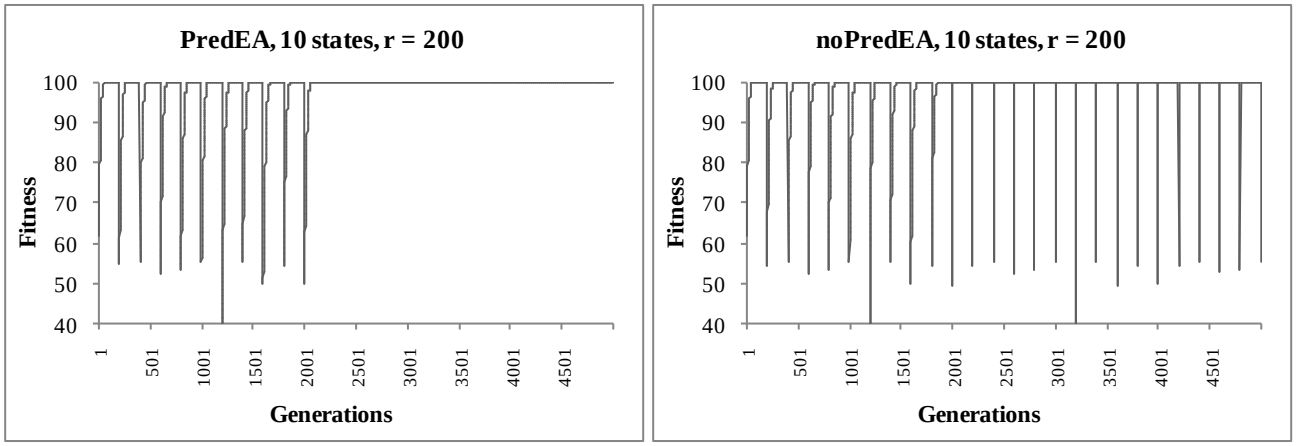


Figure 15. ***PredEA*** versus ***noPredEA***, change period r = 200, 10 states, dynamics of type P_{ij}=1

Following, we show similar results for probabilistic changing dynamics. In this casa, as the EA needs more time to capture all information about environment, we show complete results, concerning the number of generations.

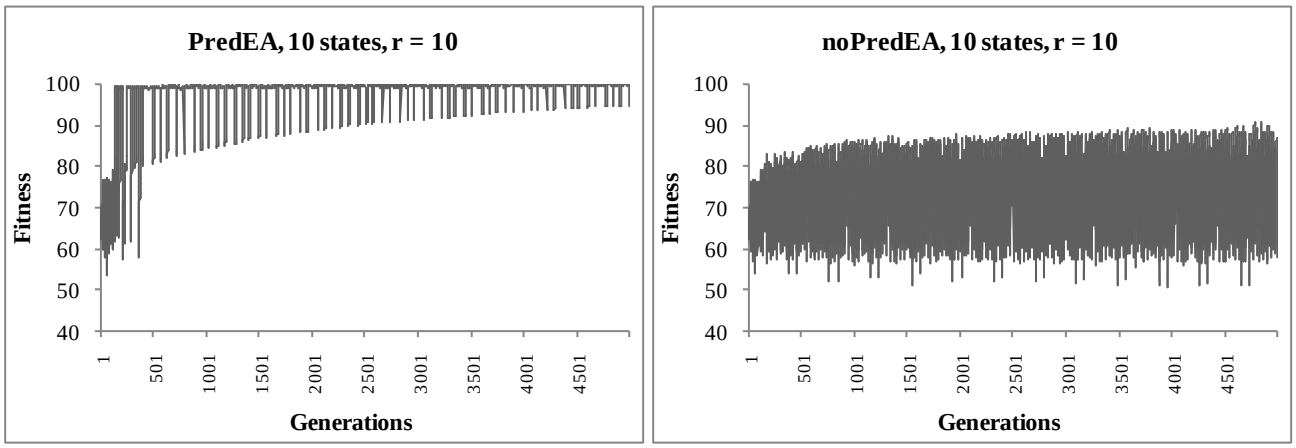


Figure 16. ***PredEA*** versus ***noPredEA***, change period r = 10, 10 states, dynamics of type P_{ij}<>1

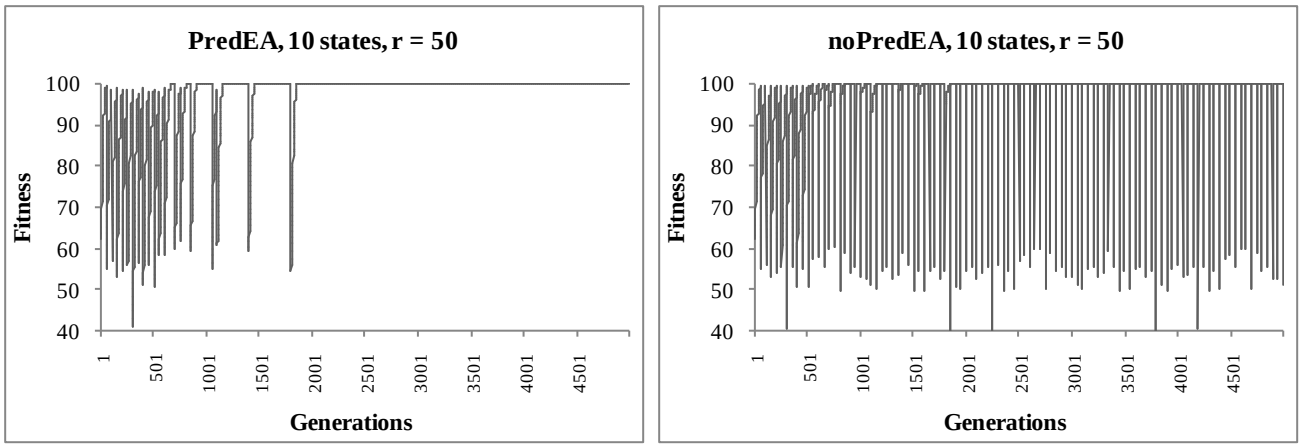


Figure 17. ***PredEA*** versus ***noPredEA***, change period r = 50, 10 states, dynamics of type P_{ij}<>1

