Using Linear Regression to Predict Changes in Evolutionary Algorithms dealing with Dynamic Environments

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 2007/005 - ISSN 0874-338X **April 2007**

Abstract. Many real-word problems change over time and usually, the moment when next change will happen is unknown. Evolutionary Algorithms have been widely used to deal with changing environments and the algorithm is constantly monitoring for alterations and just after detecting one some action is taken. Nevertheless, some of the studied environments are characterized by the periodicity of the change. In these cases it is possible to predict when the next change will occur and start using some mechanisms before the change take place. In this report we carried out an investigation in cyclic changing environments with periodic changes, using linear regression to predict when next change will occur. Based on the predicted value, the algorithm starts preparing the population for the near change. This idea is tested in a memory-based EA using a population and memory of variable sizes previously studied with considerable success. We assume that the predicted moment can have a small error. Before the change occurs two different actions can be taken in order to avoid the decrease of the algorithm's performance under the new conditions. The results show that prediction is useful for cyclic environments when the period between changes is small and when more different states can appear in the environment.

Keywords: Evolutionary Algorithms, Dynamic Environments, Prediction, Memory, Statistics, Linear Regression

1 Introduction

Most real-world problems change over time and traditional Evolutionary Algorithms (EAs) are not suited to solve them. In these cases, some improvements have been introduced in EAs, such as the addition of memory or the use of strategies to promote the population's diversity.
The main purpose of the memory-based approaches is to gather information from the past that can be useful in the future. The rationale behind promoting the population's diversity is to avoid the convergence of the complete population towards a point of the search space, which may difficult its readaptation if a change happens. Nevertheless, most of evolutionary-based approaches used in changing optimization problems wait for the observation of an environmental alteration to decide what to do: use the information of memory [1], [7], [9], [12], [19], or change the rate of mutation in order to increase population's diversity [4], for instance. Usually, the result is a reduction on the algorithm's performance after a change, taking some time to readapt and find the new optimum.
When the environment is chaotic or completely random, we should not expect the EA to readapt to new situations immediately. But, assuming that the environment has some predictability, such as periodic changes or a certain repeated pattern, it is reasonable to think that using information from the past it is possible to estimate future situations.

Recently, several studies concerning anticipation in changing environments using EAs have been proposed. The main goal of these approaches is to estimate future situations and so decide the algorithm's behavior in the present. Since that information about the future is not available, this information is attained through learning from past situations.
Stroud (2001) used a Kalman-Extended Genetic Algorithm in which a Kalman filter is used to determine when to generate a new individual, when to re-evaluate an existing individual and which one to re-evaluate [16].
Van Hemert et al. (2001) introduced an EA with a meta-learner to estimate the direction of change for a single discrete time step. This approach uses two populations, one that searches the current optimum and another which uses the best individuals in the past to predict the future best value [17].
Bosman (2005) proposed an algorithmic framework integrating evolutionary computation with machine learning and statistical learning techniques to estimate future situations [2]. An extension of this work, applied to the dynamic vehicle routing problem was proposed in [3].
In this work we explore the application of a general idea for predicting the generation when next change will occur. This way, we can introduce useful information in the population before the change occurs and increase the algorithm's adaptability. When the environments change periodically or with a fixed pattern, next changes can be estimated from information observed in previous changes. This prediction can be made with minimum error using a linear regression method.
Having estimation for the next change, we can prepare the algorithm to the future, introducing helpful information in the population, and so reduce the effects of the environmental modifications.
The remaining text is organized as follows: section 2 describes previous work that will be used to test the proposed ideas, section 3 details how the linear regression works and how we used it to predict the generation of next change. The experimental setup is explained in section 4. In section 5 we show the obtained results concerning algorithms efficiency and the accuracy of the predicted values. Finally, in section 6 we state the main conclusions and ideas for future work.

2. Previous Work

2.1. Variable-size Memory Evolutionary Algorithm

In [13] was proposed an EA called VMEA – Variable-size Memory Evolutionary Algorithm, to deal with dynamic environments. This algorithm uses a memory, responsible for storing good individuals of the evolved population in several points of the search process. The innovative aspect of the algorithm is that the two populations - search and memory – have variable sizes that can change between fixed boundaries. The global resources (the sum of the memory and population individuals) are limited up to an established maximum but it's the algorithm which decides the sizes of the memory and search population at any moment. The memory is updated from time to time and if the established limits are not reached, the best individual of the current population is stored in the memory. If there is no room to save this new solution, then the best individual of the current population is introduced replacing a memory individual chosen accordingly to the replacing scheme. The memory is evaluated every generation and a change is detected if at least one individual in the memory changes its fitness. The memory is updated following a dynamic time pattern as follows: After a memory update in generation t, next memory updating will be decided by generating a random integer R in the interval [5, 10] and the next memory update will take place in generation $t+R$. Notice that, besides the environmental changes occur periodically every r generation, the algorithm has no information about when next change will occur and so, it is necessary to make its detection at every time step. After detecting an environmental modification the best individual is retrieved from the memory and introduced into the population. In the case of either the population's size or the sum of the two populations has reached the allowed maximum, the best individual in memory replaces the worst one in the current population**.**

2.2. Replacing Strategies

The performance of the VMEA was compared using different replacing strategies and its efficiency was strongly increased using a method that replaces the worst individual stored in the memory since the last change [14]. This method was called *generational* and will be used in the experimental study of this work.

