Improving Prediction in Evolutionary Algorithms for Dynamic Environments

Anabela Simões *
Coimbra Polythechnic
Rua Pedro Nunes - Quinta da Nora
3030-199 Coimbra, Portugal
abs@isec.pt

Ernesto Costa
CISUC, University of Coimbra
Polo II - Universidade de Coimbra
3030-290 Coimbra - Portugal
ernesto@dei.uc.pt

ABSTRACT

The addition of prediction mechanisms in Evolutionary Algorithms (EAs) applied to dynamic environments is essential in order to anticipate the changes in the landscape and maximize its adaptability. In previous work, a combination of a linear regression predictor and a Markov chain model was used to enable the EA to estimate when next change will occur and to predict the direction of the change. Knowing when and how the change will occur, the anticipation of the change was made introducing useful information before it happens. In this paper we introduce mechanisms to dynamically adjust the linear predictor in order to achieve higher adaptability and robustness. We also extend previous studies introducing nonlinear change periods in order to evaluate the predictor's accuracy.

Categories and Subject Descriptors

I. [**Computing Methodologies**]: ARTIFICIAL INTELLIGENCE—*Problem Solving, Control Methods, and Search*

General Terms

Algorithms, Experimentation, Performance

Keywords

Evolutionary Algorithms, Dynamic Environments, Prediction, Markov chains, Linear regression

1. INTRODUCTION

Evolutionary Algorithms (EAs) are powerful tools to solve a great variety of stationary optimization problems. Unfortunately, techniques that are good for static problems may not be effective for dynamic problems. Hence, some modifications have been introduced in EAs in order to deal with this kind of problems: the use of memory ([3], [18], [12]), the maintenance of population's diversity ([7]) or the use of several populations ([5]). See [9] and [17] for a in-dept review.

When the environment is dynamic, in some cases, a certain repeated behavior 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 continues to be an open subject which must be further explored. The present paper extends previous work and introduces new ideas for this subject. The investigated method uses a traditional memory-based EA combined with three other modules, responsible for predicting the future and preparing the EA for the change: the first is based on linear regression and predicts when next change will take place, the second is supported by a Markov chain and estimates the environments that may appear in the future; the third collects information from the previous modules and prepares the EA to the change. The main goal of this paper is to make the linear predictor more adaptable and robust to different situations. This is done by making adjustable some elements of the linear predictor. Four different mechanisms are proposed and introduced in the linear regression module enhancing it and increasing its adaptability and robustness. The predictor's accuracy is evaluated in different situations using change periods following a *close-to-linear* behavior. Additionally, two different change periods, following *nonlinear* curves are introduced and the efficacy of the proposed predictors is evaluated.

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 Markov chain prediction and the linear regression module. The proposed mechanisms to improve the linear predictor are 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. Branke

*also belongs to CISUC

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

et al. ([6]) try to understand how the decisions made at one stage influence the problems encountered in the future. Stroud ([15]) 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. Van Hemert et al. ([16]) 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. Bosman ([1], [2]) proposed 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. Bosman propose an algorithmic framework integrating evolutionary computation with machine learning and statistical learning techniques to estimate future situations. Rossi et al ([10]) compare different techniques to improve the search for tracking a moving optimum using the information provided by a predictor mechanism using Kalman filters. The use of linear regression to predict the moment of next change was initially proposed by Simões and Costa. ([11]). Later, in [14] a predictor based on Markov chain was added and used to predict which environments may appear in the future.

3. PREDICTION IN THE EA

Simões and Costa [14] proposed a computational model called *PredEA* to deal with dynamic environments. 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, uses a predictor based on linear regression and, using information about when previous changes have occurred, estimates the generation **when** the next change will be observed. The second module, uses a Markov chain to memorize information about the different environments that appear along time. Using this information, this module provides predictions about **which environments** may appear in the future. An additional module, manages the information provided by the two predictors and decides when to use it to anticipate the change and prepare the main population to the next change. Figure 1 shows the proposed architecture. Here is a brief description about