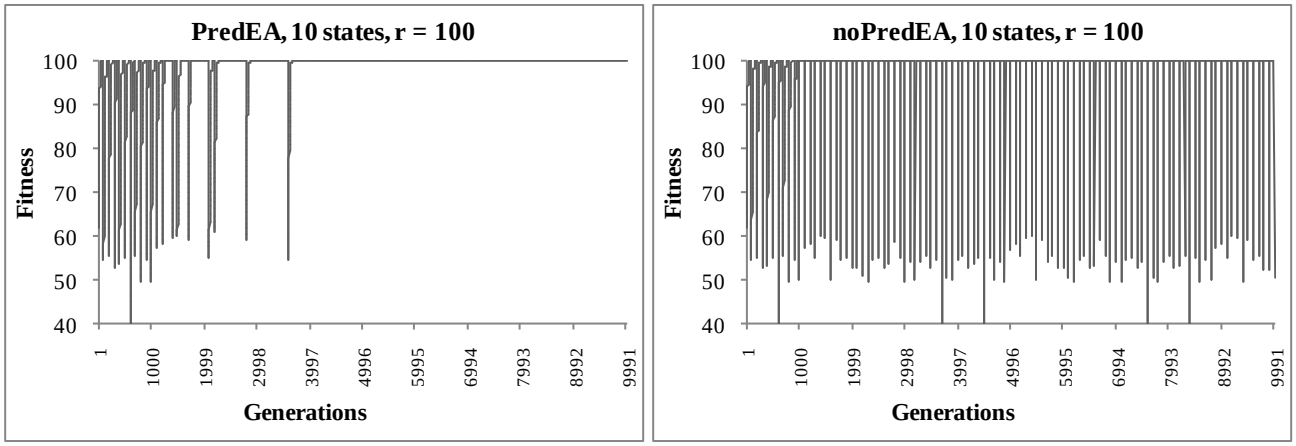


Figure 18. ***PredEA*** versus ***noPredEA***, change period r = 100, 10 states, dynamics of type P_{ij}<>1

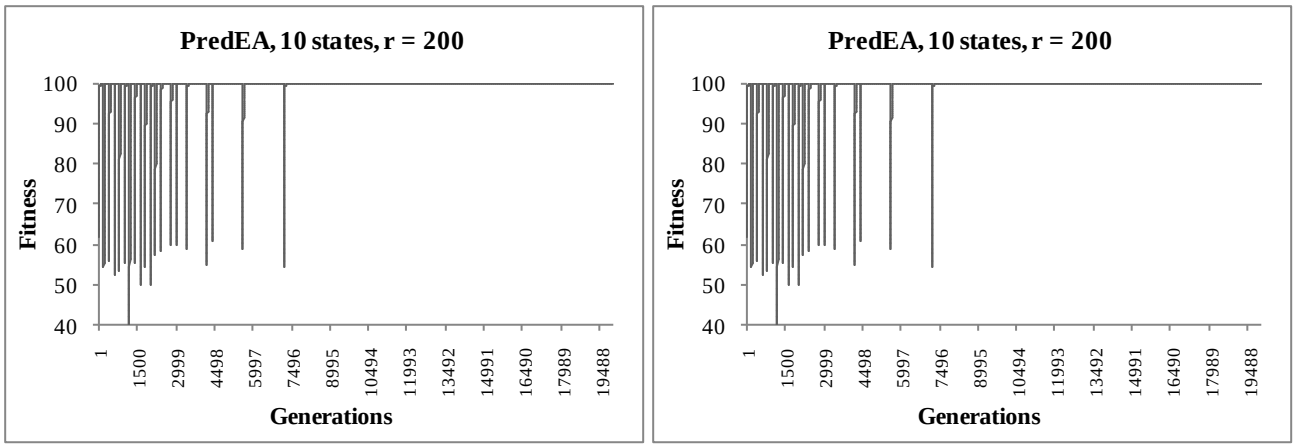


Figure 19. ***PredEA*** versus ***noPredEA***, change period r = 200, 10 states, dynamics of type P_{ij}<>1

7.2.2. Results on cyclic-pattern environments

Similar results were obtained in *cyclic-pattern* environments. As referred before, for the patterns 50-60-70 and 100-150-100, the Δ constant set to 5 was not suitable to the accuracy of the linear regression module. For these two situations, we repeated the experiments adjusting the constant value to 10 and 25. The results, as we expected, were better and since the levels of population's diversity in the two cases are practically the same, this increase in the performance is due to the introduction of retrieved memory information before the change happens. Using a fixed value for the Δ constant a limitation of the system and will be improved in future work.

Table 5 shows the global results obtained in all the experiments carried out. Best results are marked with bold. The statistical results of comparing the two EAs are given in Table 6.