2.3. Bacterial Conjugation

In biology, bacterial conjugation is the transfer of genetic material between bacteria through cell-to-cell contact. Sometimes bacterial conjugation is regarded as the bacterial equivalent of sexual reproduction or mating, but in fact, it is merely the transfer of genetic information from a donor to a recipient cell [11]. Computational conjugation was proposed by [6] and [15]. VMEA was also tested in several dynamic optimization problems using this genetic operator as an alternative to crossover. The authors applied conjugation after selecting the individuals to the mating pool, using the idea of donor-recipient genetic transfer. The exhaustive experimental study comparing this operator with crossover in different benchmark problems showed that, in general, conjugation allows VMEA to achieve improved results [13], [14]. Conjugation will be used in this work as the main genetic operator of the EA.

3. Predicting with Linear Regression

3.1. Statistical Model for 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 is assumed that they all lie on a line when plotted against *x* [10]. The equation of that line is:

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

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 about these means and it's assumed that this variation, measured by the standard deviation, is the same for all the values of *x*.

Linear regression allows inferences not only for samples for which the data is known, but also for those corresponding to x's not present in the data. 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:

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

The slope b_1 is given by:

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

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 equations (5).

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

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

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 [10] :

$$\hat{y} = b_0 + b_1 * x \qquad (6)$$

In the next section we will show an example.

Note that linear regression does not test whether 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, more inference methods are needed.

3.2. Predicting Next Change

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.

In our implementation, the first two changes of the environment are stored without prediction. After that, it is possible to start predicting the next change based on the previous observations.

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:
$b_1 = 1 * \frac{50}{1} = 50$ and $b_0 = 100 - b_1 * 2 = 100 - 50 * 2 = 0$
Using equation (7) 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.

3.3. Prediction in VMEA

We used the previously described VMEA to test these ideas. A module of prediction was included in the algorithm. This unit is responsible for keeping track of 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 predicted. It is assumed that the predicted value may not be exact and small error is considered (*error*). This error will be used to define an interval where the change probably will occur: [$next_{gen}$–*error*, $next_{gen}$+ *error*].
So, when the current generation reaches the value $next_{gen}$–*error* the algorithm starts preparing the eventuality of an environmental alteration. The actions taken can be one of following two:
(1) when algorithm reaches generation $next_{gen}$–*error*, we merge the actual population to the memory, allowing the population size to increase up to the standard limit. Then, when a change occurs or when the generation $next_{gen}$+*error* is reached, the worst individuals of the population are removed until the previous standard limit is assured (we will call this method ***upd1***).
(2) the second approach is similar to the previous one, but ***M/2*** of the individuals merged with the population are new individuals, and the remaining are the best ***M/2*** memory individuals (***M*** is the actual memory size). The new individuals are created applying the conjugation genetic operator to the individuals of memory. Once again, the population size is allowed to increase during the same period referred before, or until the change occurs (we will call this method ***upd2***).

4. Experimental Design

4.1. Dynamic Test Environments

The dynamic environments to test our approach were created using Yang's Dynamic Optimization Problems (DOP) generator [18]. This generator allows constructing different dynamic environments from any binary-encoded stationary function using the bitwise exclusive-or (XOR) operator. The basic idea of the generator is to perform the operation $x \oplus M$ to an individual *x*, where $\oplus$ is the bitwise XOR operator and *M* a binary mask previously generated. Then, the resulting individual is evaluated to obtain its fitness value. If a change happens at generation *t*, then we have $f(x, t+1) = f(x \oplus M, t)$. Using the DOP generator the characteristics of the change are controlled by two parameters: the speed of the change, *r*, which is the number of generations between two changes, and the magnitude of the change,ρ that consists in the ratio of ones in the mask *M*. The more ones in the mask the more severe is the change. The DOP generator also allows constructing problems where the changes can be cyclic, cyclic with noise or non-cyclic. In the first case, several masks are generated according to the ρ parameter and are consecutively applied when a change occurs. It is thus possible that previous environments reappear later. In the second case noise is added by mutating some bits in the mask with a small probability. In the third case, the mask applied to the individuals is always randomly generated every time we change the environment.
In this work we constructed 16 *cyclic* DOPs, setting the parameter *r* to 10, 50, 100 and 200. The ratio ρ was set to different values in order to test different levels of change: 0.1 (a light shifting) 0.2, 0.5 and 1.0 (severe change). We also introduced some changes in Yang's DOP in order to have the period between changes controlled by a pattern, instead of the fixed period size *r*. So, if we define a pattern 50-60-70, for instance, it means that the changes will occur after 50 generations, then after 60 generations, then after 70 generations, and this pattern is repeated the required number of times.