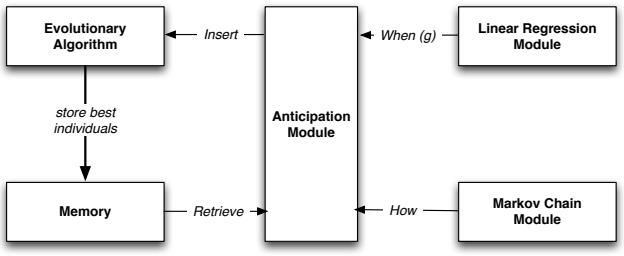

Figure 1: Computational architecture of PredEA

each one of the components of the computational system:
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 [17].
Linear regression module (LR): this module saves the generations when different changes have occurred and uses this information to foresee when next change may take place. The predictor uses a standard linear regression technique.
Markov chain module (MC): 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 environment(s).
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. However, we emphasize that all this information is **unknown** by the EA and that the Markov model is constructed during the simulation. The activation of the anticipation module (**A**) must be done at the correct time in order to prepare the population to the next environment(s) predicted by the **MC** module. This "good moment" is provided by the **LR** 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, the change is detected when it happens and the EA's performance suffers a sudden decrease taking some time to readapt to the new environment. A change is detected if at least one individual in the memory changes its fitness (as it is suggested in [3]). A parameter, called Δ, is used to decide **how many** generations before the predicted moment of change the **A** module is activated. The value of this parameter is also used to cover minor prediction errors associated to the **LR** module. The value of Δ must be chosen in order to assure that the **A** module is activated to prepare the population **before** the change. The pseudocode and the detailed description of the **PredEA** can be found in [13]. In previous work [14] a **constant value** was used for

Δ. This value was set before the algorithm starts running and if the prediction errors of the **LR** module were greater than expected it could not be enough to assure that the **A** module was activated *before* the change. Prediction errors can be **positive** or **negative**. A negative error means that the predicted value for next change (g) was smaller than the real value (g'), i.e, $g < g'$ and thus the anticipation of change can be made. A positive error means the opposite. In the latter case, if the value of Δ is greater than the error, the **A** module will be activated before the change, otherwise, the change is detected when it occurs and no effective anticipation is made. In this paper, we propose four different methods to adjust the value of Δ during the run aiming to make the system more efficient and robust. In the next section we describe the methods we implemented to make Δ an adaptable parameter, learning with previous errors and correcting its value *on-the-fly*, for each specific situation.

4. IMPROVING THE LINEAR REGRESSION MODULE

The **LR** module works fine if the change period follows a linear (or *close-to-linear*) trend [14]. 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 foreseen by the **LR** module. As referred before, in order to be effective, the **A** module must act **before** the change. Therefore, if the change is observed at generation g', the value of Δ must assure that the condition $\Delta > |g - g'|$ is observed. In addition, the value of Δ must minimize the computational costs guaranteeing that the anticipation is as closer to the change as possible. Thus, the choice of the value of Δ must be careful and correct. The use of a constant value for Δ is a weakness of the system: for one hand requires preliminary experimentation to decide what value to choose, for another, if the conditions of the change period are altered, a new value must be chosen. To overcome this limitation we propose several mechanisms that use the previously observed prediction errors to continuously change the value of Δ. Every time a change happens the predictions provided by the **LR** module are evaluated and if some prediction error is observed, the value of Δ is changed according to those observed errors. In all studied approaches the parameter Δ is initialized with the value 5 and this value is used for the first two changes when no predictions can be made. After that, the value of Δ is updated according to the studied methods, but it cannot be lower than a minimum value of 2. This restriction assures that the preparation to next change is made at least two generations before it happens. Next subsections describe the four studied methods.

4.1 Using the maximum prediction error

This method will be called Max_Err. The parameter Δ is initialized with the value 5 and in the next changes the its value is changed using the **maximum** observed prediction error given by the linear predictor.

$$\Delta_1(k) = \begin{cases} 5 & \text{if } k = 1, 2 \\ max\{2, e_0, e_1, ..., e_k\} & \text{if } k > 2 \end{cases} \quad (1)$$