		Number of states				
Pattern	Algorithm	3	5	10	20	50
5-10-5	PredEA - Δ=5 (Pij = 1)	**97.49**	**97.27**	**97.50**	**97.65**	**95.63**
	PredEA - Δ=5 (Pij<>1)	**97.10**	**96.13**	**96.77**	**96.79**	**94.60**
	NoPredEA (Pij = 1)	91.98	90.35	85.89	79.36	72.62
	NoPredEA (Pij<> 1)	93.18	90.58	86.01	79.90	73.57
10-20-10	PredEA - Δ=5 (Pij = 1)	**98.36**	**98.11**	**97.95**	**97.46**	**94.64**
	PredEA - Δ=5 (Pij<>1)	**98.24**	**97.49**	**97.12**	**96.29**	**93.01**
	NoPredEA (Pij = 1)	91.26	92.25	88.13	80.97	73.40
	NoPredEA (Pij<> 1)	94.71	91.52	87.89	81.67	74.51
50-60-70	PredEA - Δ=5 (Pij = 1)	99.54	99.37	98.82	97.04	94.60
	PredEA - Δ=5 (Pij<>1)	99.52	99.34	98.79	97.24	94.38
	PredEA - Δ=10 (Pij = 1)	**99.80**	**99.65**	**99.31**	**98.60**	**96.52**
	PredEA - Δ=10 (Pij<>1)	**99.79**	**99.61**	**99.15**	**98.23**	**95.92**
	NoPredEA (Pij = 1)	99.44	99.16	98.64	96.45	94.54
	NoPredEA (Pij<> 1)	99.44	99.17	98.58	97.02	95.13
100-150-100	PredEA - Δ=5 (Pij = 1)	99.64	99.52	99.20	98.07	95.94
	PredEA - Δ=5 (Pij<>1)	99.65	99.53	99.23	98.17	96.45
	PredEA - Δ=25 (Pij = 1)	**99.89**	**99.79**	**99.65**	**99.29**	**98.25**
	PredEA - Δ=25 (Pij<>1)	**99.89**	**99.78**	**99.55**	**99.10**	**97.95**
	NoPredEA (Pij = 1)	99.71	99.59	99.27	98.31	97.24
	NoPredEA (Pij<> 1)	99.71	99.59	99.29	98.52	97.56

Table 5. Global results on cyclic-pattern environments, with change patterns 5-10-5, 10-20-10, 50-60-70 and 100-150-100

PredEA's performance in the last pattern, using the Δ=5 was not satisfactory. ***noPredEA*** was better in this case. Adjusting the value of the constant to 25, we could significantly increase the ***PredEA***'s performance.

			Nº of states				
	T-test results	*Pattern*	3	5	10	20	50
CYCLIC PATTERN	PredEA - Δ=5 -- noPredEA (Pij == 1)	5-10-5	++	++	++	++	++
	PredEA - Δ=5 -- noPredEA (Pij<> 1)		++	++	++	++	++
	PredEA - Δ=5 -- noPredEA (Pij == 1)	10-20-10	++	++	++	++	++
	PredEA - Δ=5 -- noPredEA (Pij<> 1)		++	++	++	++	++
	PredEA - Δ=5 -- noPredEA (Pij == 1)	50-60-70	++	++	++	++	+
	PredEA - Δ=5 -- noPredEA (Pij<> 1)		++	++	++	++	--
	PredEA - Δ=10 -- noPredEA (Pij == 1)		++	++	++	++	++
	PredEA - Δ=10 -- noPredEA (Pij<> 1)		++	++	++	++	++
	PredEA - Δ=10 -- PredEA_INT 5 (Pij == 1)		++	++	++	++	++
	PredEA - Δ=10 -- PredEA_INT 5 (Pij<> 1)		++	++	++	++	++
	PredEA - Δ=5 -- noPredEA (Pij == 1)	100-150-100	--	--	--	--	--
	PredEA - Δ=5 -- noPredEA (Pij<> 1)		--	--	-	--	--
	PredEA - Δ=25 -- noPredEA (Pij == 1)		++	++	++	++	++
	PredEA - Δ=25 -- noPredEA (Pij<> 1)		++	++	++	++	++
	PredEA - Δ=25 -- PredEA_INT 5 (Pij == 1)		++	++	++	++	++
	PredEA - Δ=25 -- PredEA_INT 5 (Pij<> 1)		++	++	++	++	++

Table 6. Statistical significance of comparing ***PredEA*** with ***noPredEA*** (ttest) in cyclic-pattern environments

Figure 16 graphically shows the achieved results. Once again, in rapidly changing environments (patterns 5-10-5 and 10-20-10) the incorporation of prediction and the anticipation of change allowed outstanding improvements in the algorithm's performance. ***PredEA*** also ensures best scores as the number of states increases. In the other two situations, using a suitable value for the Δ constant, ***PredEA*** also achieves the best results.

