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

Anabela Simões[1,2] and Ernesto Costa[2]

[1] Department of Informatics and Systems Engineering, Coimbra Polytechnic
[2] Centre of Informatics and Systems of the University of Coimbra
abs@isec.pt, ernesto@dei.uc.pt

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. Different types of environmental changes were studied. A memory-based evolutionary algorithm empowered by these two techniques was successfully applied to several instances of the dynamic bit matching problem.

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, the maintenance of population's diversity or the use of several populations. See ([1],[2]) for a review.

When the environment is dynamic, in some cases 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. 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. We assume that the reader is familiar with the concepts of

G. Rudolph et al. (Eds.): PPSN X, LNCS 5199, pp. 306–315, 2008.

linear regression and Markov chains. Also, due to lack of space we only present a small part of the results we obtained. The interested reader should consult [3]. 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. In section 4 we present the experimental setup used to test the proposed ideas. Experimental results are summarized in section 5. 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 et al. ([4]) 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.

Stroud ([5]) 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. ([6]) 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.

Bosman ([7], [8]) 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.

Linear regression was used in [9] to improve local convergence in dynamic problems.

3 System's Overview

In this section we will detail each component of the proposed system, called **PredEA**. The major components of the complete architecture are the following: (1) a standard evolutionary algorithm; (2) memory of past good individuals; (3) Markov chain model module; (4) linear regression module.

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 completely **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 module is generation g, at generation $g - \Delta$, the Markov model ([10]) is used to predict the next possible 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 situation, the new state is included in the model and, again, the transition matrix is updated. Notice that, during the earlier stages of the simulation prediction is difficult, because 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 now.

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, which we do 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.

Memory. Memory is used to store best individuals of the current population. It starts empty and has a limited size (20 individuals). An individual is stored into memory in two situations: (1) if the environment changed in the meantime and no individual related to this new environment was previously stored; (2) if an individual already exists in memory for the current environment, but it is worst than the 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 ([11]). 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 [12] 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.

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, a 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 ones. Again we stress that all this information is *unknown* to the algorithm, which works with a Markov model that it builds dynamically. Ideally, after some generations and environmental changes our algorithm will construct a Markov model identical to the hidden one. From then on, the next state(s) can be correctly predicted making possible the introduction of important information **before** the effective change, allowing the continuous adaptation of the EA to the new conditions. Our algorithm starts with its transition matrix filled with zeros. Each time a transition is detected, say from state i to another state j, the probability values involving state i and all the other states j are changed to take into account the number of times the environment moved from i to j.

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 the regression line is updated.

***PredEA* Pseudocode.** Now that we have described the different components we can present the pseudocode of **PredEA**.

```
PredEA(max, markov, initial-state)
 1  Randomly create initial population
 2  Create empty memory
 3  Create the transition matrix with max states filled with zeros
 4  repeat
 5          Evaluate population
 6          Evaluate memory
 7          if Is time to update memory
 8            then Store the best individual
 9                 Set next time to update memory
10          if An environmental change happens
11            then Store performance measures
12                 Update the linear regression line
13                 Predict g (next-change)
14                 Update the algorithm's Markov transition matrix
15          if g (next-change) is close (as defined by g − Δ)
16            then 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
```

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.

4 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 [2]. Memory is updated using the same time method, but instead of the replacing strategy used by [2] 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). The benchmark used was the *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.

Experimental Setup. In the Table 4 we summarize the EA's settings used in our experiments.

For each experiment, 30 runs were executed and 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. The results were statistically validated ([3]).

Usually, in papers related with the algorithms' performance on changing environments (e.g. [11], [13], [2]), 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.

The number of different states (templates) used in the experimentation was 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

Table 1. Parameters' 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 1
stop criterion	number of generations necessary for 200 environmental changes
goal	maximize matching with template
Δ	{5,10,25}

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 denoted by $P_{ij} <> 1$). 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 called *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. This means that we tested **80** different situations. Only partial results will be shown (see [3]).

5 Results

Accuracy of Predicted Values using Linear Regression. When changes occur every r generations, linear regression gives correct predictions, since all the observed values are on the regression line. Using a pattern to generate the periodicity of the change, we may have situations where the predicted values are not precise. In the cases of patterns 5-10-5 and 10-20-10, there is an associated error that slowly decreases over time. In these cases, the Δ constant assumed in our implementation (5 generations), was sufficient to reduce to zero that error, and the insertion of individuals in the population was always made before the change occurs. That is not true for the patterns 50-60-70 and 100-150-100, and we had to use an increased value of Δ to avoid a decrease in performance of our algorithm **PredEA**.