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

4.2 Using the average of the positive prediction errors

This approach will be known as $Av\_Err(+)$. As before, the parameter Δ is initialized with the value 5 and this value is used for the first two changes when no predictions can be made. In the next changes the value of Δ is changed using the average of the **positive errors** given by the linear predictor:

$$\Delta_2(k) = \begin{cases} 5 & \text{if } k = 1, 2 \\ max\{2, \frac{\sum_{i=1}^{k} e_i}{k}\} & \text{if } k > 2 \text{ and } e_i > 0 \end{cases} \quad (2)$$

4.3 Using the average of the absolute value of all the prediction errors

Known as $Av\_Err(all)$, at change k this method updates the value of Δ using the average of the **absolute values of all errors**.

$$\Delta_3(k) = \begin{cases} 5 & \text{if } k = 1, 2 \\ max\{2, \frac{\sum_{i=1}^{k} |e_k|}{k}\} & \text{if } k > 2 \end{cases} \quad (3)$$

4.4 Using the maximum prediction error and the average of the positive prediction errors

We will call this method Max_Av_Err. After the two first changes the value of Δ is computed using the average of the sum of the maximum observed error and the average of all positive errors.

$$\Delta_4(k) = \begin{cases} 5 & \text{if } k = 1, 2 \\ max\{2, \frac{\Delta_1(k) + \Delta_2(k)}{2}\} & \text{if } k > 2 \text{ and } e_i > 0 \end{cases} \quad (4)$$

5. EXPERIMENTAL DESIGN

Experiments were carried out to compare the different versions of PredEA. The algorithm using a constant value for Δ as proposed in [14] ($PredEA\_Const$) and the different versions of the algorithm, using the methods described in the previous section. The four EAs will be called:
$PredEA\_Max\_Err$**:** the Δ parameter is adapted using the maximum observed prediction error;
$PredEA\_Av\_Err(+)$**:** the Δ parameter is adapted using the average of the positive observed prediction errors;
$PredEA\_Av\_Err(all)$**:** the Δ parameter is adapted using the average of the absolute value of all observed prediction errors;
$PredEA\_Max\_Av\_Err$**:** the Δ parameter is adapted using a linear combination of the maximum and the average of the positive observed prediction errors.
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. The knapsack problem consists in selecting a number of items to a knapsack with limited capacity. Each item has a value and a weight and the objective is to choose the items that maximize the total value, without exceeding the capacity of the bag. For the bitmatching problem, a set of different binary templates was generated at the beginning of the run. When a change happens, a different template is chosen from that set. We used templates of length 100. For

the knapsack problem, different capacities are generated at the beginning of the run. Each different capacity value differs from the previous one in 10%. When a change happens, a different value is chosen from that set.

Experimental Setup 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 Δ constant referred above was 5 generations. 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.
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. More details can be found in [13].

When does the environment change?
Two different types of change period were used. In the first, also used in previous work [14], the change period is generated through the repetition of a certain pattern. This type of change period follows a linear trend 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). Another type of change, based on nonlinear functions, was analyzed in this work. This situation was introduced to see how the **LR** module behaves when the period of change follows a nonlinear behavior. Two different nonlinear curves were studied, according to the following equations [8]:

$$f_1(x) = a + bx + cx^2,\ \text{a} = 60,\ \text{b} = 2,\ \text{c} = 0.1\ \text{[NL1]} \quad (5)$$

$$f_2(x) = \frac{ax}{b + cx},\ \text{a} = 63,\ \text{b} = 0.63,\ \text{c} = 0.005\ \text{[NL2]} \quad (6)$$

In the first (f_1), the environment changes slower at the beginning and it becomes 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.

6. RESULTS

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 1 shows the accuracy of the linear predictor (Efficacy column), the average of prediction error, the absolute value of the error is also provided in terms of absolute value and the corresponding value of Δ. As we can see in Table 1 the value of Δ influences the prediction efficacy. For the changes following a linear trend, based on a fixed pattern, accurate predictions were provided. In this case, each adjusting method used a different value for Δ and, in general, the $\Delta = Max\_Av\_Err$ provided the best responses. For the situations where the changes occurred in a nonlinear trend, the predictor failed. Since the predictor is based on linear regression these results were expected. As shown in Table 1, the predictions in the two studied situations were mainly wrong. In the first situation (NL1) the predicted values correspond to generations before the real change (negative value), but the associated error is huge and to a great extent the predicted value was not even close to the next change. In the second situation (NL2) all the predicted values were provided after the change occur (positive values for the average prediction error), so the algorithm reacted untimely and wrong. Table 2 shows the variation of the prediction error for the analyzed situations. As we can see in the four situations based on linear behaviors the prediction error tends to decrease along time. The opposite is observed in the nonlinear situations. The linear prediction module used in the EA is obviously unsuitable for situations where the changes follow a nonlinear pattern.