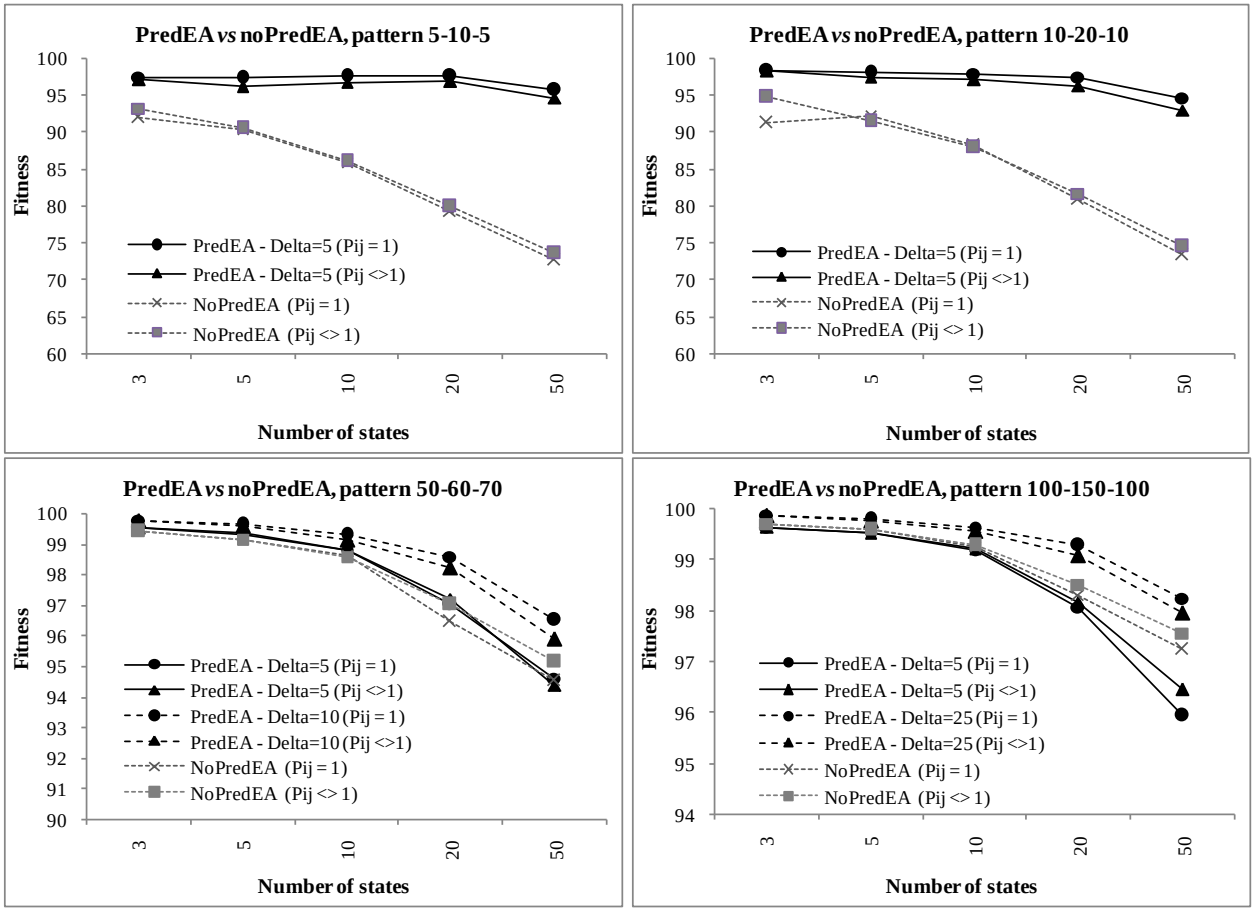


Figure 20. Global results for ***PredEA*** and ***noPredEA*** in cyclic-pattern environments (5-10-5, 10-20-10, 50-60-70 and 100-150-100)

In figures 17 to 20 we can see the behaviour of the algorithms in the first 5000 generations, using 10 different states with *cyclic* (P_{ij}=1) dynamics. As in the case of cyclic-periodic environments, analysed in the previous section, we observe the presence of the *learning* and *equilibrium* phases when using ***PredEA.*** Once again, in quickly changing environments the equilibrium was not attained in the first 5000 generations. ***noPredEA*** behave in the same way as the previous cases.

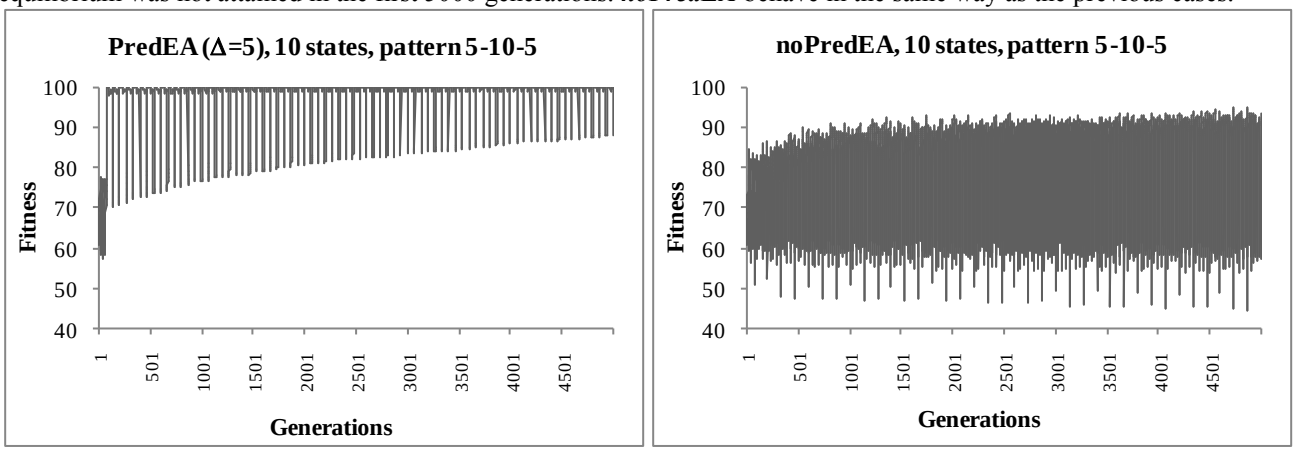


Figure 21. ***PredEA*** versus ***noPredEA***, pattern 5-10-5, 10 states, dynamics of type P_{ij}=1