PredEA *versus* noPredEA. Results obtained for cyclic-periodic environments (changing every r generations) are given in Table 2 . The best scores are marked in bold.

We used a paired one-tailed **t-test** at a 0.01 level of significance to compare the two algorithms. The results obtained with **PredEA** were always statistically significantly 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 1 shows the typical behavior 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 unstable with a decrease in fitness every time there is an environmental change. This is the 'learning phase' when the algorithm is building the Markov chain model and its transition matrix.

Table 2. Global Results for cyclic-periodic environments and change period **r**

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

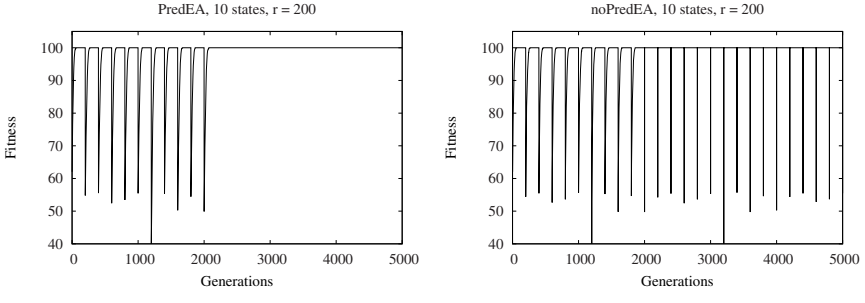

Fig. 1. PredEA versus **NoPredEA**, $r = 200$, 10 states, deterministic

After that initial phase the predictions are correctly made by the two predictor modules and the **PredEA**'s performance reaches an 'equilibrium phase'. On the other hand, the behaviour of **noPredEA** is always very unstable. After a change, we observe a decrease of the fitness and only after retrieving information from memory, which is made immediately after a change happens, the EA recovers. Similar results were obtained for cyclic-pattern environments. For the patterns 50-60-70 and 100-150-100, the value of 5 for Δ constant was not a good choice for the prediction modules. For these two situations, we repeated the experiments adjusting the constant value to 10 and 25, respectively. That way we always anticipated the actual moment of change. The results 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. Table 3 shows the global results obtained in all the experiments carried out. Best results are marked with bold. The statistical analysis of the obtained results can be found in [3].

Table 3. Global Results for cyclic-pattern environments 5-10-5, 10-20-10, 50-60-70 and 100-150-100

		Number of States				
Pattern	**Algorithm**	**3**	**5**	**10**	**20**	**50**
5-10-5	PredEAΔ5 ($P_{ij} = 1$)	**97.49**	**97.27**	**97.50**	**97.65**	**95.63**
	PredEAΔ5($P_{ij} <> 1$)	**97.10**	**96.13**	**96.77**	**96.79**	**94.60**
	NoPredEA ($P_{ij} = 1$)	91.98	90.35	85.89	79.36	72.62
	NoPredEA ($P_{ij} <> 1$)	93.18	90.58	86.01	79.90	73.57
10-20-10	PredEAΔ5 ($P_{ij} = 1$)	**98.36**	**98.11**	**97.95**	**97.46**	**94.64**
	PredEAΔ5 ($P_{ij} <> 1$)	**98.24**	**97.49**	**97.12**	**96.29**	**93.01**
	NoPredEA ($P_{ij} = 1$)	91.26	92.25	88.13	80.97	73.40
	NoPredEA ($P_{ij} <> 1$)	94.71	91.52	87.89	81.67	74.51
50-60-70	PredEAΔ5 ($P_{ij} = 1$))	99.54	99.37	98.82	97.04	94.60
	PredEAΔ5 ($P_{ij} <> 1$))	99.52	99.34	98.79	97.24	94.38
	PredEAΔ10 ($P_{ij} = 1$)	**99.80**	**99.65**	**99.31**	**98.60**	**96.52**
	PredEAΔ10 ($P_{ij} <> 1$)	**99.79**	**99.61**	**99.15**	**98.23**	**95.92**
	NoPredEA ($P_{ij} = 1$)	99.44	99.16	98.64	96.45	94.54
	NoPredEA ($P_{ij} <> 1$)	99.44	99.17	98.58	97.02	95.13
100-150-100	PredEAΔ5 ($P_{ij} = 1$)	99.64	99.52	99.20	98.07	95.94
	PredEAΔ5 ($P_{ij} <> 1$)	99.65	99.53	99.23	98.17	96.45
	PredEAΔ25 ($P_{ij} = 1$)	**99.89**	**99.79**	**99.65**	**99.29**	**98.25**
	PredEAΔ25 ($P_{ij} <> 1$)	**99.89**	**99.78**	**99.55**	**99.10**	**97.95**
	NoPredEA ($P_{ij} = 1$)	99.71	99.59	99.27	98.31	97.24
	NoPredEA ($P_{ij} <> 1$)	99.71	99.59	99.29	98.52	97.56