Algorithm's Performance
The EA's efficiency is directly related to the predictor's accuracy. In the nonlinear situations where the accuracy of the predictor was very poor, the performance is similar for all the situations including those where no prediction is used. This happens because, in fact, there was no prediction for all the situations. In the remaining cases we can see that the use of the prediction mechanisms improved the EA's performance, especially for rapid change periods and using more different states.
Table 3 shows the results obtained for the bitmatching problem. The best scores are marked with bold. Figures 2 to 5 show the evolution of the algorithm's performance along time. The plots show the offline performance obtained by the EA **with** and **without** prediction. Offline performance is calculated as the average of the best values found so far at each time step ([4]). Only the individuals evaluated since the last change are considered:

$$offline = \frac{1}{G}\sum_{t=1}^{G} e'(t), e'(t) = max\{e_\tau, e_{\tau+1}, ..., e_t\} \quad (7)$$

where G is the total number of generations and τ is the last time step before t at which a change in the environment occurred.
The Δ adjustment type used in the next examples was the Max_Av_Err strategy. We show results for both problems using 10 states in deterministic and probabilistic changes, for the **5-10-5** and **NL1** patterns. More results can be consulted in [13]. As we can see the use of prediction considerably improves the EA's performance, mostly in situations where the number of generations between changes is smaller.
Statistical Analysis The major statistical results of comparing the different methods can be found in [13]. We used paired one-tailed t-test at a 0.01 level of significance [8]. We've just saw that choosing different values for the parameter Δ affects the prediction's accuracy and the statistical analysis support that this difference is statistically significant.

Table 1: Prediction accuracy using different methods

Chg. period	Adj. type	Efficacy	Av. Err.	Av. Err.(abs)	Av. Δ
5-10-5	Δ=$Const = 5$	66.71%	-0.32	1.01	5.00
	Δ=Max_Err	66.71%			5.00
	Δ=$Av\_Err(+)$	67.38%			2.00
	Δ=$Max\_Err(all)$	67.65%			2.00
	Δ=Av_Max_Err	**99.87%**			3.01
10-20-10	Δ=$Const = 5$	99.46%	-0.29	2.37	5.00
	Δ=Max_Err	67.02%			9.97
	Δ=$Av\_Err(+)$	68.10%			3.11
	Δ=$Max\_Err(all)$	68.63%			3.13
	Δ=Av_Max_Err	**99.73%**			6.06
50-60-70	Δ=$Const = 5$	98.32%	-0.29	4.43	5.00
	Δ=$Const = 10$	99.66%			10.00
	Δ=Max_Err	99.66%			12.89
	Δ=$Av\_Err(+)$	41.95%			3.32
	Δ=$Max\_Err(all)$	**99.66%**			4.34
	Δ=Av_Max_Err	**99.66%**			8.06
100-150-100	Δ=$Const = 5$	98.32%	-0.02	11.53	5.00
	Δ=$Const = 25$	**99.66%**			25.00
	Δ=Max_Err	**99.66%**			49.55
	Δ=$Av\_Err(+)$	41.95%			15.12
	Δ=$Max\_Err(all)$	**99.66%**			16.33
	Δ=Av_Max_Err	**99.66%**			31.95
NL1	Δ=$Const = 5$	0.00%	-1885.52	1885.52	5.00
	Δ=Max_Err	0.00%			5.00
	Δ=$Av\_Err(+)$	0.00%			5.00
	Δ=$Max\_Err(all)$	**1.21%**			631.70
	Δ=Av_Max_Err	0.00%			5.00
NL2	Δ=$Const = 5$	**1.01%**	778.56	778.56	5.00
	Δ=Max_Err	0.25%			775.32
	Δ=$Av\_Err(+)$	0.25%			410.25
	Δ=$Max\_Err(all)$	0.25%			410.25
	Δ=Av_Max_Err	0.25%			592.53