4.2. Knapsack Problem

The benchmark used was the 0/1 dynamic knapsack. The knapsack problem is a NP-complete combinatorial optimization problem often used as benchmark. It consists in selecting a number of items to a knapsack with limited capacity. Each item has a value (v_i) and a weight (w_i) and the objective is to choose the items that maximize the total value, without exceeding the capacity of the bag:

$$\max v(x) = \sum_{i=1}^{m} v_i x_i \qquad (7)$$

subject to the weight constraint:

$$\sum_{i=1}^{m} w_i x_i \leq C \qquad (8)$$

We used a knapsack problem with 100 items using strongly correlated sets of randomly generated data constructed in the following way ([8], [18]):

w_i = uniformly random integer [1, 50] (9)

v_i = w_i + uniformly random integer [1, 5] (10)

$$C = 0.6 \times \sum_{i=1}^{100} w_i \qquad (11)$$

The fitness of an individual is equal to the sum of the values of the selected items, if the weight limit is not reached. If too many items are selected, then the fitness is penalized in order to ensure that invalid individuals are distinguished from the valid ones. The fitness function is defined as follows:

$$f(x) = \begin{cases} \sum_{i=1}^{100} v_i x_i, if \sum_{i=1}^{100} w_i x_i \leq C \\ 10^{-10} \times \left[\sum_{i=1}^{100} w_i - \sum_{i=1}^{100} w_i x_i \right], otherwise \end{cases} \qquad (12)$$

4.3. Experimental Setup

4.3.1. Algorithms' parameters

Previous work ([13], [14]) compared VMEA with other evolutionary algorithms: the random immigrants' algorithm (RIGA) [5], the memory- based immigrants GA (MIGA) [18] and the memory-enhanced GA (MEGA) [18]. The results proved that VMEA, generally outperformed the other approaches. The best results were achieved when VMEA used the *generational* replacing strategy and the conjugation operator [14].

We will compare VMEA enhanced with the prediction unit, using bacterial conjugation and the generational replacing method as well. This VMEA was run using the two possible actions formerly described: *upd1* (*VMEAp_upd1*) and *upd2* (*VMEAp_upd1*).

The algorithms' parameters were set as follows: generational replacement with elitism of size one, tournament selection with tournament of size two, conjugation with probability p_c=0.7 and mutation applied with probability p_m=0.01. VMEA using prediction will start with a population of 100 individuals and a memory of 10 individuals. The sum of the two populations cannot go beyond 150 individuals, except in the intervals described in section 3.3. In this interval the allowed maximum raised to 200 individuals. VMEA was run as described in [13], using initial sizes for population and memory of 125 and 25 individuals, respectively. The sum of the two populations could not surpass the 200 individuals. The assumed error for the predicted values in *VMEA_p*, to identify the interval [$next_{gen}$–$error$, $next_{gen}$+$error$], was 5 generations.

For each experiment of an algorithm, 30 runs were executed and the number of environmental changes was: (1) in environments changing every *r* generations: 200 when *r* = 10 (2000 generations), 80 when *r* = 50 (4000 generations) and 40 when *r* = 100 and 200 (4000 and 8000 generations, respectively); (2) using the patterned periodic changes, the number of generations was computed based in 200 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:

$$Foverall = \frac{1}{G} \sum_{i=1}^{G} \left[\frac{1}{N} \sum_{j=1}^{N} F_{bestij} \right] \qquad \textbf{(13)}$$

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

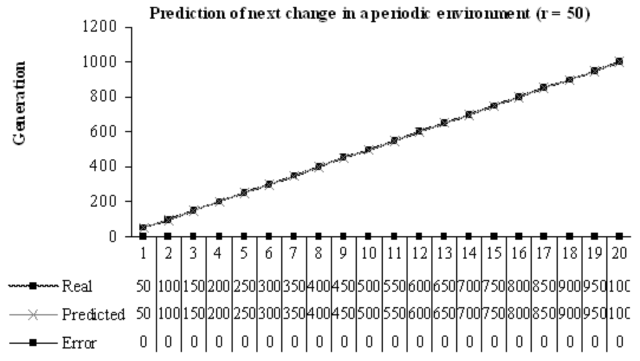
5. Results

5.1. Accuracy of the Prediction Method

When changes are defined by the parameter ***r*** and occur every ***r*** generations, the statistical model using linear regression gives exact predictions, since the observed values lie down in the regression line. Figure 1 shows the predicted values using periodic changes of size 50 ($r = 50$).