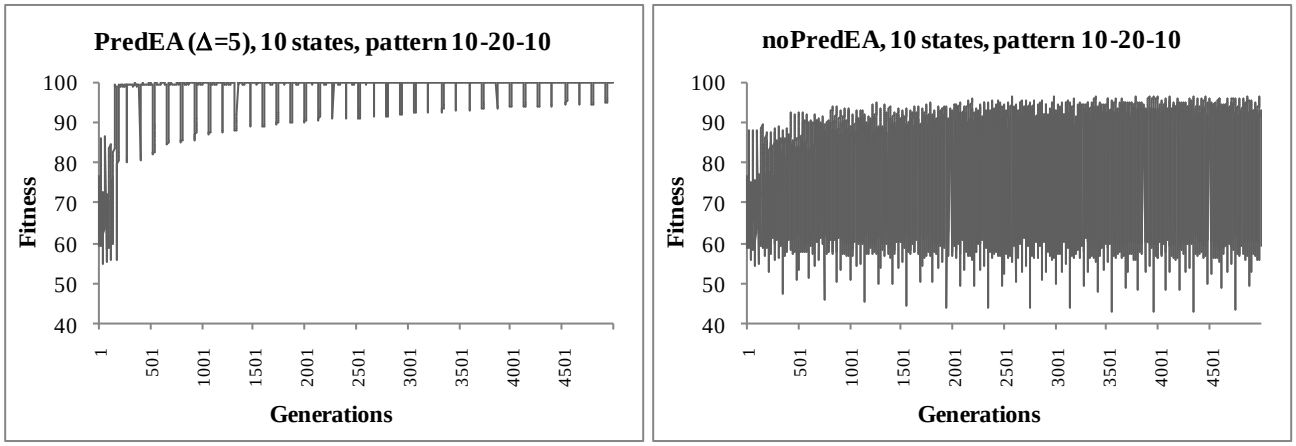


Figure 22. ***PredEA*** versus ***noPredEA***, pattern 10-20-10, 10 states, dynamics of type P_{ij}=1

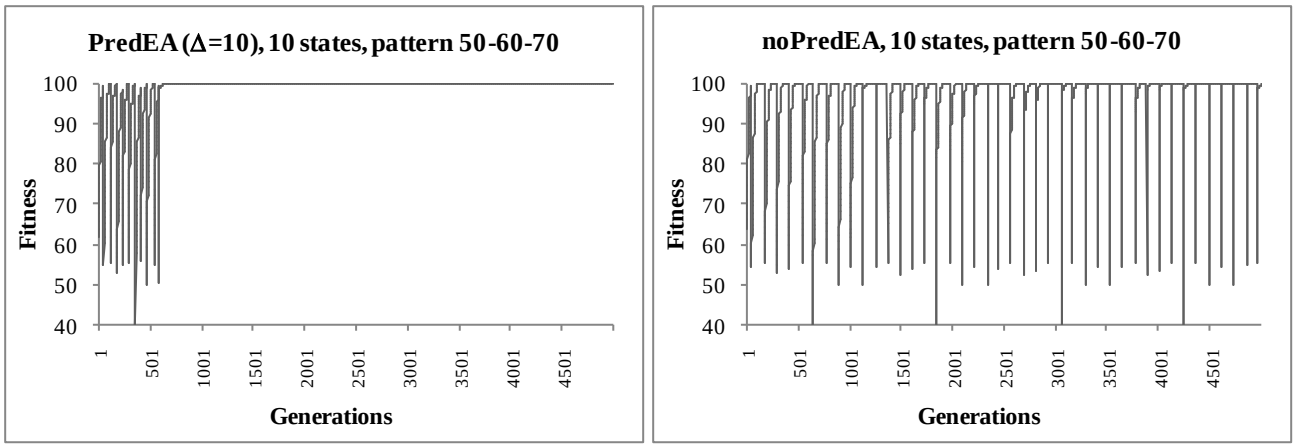


Figure 23. ***PredEA*** (Δ=10) versus ***noPredEA,*** pattern 50-60-70, 10 states, dynamics of type P_{ij}=1