Table 2: Prediction errors for the different types of change periods

Chg. number	5-10-5	10-20-10	50-60-70	100-150-100	NL1	NL2
	Prediction Error					
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	5	10	-10	50	-1	2
4	3	6	13	33	-1	2
5	-3	-5	5	-25	-1	3
6	1	2	-8	10	-1	4
7	2	4	7	23	-1	6
8	-3	-5	4	-22	-1	7
9	0	1	-8	5	-2	10
10	2	4	5	20	-2	12
...						
295	1	3	3	16	-1456	1130
296	-2	-4	3	-17	-1466	1132
297	0	0	-7	0	-1476	1135
298	1	3	3	16	-1485	1137
299	-2	-4	3	-17	-1495	1139
300	0	0	-7	0	-1505	1141

Table 3: Global results for the dynamic bitmatching problem

Bitmatching Problem		Number of states									
		3		5		10		20		50	
Ch. per	Δ Type	Det.	Prob.	Det.	Prob.	Det.	Prob.	Det.	Prob.	Det.	Prob.
5-10-5	*noPredEA*	85.56	89.94	85.59	86.53	80.97	81.48	74.05	75.73	67.99	69.04
	Const = 5	94.92	94.14	92.85	91.99	89.18	88.05	85.87	85.65	81.00	80.75
	Max_Err	94.46	**98.81**	92.55	91.89	89.21	88.77	85.90	85.81	81.04	80.85
	Av_Err(+)	96.02	95.67	93.40	**97.16**	88.92	89.47	83.37	84.22	83.67	75.74
	Av_Err(*all*)	95.98	97.59	93.47	93.03	89.08	**96.61**	83.36	84.25	75.24	75.77
	Av_Max_Err	**96.57**	96.45	**94.74**	94.04	**90.52**	90.34	**86.88**	**87.10**	**94.23**	**81.53**
10-20-10	*noPredEA*	91.43	94.68	92.42	92.47	87.69	88.42	81.00	82.87	73.40	75.00
	Const = 5	97.60	**98.59**	**97.78**	**96.91**	**95.67**	**94.69**	**92.99**	**92.57**	**87.25**	**86.72**
	Max_Err	96.47	96.82	96.82	94.60	94.77	92.71	92.35	91.38	87.11	85.98
	Av_Err(+)	97.32	97.74	96.93	96.22	94.53	93.94	90.02	90.06	81.18	81.12
	Av_Err(*all*)	97.22	97.74	97.09	96.24	94.67	93.97	89.72	90.18	81.13	81.07
	Av_Max_Err	**97.79**	98.31	97.67	96.40	95.14	94.22	92.80	92.32	87.22	86.52
50-60-70	*noPredEA*	99.15	99.15	99.06	99.07	98.83	98.86	98.15	97.88	90.78	92.34
	Const = 5	99.87	99.87	99.81	99.79	99.62	99.60	**99.25**	**99.18**	**98.09**	97.85
	Const = 10	**99.89**	99.87	99.81	99.71	**99.63**	99.60	**99.25**	**99.18**	**98.09**	97.85
	Max_Err	**99.89**	**99.88**	**99.82**	99.79	99.62	99.60	99.24	**99.18**	**98.09**	97.86
	Av_Err(+)	99.46	99.45	99.38	99.36	99.17	99.17	98.66	98.49	92.96	94.26
	Av_Err(*all*)	99.88	**99.88**	99.81	99.79	99.62	99.59	**99.25**	**99.18**	98.08	**97.87**
	Av_Max_Err	**99.89**	99.87	**99.82**	**99.80**	**99.63**	**99.61**	**99.25**	**99.18**	**98.09**	97.86
100-150-100	*noPredEA*	99.57	99.57	99.52	99.52	99.41	99.42	99.25	98.98	95.22	96.17
	Const = 5	99.82	99.81	99.78	99.77	99.68	99.67	99.49	99.35	97.63	97.92
	Const = 25	**99.94**	**99.94**	**99.91**	**99.90**	**99.81**	**99.80**	**99.62**	**99.59**	**99.02**	**98.92**
	Max_Err	**99.94**	**99.94**	**99.91**	**99.90**	**99.81**	**99.80**	**99.62**	99.58	**99.02**	98.91
	Av_Err(+)	99.83	99.82	99.78	99.78	99.68	99.67	99.49	99.37	97.63	97.94
	Av_Err(*all*)	99.83	99.82	99.78	99.78	99.68	99.67	99.49	99.37	97.62	97.93
	Av_Max_Err	**99.94**	**99.94**	99.91	**99.90**	**99.81**	**99.80**	**99.62**	**99.59**	**99.02**	98.91
NL1	*noPredEA*	98.95	98.94	98.73	98.77	98.38	98.49	97.02	97.41	91.60	92.83
	Const = 5	**99.20**	98.70	99.02	98.63	99.01	**99.86**	**99.13**	98.75	98.56	98.45
	Max_Err	99.18	**99.92**	**99.04**	**98.68**	98.99	98.78	99.10	98.75	98.56	**98.46**
	Av_Err(+)	99.18	98.67	99.00	96.73	**99.02**	98.77	99.11	98.73	**99.60**	98.45
	Av_Err(*all*)	99.18	**99.92**	99.02	98.65	98.98	**99.86**	99.12	**98.77**	98.56	98.45
	Av_Max_Err	99.19	98.68	99.00	98.67	98.96	98.80	99.12	98.75	99.55	**98.46**
NL2	*noPredEA*	98.11	98.14	**97.96**	97.97	**97.58**	97.35	**97.01**	**94.40**	84.08	85.36
	Const = 5	**98.13**	98.15	**97.96**	**97.98**	**97.58**	97.37	**97.01**	94.09	**84.15**	85.48
	Max_Err	98.12	**98.15**	**97.96**	97.97	97.57	97.35	97.00	94.07	84.07	85.42
	Av_Err(+)	98.12	**98.15**	**97.96**	97.97	97.57	**97.44**	**97.01**	94.21	84.05	**85.50**
	Av_Err(*all*)	98.12	**98.15**	**97.96**	97.97	**97.58**	97.35	**97.01**	94.26	84.14	85.40
	Av_Max_Err	98.12	**98.15**	**97.96**	97.97	97.57	97.35	**97.01**	94.22	84.10	85.43