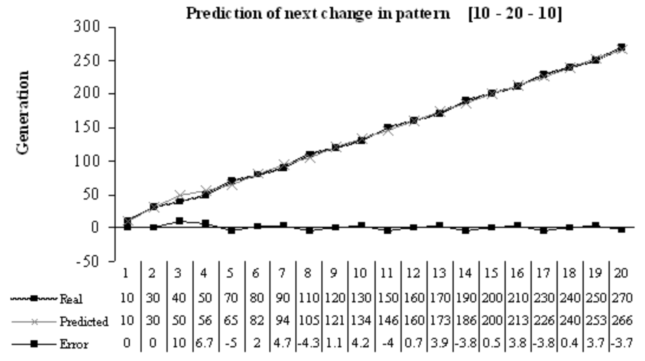
Using a pattern to generate the periodicity of the change, we can have situations where the predicted values are not precise. In these cases, there is an associated error that decreases over time. Nevertheless this decreasing is very slow. For the 20 first changes observed using the pattern 10-20-10 and plotted in Figure 2, we can see that the prediction model was not exact. In this case, the interval assumed in our implementation (5 generations) was enough to cover that error, and in this case, the insertion of individuals in the population was always made *before* the change occurs.

For a different pattern, 50-60-70, the same observations were made. The predicted values for the next change were not precise, the error decreases during the time, and the used interval $[next_{gen}-error,\ next_{gen}+error]$ was able to guarantee, most of the times, that the change was detected before it happens. Figure 3 shows the predicted values using this pattern change.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Real	50	100	150	200	250	300	350	400	450	500	550	600	650	700	750	800	850	900	950	100
Predicted	50	100	150	200	250	300	350	400	450	500	550	600	650	700	750	800	850	900	950	100
Error	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

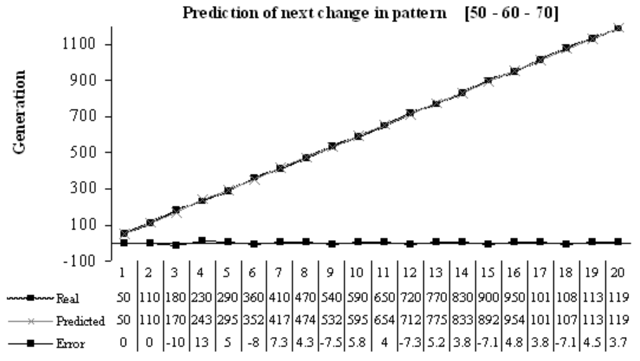
Figure 1 – Prediction in periodic environments with a fixed period (r = 50)



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Real	10	30	40	50	70	80	90	110	120	130	150	160	170	190	200	210	230	240	250	270
Predicted	10	30	50	56	65	82	94	105	121	134	146	160	173	186	200	213	226	240	253	266
Error	0	0	10	6.7	-5	2	4.7	-4.3	1.1	4.2	-4	0.7	3.9	-3.8	0.5	3.8	-3.8	0.4	3.7	-3.7

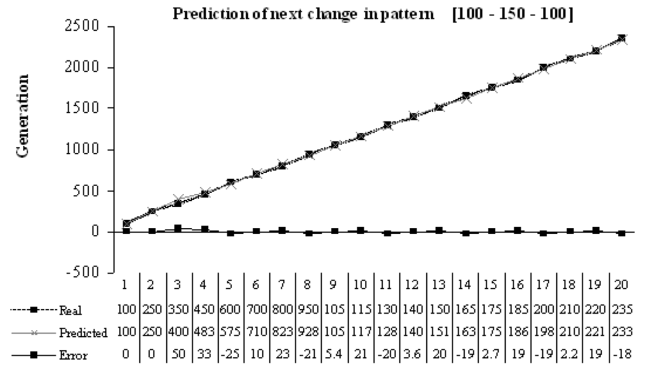
Figure 2 - Prediction in periodic environments with a pattern change (10 – 20 – 10)

Using a different pattern, 100-150 the used prediction model had worse predicted values. The errors associated to the estimated values were in some cases superior to 5, and our module using an interval margin of 5 generations didn't deal with these changes before they happen. Figure 4 shows the expected values and the resultant error. For cases like this it is necessary to introduce in the prediction correction mechanisms based on the previous measured errors.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Real	50	110	180	230	290	360	410	470	540	590	650	720	770	830	900	950	101	108	113	119
Predicted	50	110	170	243	295	352	417	474	532	595	654	712	775	833	892	954	101	107	113	119
Error	0	0	-10	13	5	-8	7.3	4.3	-7.5	5.8	4	-7.3	5.2	3.8	-7.1	4.8	3.8	-7.1	4.5	3.7

Figure 3 - Prediction in periodic environments with a pattern change (50 – 60 – 70)



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Real	100	250	350	450	600	700	800	950	105	115	130	140	150	165	175	185	200	210	220	235
Predicted	100	250	400	483	575	710	823	928	105	117	128	140	151	163	175	186	198	210	221	233
Error	0	0	50	33	-25	10	23	-21	5.4	21	-20	3.6	20	-19	2.7	19	-19	2.2	19	-18

Figure 4 - Prediction in periodic environments with a pattern change (100 – 150 – 100)

5.2. Performance of the Algorithms

In this section we will show the results obtained by the VMEA without the prediction module (*VMEA)* and by VMEA using prediction (*VMEAp_upd1* and *VMEAp_upd2*). The results are exposed in two separated sections: the first for changes in every ***r*** generation and another for changes determined by a pattern such as described in section 4.1.