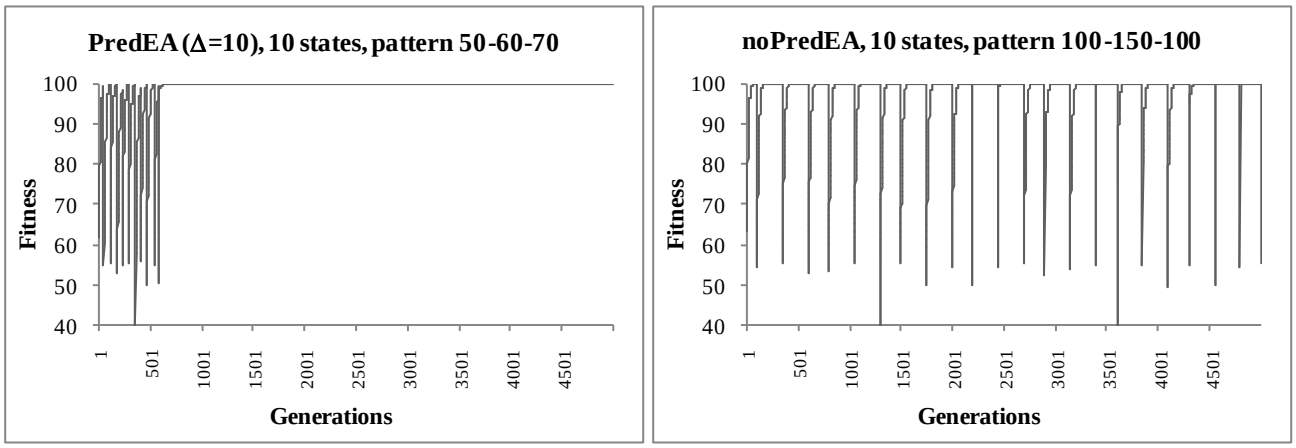


Figure 24. ***PredEA*** versus ***noPredEA*** (Δ=25), pattern 100-150-100, 10 states, dynamics of type P_{ij}=1

Similar results were obtained in environments changing in a *probabilistic* manner. Figures 25 to 28 show the results obtained with 10 states.

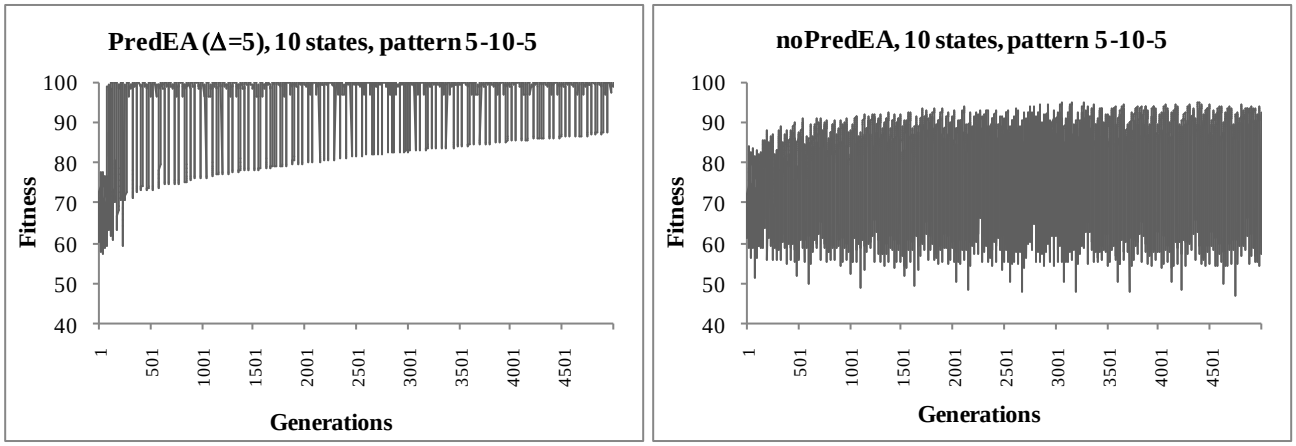


Figure 25. ***PredEA*** versus ***noPredEA*** (Δ=5), pattern 5-10-5, 10 states, dynamics of type P_{ij}<>1

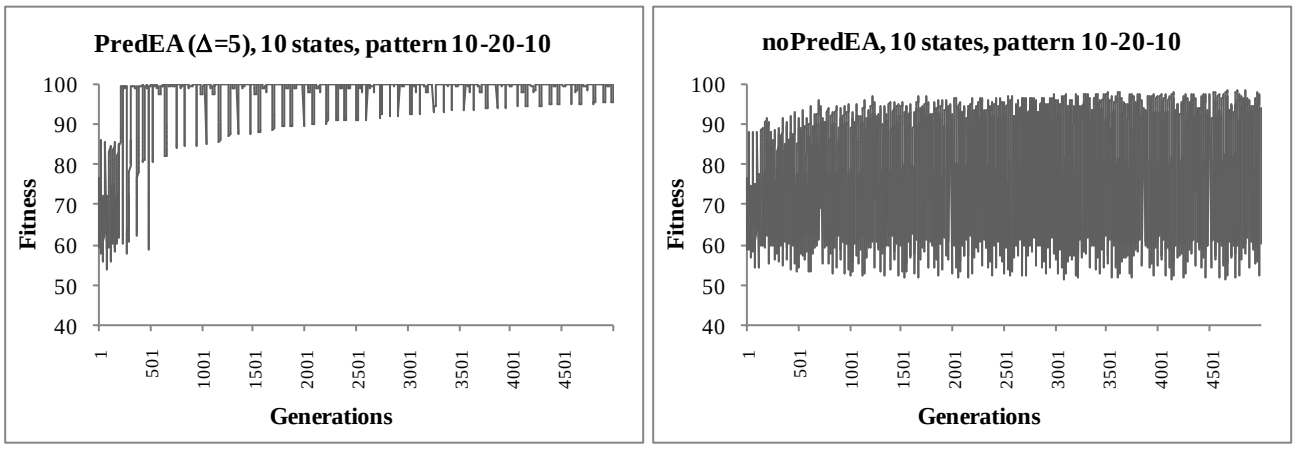


Figure 26. ***PredEA*** versus ***noPredEA*** (Δ=5), pattern 10-20-10, 10 states, dynamics of type P_{ij}<>1