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 2 shows the typical behavior 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 we observe the presence of the learning and equilibrium phases when using **PredEA**. **noPredEA** behaves in the same way as in previous cases. In all situations, except when the **PredEA** has a Δ value of 5, the pattern

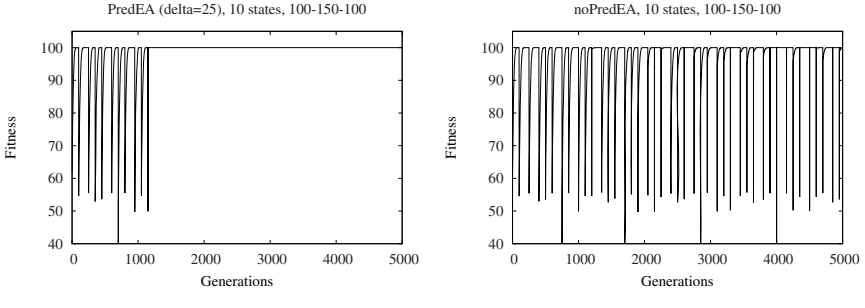

Fig. 2. PredEAversus **NoPredEA**, pattern 100-150-100, 10 states, deterministic

was 100-150-100 and the change deterministic, our algorithm was statistically significantly better than **NoPredEA**.

6 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 additional 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).

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 algorithm for future environmental changes, 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 chains predictions unhelpful. Finally, the use of a Markov chain to store the environmental information proved to be a powerful mechanism to keep the history of 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 may fail, due to large prediction errors. Second, the use of a fixed value for the error interval (the Δ parameter) assumed in the linear regression predicted values is not always 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 landscapes are also being used to test the proposed ideas.

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.

References

1. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments: a survey. IEEE Transactions on Evolutionary Computation 9(3), 303–317 (2005)
2. Yang, S.: Explicit memory schemes for evolutionary algorithms in dynamic environments. In: Yang, S., Ong, Y.S., Jin, Y. (eds.) Evolutionary Computation in Dynamic and Uncertain Environments, pp. 3–28. Springer, Heidelberg (2007)
3. Simões, A., Costa, E.: Evolutionary algorithms for dynamic environments: prediction using linear regression and markov chains. Technical Report TR 2008/01, CISUC (2008)
4. Branke, J., Mattfeld, D.: Anticipation in dynamic optimization: The scheduling case. In: Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J., Schwefel, H.P. (eds.) PPSN VI 2000. LNCS, vol. 1917, pp. 253–262. Springer, Heidelberg (2000)
5. Stroud, P.D.: Kalman-extended genetic algorithm for search in nonstationary environments with noisy fitness evaluations. IEEE Transactions on Evolutionary Computation 5(1), 66–77 (2001)
6. van Hemert, J., Hoyweghen, C.V., Lukshandl, E., Verbeeck, K.: A futurist approach to dynamic environments. In: GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems, pp. 35–38 (2001)
7. Bosman, P.: Learning, anticipation and time-deception in evolutionary online dynamic optimization. In: Yang, S., Branke, J. (eds.) GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization (2005)
8. Bosman, P.A.N.: Learning and anticipation in online dynamic optimization. In: Yang, S., Ong, Y.S., Jin, Y. (eds.) Evolutionary Computation in Dynamic and Uncertain Environments. Springer, Heidelberg (2007)
9. Bird, S., Xiaodong, L.: Using regression to improve local convergence. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2007), pp. 592–599. IEEE Press, Los Alamitos (2007)
10. Norris, J.R.: Markov Chains. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge (1997)
11. Branke, J.: Memory enhanced evolutionary algorithms for changing optimization problems. In: IEEE Congress on Evolutionary Computation (CEC 1999), pp. 1875–1882. IEEE Press, Los Alamitos (1999)
12. Simões, A., Costa, E.: Improving memory's usage in evolutionary algorithms for changing environments. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2007). IEEE Press, Los Alamitos (2007)
13. Simões, A., Costa, E.: An immune system-based genetic algorithm to deal with dynamic environments: Diversity and memory. In: Proc. of the 6th Int. Conf. on Artificial Neural Networks, pp. 168–174. Springer, Heidelberg (2003)