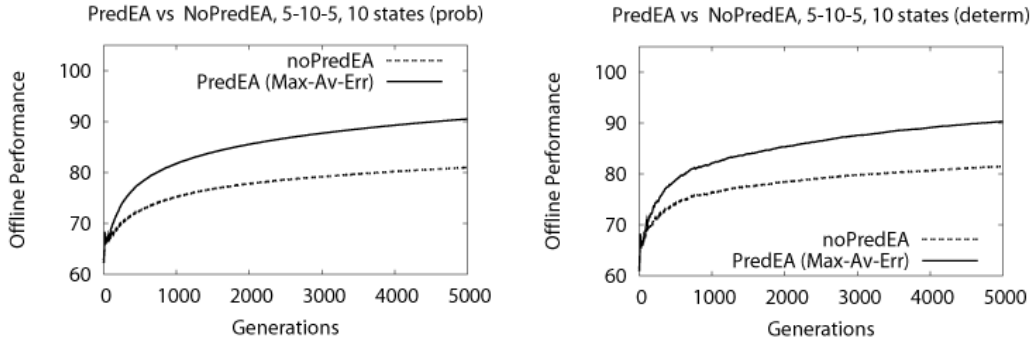

Figure 2: Comparing offline performance for the bitmatching problem: noPredEA vs PredEA, 10 states, pattern 5-10-5