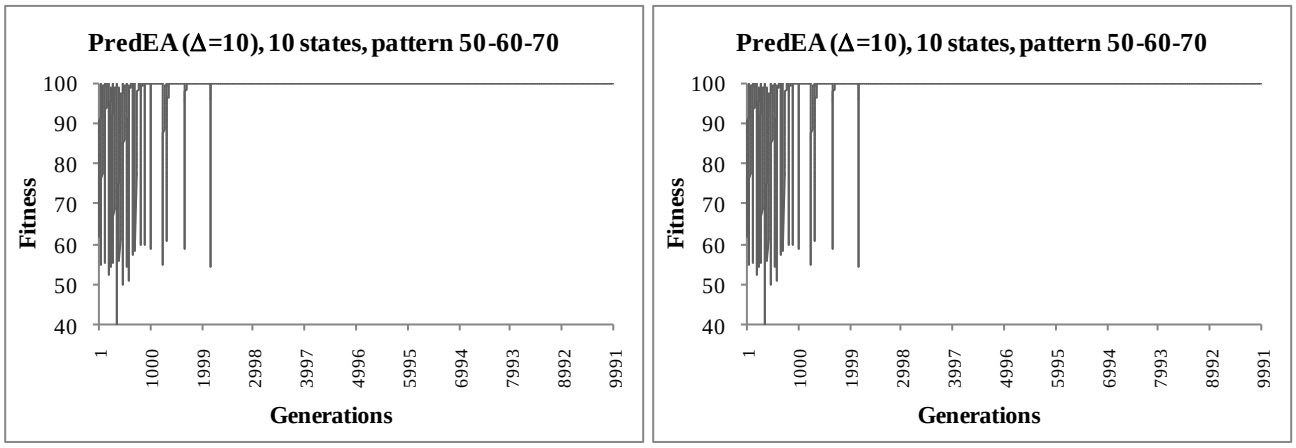


Figure 27. ***PredEA*** versus ***noPredEA*** (Δ=10), pattern 50-60-70, 10 states, dynamics of type P_{ij}<>1

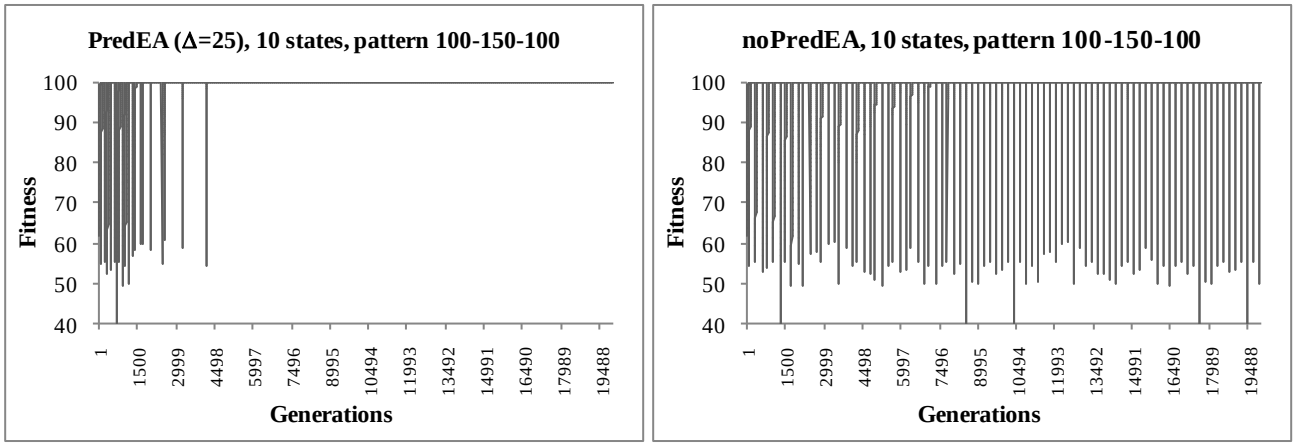


Figure 28. ***PredEA*** versus ***noPredEA*** (Δ=25), pattern 100-150-100, 10 states, dynamics of type P_{ij}<>1

8 Conclusions and Future Work

We have proposed the integration of prediction capabilities in a standard memory-based evolutionary algorithm to cope with changing environments. Two modules were used: one based on linear regression to predict when next change will occur, the other uses a Markov chain which stores the environmental information and is used to predict the next possible state(s).

The investigated algorithm, called ***PredEA,*** was tested using several instances of the dynamic bit matching problem and compared with a similar EA without the prediction modules.

Analyzing the obtained results, some conclusions can be stated: first, anticipating the moment of change and using information gathered in the past to prepare the future, significantly improves the EA's adaptability. Second, these improvements have more impact when the environment changes faster. In these cases, if the prediction capabilities are removed, the algorithm has a very poor performance. Third, ***PredEA*** is more robust than ***noPredEA*** as the number of repeated states increases. Fourth, the linear regression method, used to predict the moments of subsequent changes, is suitable only for a restricted kind of changing periods. In fact, if there is an error because the effective change occurred before the predicted one, the algorithm's performance is compromised. This is due to an untimely use of the solution obtained from the Markov chain module, making the use of Markov chain's predictions unhelpful. Finally, the use of a Markov chain to store the environmental information proved to be a powerful mechanism to keep the history from the changing dynamics which allows the algorithm to learn and predict which states can appear in the next step.

The major limitations of the proposed architecture are related to the linear regression module. First, the use of linear regression to predict future change points is feasible only for certain patterns. For more complex patterns, linear regression fails, giving large prediction errors. Second, the use of a fixed value for the error, assumed in the linear regression predicted values, it's not very effective. If an unsuitable value is used for this constant, the algorithm's performance considerably decreases. Some enhancements are being introduced to improve this module: the use of non linear regression and the dynamic adjustment of the Δ constant during the simulation.