The statistical results comparing the algorithms are reported in tables 1 and 2. We used paired one-tailed t-test at a 0.01 level of significance. The notation used in tables 1 and 2, 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.

5.1.1. Results in cyclic environments with a fixed change period

The global results given by equation (13) are plotted in Figure 5.

We can see that for small change periods, the prediction module introduced in the algorithm increased its performance. For larger change periods, all the algorithms had similar behavior, except when the change ratio was higher.

Table 1. T-test results comparing VMEAs in cyclic environments with fixed change period = 10, 50, 100, 200

		Change ratio			
T-test results	***Chg. period***	**0.1**	**0.2**	**0.5**	**1.0**
VMEA - VMEAp_upd1	10	--	--	--	--
VMEA - VMEAp_upd2		--	--	--	--
VMEAp_upd1 - VMEAp_upd2		-	-	--	+
VMEA - VMEAp_upd1	50	--	--	-	--
VMEA - VMEAp_upd2		-	--	--	--
VMEAp_upd1 - VMEAp_upd2		++	-	--	-
VMEA - VMEAp_upd1	100	+	++	+	--
VMEA - VMEAp_upd2		-	-	--	--
VMEAp_upd1 - VMEAp_upd2		-	--	--	-
VMEA - VMEAp_upd1	200	+	-	+	--
VMEA - VMEAp_upd2		-	+	--	--
VMEAp_upd1 - VMEAp_upd2		-	++	--	-

In this case, the introduction of the prediction module allowed faster adaptation of the algorithm to the observed alterations. Table 1 confirms these observations. We see that VMEA globally achieved the worst performances. In table 1 we can also see that the method *upd2* used to prepare the population before the change was in general better than the *upd1* method.

Figures 6 and 7 show some examples of how the VMEAs' performance evolved through the time. It is clear that for smaller change periods the prediction unit was very helpful and increased the algorithm's performance. For larger change periods (200) the improvements are visible for inferior change ratios, because in these cases the number of different states is superior and the memory in certain periods wasn't enough to allow continuous adaptability.

5.1.2. Results in periodic environments following a pattern change

In the environments with periodic changes based on a repeated pattern the conclusions were similar to the previously described. In the patterns with smaller periods, 10-20-10 and 50-60-70, the VMEA using prediction obtained the best scores.

In the pattern with larger periods, 100-150-100, the improvements introduced by the prediction method were not so visible. Table 2 and Figure 8 show the global results comparing all the algorithms in the three tested patterns. In all the analyzed cases the *upd2* and *upd1* methods obtained analogous results. Figures 9 and 10 show some examples of the dynamics of the algorithms along the time. All the conclusions stated before are once again visible in these plots.

Table 2. T-test results comparing VMEAs in cyclic environments with changes following a fixed pattern

		Change ratio			
T-test results	***pattern***	**0.1**	**0.2**	**0.5**	**1.0**
VMEA - VMEAp_upd1		--	+	--	--
VMEA - VMEAp_upd2	10-20-10	--	-	--	--
VMEAp_upd1 - VMEAp_upd2		+	-	-	+
VMEA - VMEAp_upd1		--	--	--	--
VMEA - VMEAp_upd2	50-60-70	--	--	--	-
VMEAp_upd1 - VMEAp_upd2		+	-	+	+
VMEA - VMEAp_upd1		--	--	+	+
VMEA - VMEAp_upd2	100-150-100	--	--	++	+
VMEAp_upd1 - VMEAp_upd2		-	+	+	+

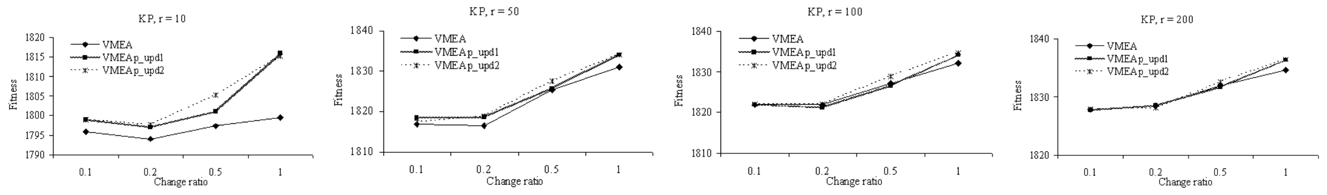


Figure 5. Overall performance of VMEAs in periodic environments: r=10; r=50; r=100; r=200