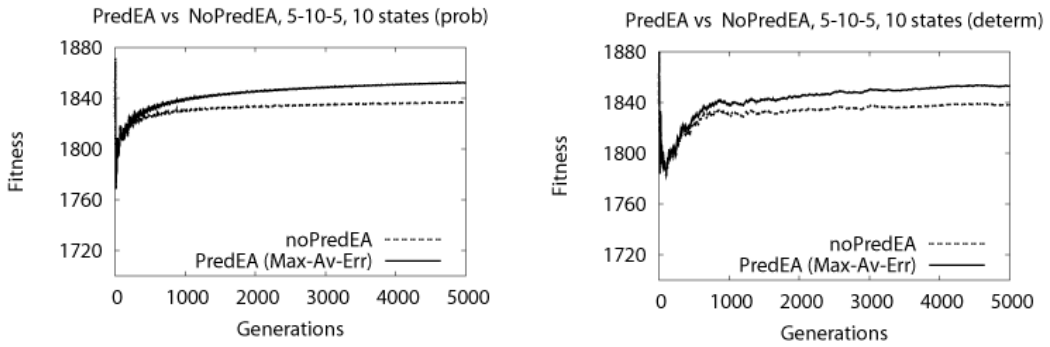

Figure 3: Comparing offline performance for the knapsack problem: noPredEA vs PredEA, 10 states, pattern 5-10-5

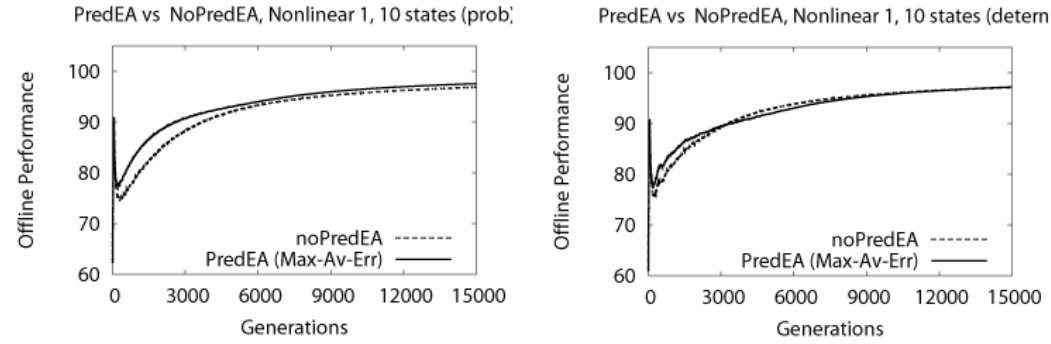

Figure 4: Comparing offline performance for the bitmatching problem: noPredEA vs PredEA, 10 states, nonlinear 1

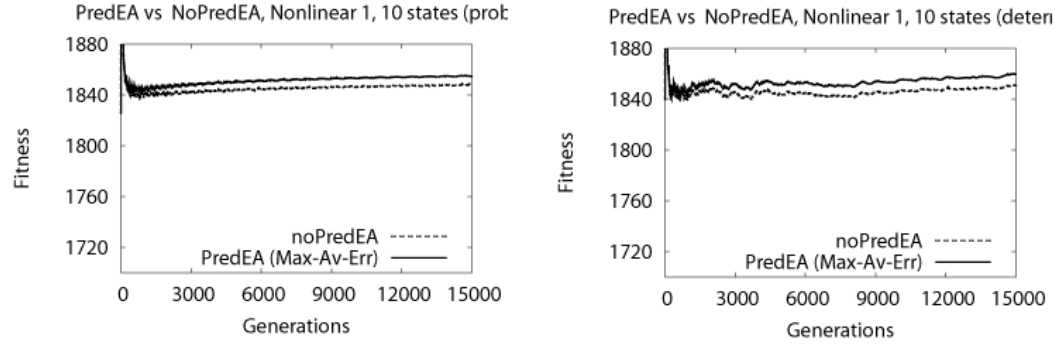

Figure 5: Comparing offline performance for the knapsack problem: noPredEA vs PredEA, 10 states, nonlinear 1

7. CONCLUSIONS AND FUTURE WORK

This paper analyzes the use of prediction in an EA dealing with dynamic environments. The accuracy of the linear prediction module incorporated in the EA is evaluated using different schemes for adjusting the value of Δ, a parameter that considers the error associated to the **LR** predictions and assures that the useful information supplied for the **MC** module is introduced into the population **before** the change happens. Four mechanisms were proposed to adjust the Δ parameter during the run. All the proposed approaches use, in different ways, the previously observed errors to learn how to adapt the value of Δ.