Other issues we intend to investigate include: *(a)* the influence of the severity change, *(b)* what should be stored as a representation of the "environmental information", *(c)* the application of the proposed approach to noisy environments and *(d)* the use of high-order Markov chains to allow predictions based not only in the current state, but use more information from the past to estimate the future. Also, more experimentation, using different benchmark problems, must be carried out.

9 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.

10 References

1. J. Branke. Memory Enhanced Evolutionary Algorithms for Changing Optimization Problems. IEEE Congress on Evolutionary Computation (CEC 1999), pp. 1875-1882, IEEE, 1999.
2. J. Branke, D. Mattfeld. Anticipation in dynamic optimization: The scheduling case. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo, and H.-P. Schwefel (Eds), Parallel Problem Solving from Nature (PPSN VI), pp. 253-262, Springer Verlag, 2000.
3. J. Branke, T. Kaußler, C. Schmidt, H. Schmeck. A Multi-Population Approach to Dynamic Optimization Problems. I. Parmee (Ed.): Adaptive Computing in Design and Manufacture (ACDM 2000), pp. 299-308, Springer, 2000.
4. P. A. N. Bosman. Learning, Anticipation and Time-deception in Evolutionary Online Dynamic Optimization. In S. Yang and J. Branke, editors, GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization, 2005.
5. P. A. N. Bosman, H. La Poutré. Computationally Intelligent Online Dynamic Vehicle Routing by Explicit Loa Prediction in Evolutionary Algorithm. In T. P. Runarsson, H.-G. Beyer, E. Burke, J.J. Merelo-Guervós, L. D. Whitley, X. Yao (Eds), Proceedings of Parallel Problem Solving from Nature (PPSN IX), Lecture Notes in Computer Science 4193, pp. 312-321, Springer Verlag Berlin, 2006.
6. P. A. N. Bosman, H. La Poutré. Inventory Management and the Impact of Anticipation in Evolutionary Stochastic Online Dynamic Optimization. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2007), pages 268-275, IEEE Press, Piscataway, New Jersey, 2007.
7. P. A. N. Bosman. Learning and Anticipation in Online Dynamic Optimization. In S. Yang, Y.S. Ong and Y. Jin, editors, Evolutionary Computation in Dynamic and Uncertain Environments, pages 129-152, Springer-Verlag, Berlin, 2007.
8. H. G. Cobb. An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms having Continuous, Time-Dependent Nonstationary Environments. TR AIC-90-001, Naval Research Laboratory, 1990.
9. J. J. Grefenstette. Genetic Algorithms for Changing Environments. In R. Männer and B. Manderick (Eds), Parallel Problem Solving from Nature (PPNS 2), pp. 137-144, North-Holland, 1992.
10. N. Mori, H. Kita, Y. Nishikawa. Adaptation to a changing environment by means of the thermo dynamical genetic algorithm. In H.-M. Voigt (Ed.), Parallel Problem Solving from Nature (PPSN IV), 1141 in LNCS, pp. 513-522. Springer Verlag, 1996.
11. D. S. Moore, G. P. McCabe. Introduction to the Practice of Statistics. Freeman and Company, 4th Edition, 2003.
12. J. R. Norris. Markov Chains. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1997.
13. A. Simões, E. Costa. An Immune System-Based Genetic Algorithm to Deal with Dynamic Environments: Diversity and Memory. Proc. of the 6th Int. Conf. on Artificial Neural Networks, pp. 168-174, Roanne, France, 23-25 April, Springer, 2003.
14. A. Simões, E. Costa. Variable-size Memory Evolutionary Algorithm to Deal with Dynamic Environments. In M. Giacobini et al. (Eds.): EvoWorkshops 2007, Applications of Evolutionary Computing, LNCS 4448, pp. 617–626, Springer Verlag, 2007.
15. A. Simões and E. Costa. Improving Memory's Usage in Evolutionary Algorithms for Changing Environments. Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2007), Singapore, 25-28 September, pp. 276-283, IEEE, 2007.
16. A. Simões, E. Costa. Using Linear Regression to Predict Changes in Evolutionary Algorithms dealing with Dynamic Environments. CISUC TR 2007/005, ISSN 0874-338X, April 2007.
17. P. D. Stroud. Kalman-extended Genetic Algorithm for Search in Nonstationary Environments with Noisy Fitness Evaluations. IEEE Transactions on Evolutionary Computation, 5(1):66-77, 2001.
18. J. van Hemert, C. Van Hoyweghen, E. Lukshandl, K. Verbeeck. A Futurist Approach to Dynamic Environments. GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems, pages 35-38, 2001.
19. M. Wineberg, F. Oppacher (2000). Enhancing the GA's Ability to Cope with Dynamic Environments. In D. Whitley (Ed.) Proc. 2nd Genetic and Evolutionary Computation. Conference (GECCO 2000), pp. 3-10, San Francisco: Morgan Kaufmann, 2000.
20. S. Yang, X. Yao. Experimental Study on Population-Based Incremental Learning Algorithms for Dynamic Optimization problems. Soft Computing, vol. 9, nº 11, pp. 815-834, 2005.
21. S. Yang. Explicit Memory Schemes for Evolutionary Algorithms in Dynamic Environments. In S. Yang, Y-S. Ong, Y. Jin (Eds.), Evolutionary Computation in Dynamic and Uncertain Environments, pp. 3-28, Studies on Computational Intelligence, Springer, 2007.