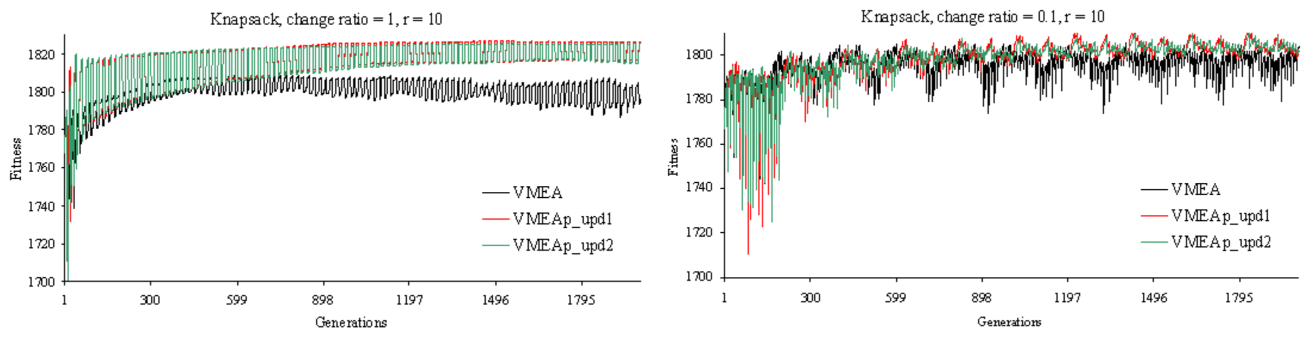


Figure 6. VMEAs performance in cyclic environments with change period 10, ρ=1 and ρ=0.1

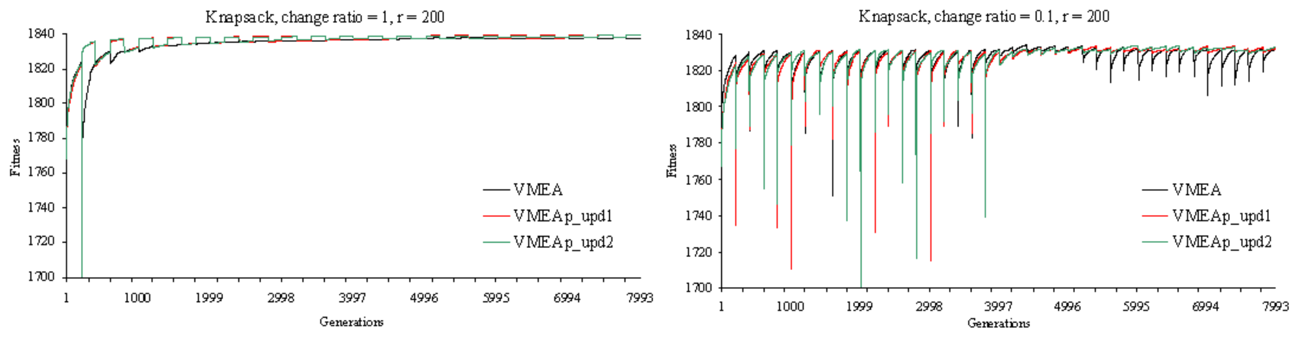


Figure 7. VMEAs performance in cyclic environments with change period 200, ρ=1 and ρ=0.

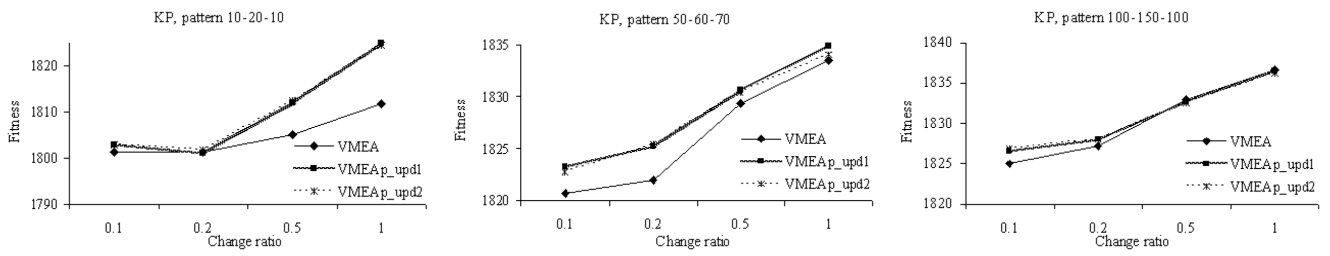


Figure 8. Overall performance of VMEAs in cyclic environments with changes following a fixed pattern