Analyzing the obtained results, some conclusions can be stated. First, the use of prediction mechanisms in EA dealing with dynamic environments significantly improves its performance. Second, analyzing the four proposed methods for adjusting the value of Δ in the linear prediction module, in general the Max_Av_Err strategy was the most efficient. Although in some situations the results are not clearly superior to the ones obtained using a constant value for Δ, the use of the adjustable mechanisms avoids the difficulty of choosing the correct value for Δ and, for each particular situation, this value will be found. In addition, no preliminary experimentation is necessary to decide which value to use. Third, the accuracy of predicted values for the moment of next change depends on the behavior of the change period. The proposed methods are effective if a linear or a close to linear trend is observed in the change period. When the change period follows a nonlinear behavior, as expected, the linear predictor has a poor efficacy. The prediction errors are huge and even though the Δ in some situations "covers" the prediction error, its value implies computational costs too expensive. To overcome this last conclusion, the linear prediction module should be replaced by a scheme based on **nonlinear regression**. This way, both **linear** and **nonlinear** changes can be successfully predicted. This work is already in course and the first results are very promising.

8. ACKNOWLEDGEMENTS

The work of the first author 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

[1] P. A. 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.

[2] P. A. 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*. Springer-Verlag, 2007.

[3] J. Branke. Memory Enhanced Evolutionary Algorithms for Changing Optimization Problems. In *IEEE Congress on Evolutionary Computation (CEC 1999)*, pages 1875–1882. IEEE Press, 1999.

[4] J. Branke. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, 2002.

[5] J. Branke, T. Kaußler and C. Schmidt, A Multi-Population Approach to Dynamic Optimization Problems. In I. Parmee, editor, *Adaptive Computing in Design and Manufacture (ACDM 2000)*, pages 299–308. Spriger-Verlag, 2000.

[6] J. Branke and 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, editors, *Parallel Problem Solving from Nature*, pages 253–262. Springer, 2000.

[7] H. G. Cobb. An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms having Continuous, Time-Dependent Nonstationary Environment. Technical Report TR AIC-90-001, Naval Research Laboratory 1990.

[8] D. S. Moore and G. P. McCabe. *Introduction to the Practice of Statistics (4th edition)*. Freeman and Company, 2003.

[9] Y. Jin and J. Branke. Evolutionary Optimization in uncertain Environments: a survey. *IEEE Transactions on Evolutionary Computation*, 9(3): 303–317, 2005.

[10] C. Rossi, M. Abderrahim and J. C. Díaz. Tracking Moving Optima Using Kalman-Based Predictions. *Evolutionary Computation*, 16(1): 1–30, MIT Press, 2008.

[11] A. Simões and E. Costa. Using Linear Regression to Predict Changes in Evolutionary Algorithms dealing with Dynamic Environments. Technical Report TR 2007/005, ISSN 0874-338X, CISUC, 2007.

[12] A. Simões and E. Costa. Variable-size Memory Evolutionary Algorithm to Deal with Dynamic Environments. In M. Giacobini et al., editors, *Applications of Evolutionary Computing*, volume 4448 of *Lecture Notes in Computer Science*, pages 617–626. Springer, 2007.

[13] A. Simões and E. Costa. Evaluating Prediction's Accuracy in Evolutionary Algorithms for Dynamic Environments. Technical Report TR 2008/04, ISSN 0874-338X, CISUC, 2008.

[14] A. Simões and E. Costa. Evolutionary Algorithms for Dynamic Environments: Prediction using Linear Regression and Markov Chains. In *Parallel Problem Solving from Nature (PPSN X)*, pages 306–315. Springer, 2008.

[15] 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.

[16] J. van Hemert, C. Van Hoyweghen, E. Lukshandl and K. Verbeeck. A Futurist Approach to Dynamic Environments. In *GECCO EvoDOP Workshop*, pages 35–38, 2001.

[17] S. Yang. Explicit Memory Schemes for Evolutionary Algorithms in Dynamic Environments. In S. Yan, Y-S. Ong and Y. Jin, editors, *Evolutionary Computation in Dynamic and Uncertain Environments*, pages 3–28. Springer-Verlag, 2007.

[18] S. Yang and X. Yao. Experimental Study on Population-Based Incremental Learning Algorithms for Dynamic Optimization problems. *Soft Computing*, 9(11): 815–834, 2005.