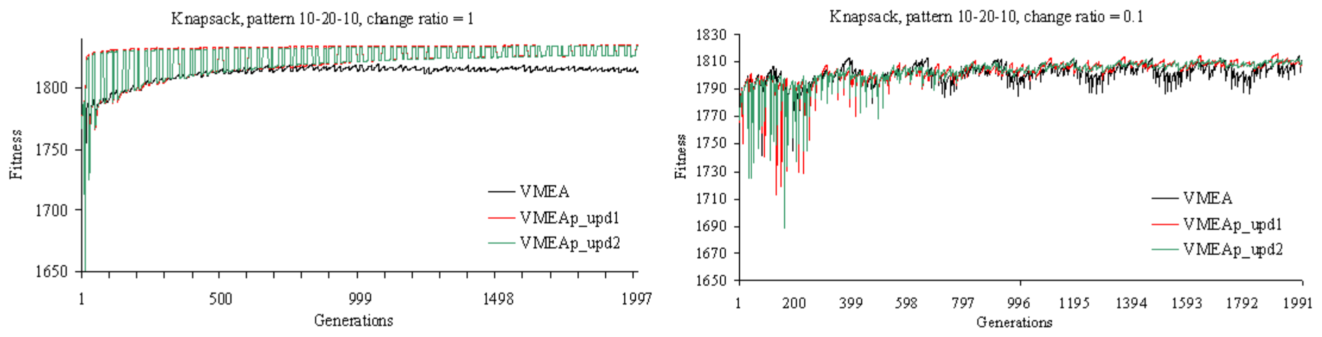


Figure 9. VMEAs performance in cyclic environments with change period following a fixed pattern 10-20-10, ρ=1 and ρ=0.1

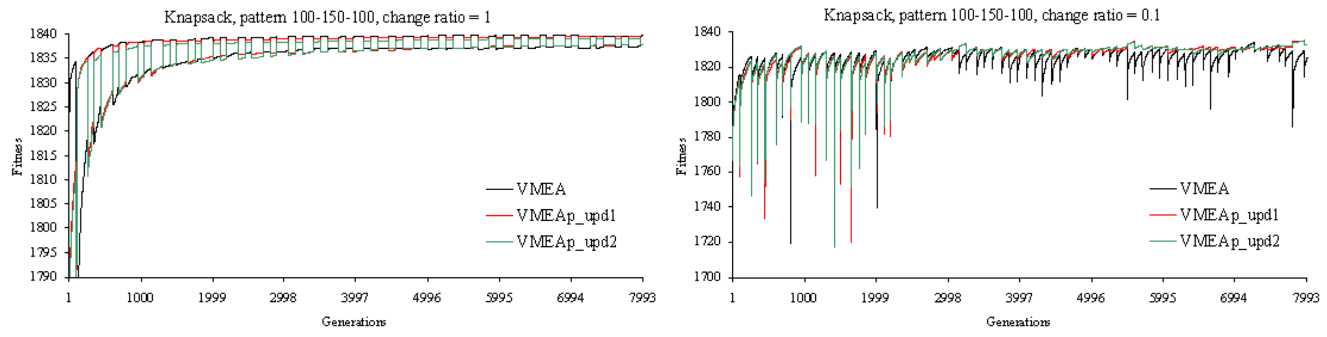


Figure 10. VMEAs performance in cyclic environments with change period following a fixed pattern 100-150-100, ρ=1 and ρ=0.1

5.1.3. Is prediction really useful?

To see if the prediction unit was responsible for the observed improvements, we run VMEA without the prediction module, but using the upd1 and upd2 methods after the change occur. So, instead of using the scheme of the original VMEA (retrieve from the memory the best individual and introduce it into the population), we allowed the global number of individuals to increase up to 200 in the moment after the change, merging the memory with the population (upd1) or creating some new individuals applying conjugation in the memory and then merging this individuals with the population and with part of the memory (upd2).

We observed that using these schemes the VMEA's performance was increased, when compared with the original VMEA, but the algorithms using the prediction unit were in fact the most proficient, Figures 11 and 12 show some of the results comparing VMEA with no prediction and VMEA without prediction and using the upd1 and upd2 schemes (applied after the change). So, it seems that the prediction unit was in fact responsible for the observed improvements. Preparing the population before the change happens proved to be helpful to the readaptation of the algorithm. Currently we are trying to use different methods before the change occur, such us, estimating also the direction of the change and introducing individuals that probably will be the fittest after the next change happens.

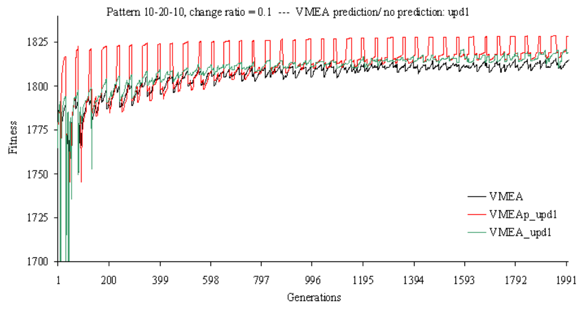


Figure 11. Comparing VMEA with *VMEAp_upd1* and VMEA without the prediction unit using the *upd1* method after the change: pattern 10-20-10.

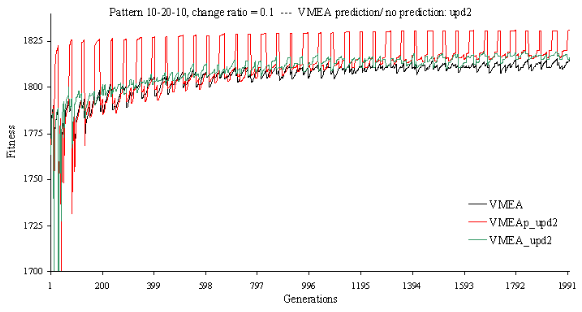


Figure 12. Comparing VMEA with *VMEAp_upd2* and VMEA without the prediction unit using the *upd2* method after the change: pattern 10-20-10.

6. Conclusions and Future Work

A prediction method was applied using linear regression to estimate the moment when next change will occur. This prediction unit was applied in an EA to deal with dynamic environments. Based on the predicted moment of future changes we tried to prepare the population *before* the alteration occurs. Since the prediction could not be precise, we assume a possible error of five generations. When next change was close, we used two methods for changing the population *before* the change and minimize the consequences in the EA's efficiency: merge population and memory or merge the population with part of the memory and with some new created individuals. We observed that the proposed schemes enhanced the EA's performance in cyclic environments with smaller change periods. It was observed a continuous improving of the EA's adaptability along time. Currently, we are extending this work, introducing methods for estimate the direction of the change and try to be more exact in the information introduced in the population before the change happens. Mechanisms to reduce the error in the predicted values are also being tested. We are also interested in use this approach and compare it with other methods based on estimation.

7. Acknowledgment

The work 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.

8. References

[1] J. Branke. Memory Enhanced Evolutionary Algorithms for Changing Optimization Problems. Proc. of Congress on Evolutionary. Computation, pp. 1875-1882, IEEE, 1999.

[2] 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.

[3] P. A. N. Bosman, H. La Poutré. Computationally Intelligent Online Dynamic Vehicle Routing by Explicit Loa Prediction in Evolutionary Algorithm. Proceedings of Parallel Problem Solving from Nature, Lecture Notes in Computer Science 4193, pp. 312-321, Springer Verlag Berlin, 2006.

[4] 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.

[5] J. J. Grefenstette. Genetic Algorithms for Changing Environments. Proc. of the 2nd Int. Conf. Parallel Problem Solving from Nature 2, pp. 137-144, North-Holland, 1992.

[6] I. Harvey. The Microbial Genetic Algorithm. Unpublished, 1996.

[7] S. J. Louis and Z. Xu. Genetic algorithms for open shop scheduling and re-scheduling. In M. E. Cohen and D. L. Hudson, editors, ISCA Eleventh International Conference on Computers and their Applications, pp. 99-102, 1996.

[8] Z. Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. 3rd edition, Springer Verlag Berlin, 1999.

[9] N. Mori, H. Kita, and Y. Nishikawa. Adaptation to a changing environment by means of the thermodynamical genetic algorithm. In H.-M. Voigt, editor, Parallel Problem Solving from Nature, 1141 in LNCS, pp. 513-522. Springer Verlag Berlin, 1996.

[10] D. S. Moore and G. P. McCabe. Introduction to the Practice of Statistics. Freeman and Company, 4th Edition, 2003.

[11] P. J. Russell. Genetics. 5th edition, Addison-Wesley, 1998.

[12] A. Simões and 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.

[13] A. Simões and 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.

[14] A. Simões and E. Costa. Improving Memory-based Evolutionary Algorithms in Changing Environments. In CISUC Technical Report TR2007/004, ISSN 0874-338X, March 2007.

[15] P. Smith. Conjugation: A Bacterially Inspired Form of Genetic. Late Breaking Papers at the Genetic Programming 1996 Conf., Stanford Univ., July 28-31, 1996.

[16] 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.

[17] J. van Hemert, C. Van Hoyweghen, E. Lukshandl, and K. Verbeeck. A Futurist Approach to Dynamic Environments. GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems, pages 35-38, 2001.

[18] S. Yang. Experimental Study on Population-Based Incremental Learning Algorithms for Dynamic Optimization problems. Soft Computing, vol. 9, nº 11, pp. 815-834, 2005.

[19] S. Yang. Memory-Based Immigrants for Genetic Algorithms in Dynamic Environments. Proc. of the 2005 Genetic and Evolutionary Computation Conference, Vol. 2, pp. 1115-1122, ACM Press, 2005.