The Influence of Population and Memory Sizes on the Evolutionary Algorithm's Performance for Dynamic Environments

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

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

Abstract. Usually, evolutionary algorithms keep the size of the population fixed. In the context of dynamic environments, many approaches divide the main population into two, one part that evolves as usual another that plays the role of memory of past good solutions. The size of these two populations is often chosen off-line. Usually memory size is chosen as a small percentage of population size, but this decision can be a strong weakness in algorithms dealing with dynamic environments. In this work we do an experimental study about the importance of this parameter for the algorithm's performance. Results show that tuning the population and memory sizes is not an easy task and the impact of that choice on the algorithm's performance is significant. Using an algorithm that dynamically adjusts the population and memory sizes outperforms standard approach.

1 Introduction

Evolutionary Algorithms (EAs) have been used with success in a wide area of applications, involving environments either static or dynamic. Traditional EAs usually have a number of control parameters that are specified before starting the algorithm. These parameters include, for example, the probabilities of mutation and crossover or the population size. The effects of setting the parameters of EAs has been the subject of extensive research and in the last years several approaches using self-adaptive EAs, which can adjust the parameters on-the-fly, have been proposed. The study of EAs for *stationary domains* has previously focused on the adjustment of parameters of genetic operators. The choice of the size of the population has received less attention by researchers. However, if we look to natural systems that inspire EAs, the number of individuals of a certain species changes over time and tends to become stable around appropriate values, according to environmental characteristics or natural resources ([1]). In evolutionary computation the population size is usually an unchanging parameter and finding an appropriate dimension is a difficult task. Several adaptive population sizing methods have been suggested (see [2] for a review). When EAs are used to deal with *dynamic environments*, some modifications have to be introduced to avoid premature convergence since convergence is disadvantageous

M. Giacobini et al. (Eds.): EvoWorkshops 2009, LNCS 5484, pp. 705–714, 2009.

when a change happens. These enhancements include increasing diversity after a change ([3]), maintaining diversity throughout the run ([4]), using memory mechanisms ([5], [6]) and multi-population schemes ([7]). The global functioning of EAs designed to deal with dynamic applications had inherited most of the characteristics of EAs used for static domains: the adjustment of mutation rate, the use of typical values for the crossover operator and the specification of unchanging population sizes. Even, when an explicit memory is used, its size is also set at the beginning and is usually a small percentage of the main population size. Less attention has been devoted to study the influence that population size can have in the performance of the EA.

Some research has been made about this issue: Schönemann ([8]) studied the impact of population size in the context of Evolutionary Strategies for dynamic environments and the results showed that the choice of the population size can affect the algorithm's performance. Simões et al. ([9]) proposed an EA to deal with changing environments which controls the size of population and memory during the run. Results from that work show that the performance of the EA can be considerably improved. Recently, Richter et al. ([10]) proposed a memory-based abstraction method using a grid to memorize useful information and the results obtained suggest that an optimal grid size may depend on the type of dynamics.

In this work we describe an extensive empirical study whose focus was the performance of distinct memory-based EAs that faced different environmental characteristics. The study is based on two benchmark problems and is divided in two parts: in the first part, we kept the populations sizes fixed and ran the EAs for different values for these parameters in order to analyze the impact of the chosen values on the EA's performance. In the second part, an EA using population and memory whose size may vary is ran and compared with the results of the previous experimentation. We show that the values set to population and memory sizes can have a great influence in the EA's performance. The use of an algorithm capable of dynamically adjusting the population and memory sizes during the run outperforms, in general, all the scenarios analyzed using constant populations sizes. These results are statistically supported. Due to space restrictions, we will not be able to show the complete and detailed description of this work. For more details see [11]. The rest of the paper is organized as follows: next section briefly reviews memory schemes for EAs in dynamic environments. Section 3, describes the three implemented memory-based EAs and Section 4 details the dynamic test environments used for this study. The experimental results and the analysis are presented in section 5. Section 6 concludes the paper and some considerations are made about future work.

2 Memory-Based Evolutionary Algorithms for Dynamic Environments

EAs using memory schemes work by storing information about the different environments that may occur and selectively retrieving that information later when another change in the environment is detected. Information can be stored

in memory implicitly or explicitly ([12]). In the case of implicit memory the stored information is kept in redundant representations of the genotype. There are some variants, e.g., using diploid (or multiploid) representations ([13]) or dualism mechanisms ([14]). In explicit memory schemes an extra space, called **memory population**, is used. Its main goal is to store useful information about the current environment and possibly reuse it each time a change occurs. It may also permit the main population to move to a different area in the landscape in one step, which in principle is harder when we rely only on standard genetic operators. ([12]).

Considerations about Memory: When using memory-based EA some questions can be asked concerning memory: 1) when and which individuals to store in memory, 2) which size to choose, 3) which individuals should be replaced when memory is full, 4) which individuals should be selected from memory and introduced in the main population when a change happens ([12]). In explicit memory schemes, the information stored in memory corresponds to the current best individual of the main population. Other approaches, together with this information, also keep information about the environment. The size of the memory is an important issue. Since the size of the memory is limited, it is necessary to decide which individuals should be replaced when new ones need to be stored. This process is usually called *replacing strategy*. Several replacing schemes have been proposed ([5], [9], [6]). The retrieval of memory information when a change occurs depends on what was stored. In some cases, memory is re-evaluated and the best individuals of memory replace the worst of the main population. Other approaches use the stored environmental information together with the best memory individual to create new information to introduce in the population when a change happens.

Population and Memory Sizes: Usually, memory-based EAs for changing environments use a memory of small size, when compared with the dimension of the main population. In most cases, the dimension of memory is chosen between 5% and 20% of the population size, with 10% the most chosen one, as can be seen in the used values in several studies listed in [11]. Choosing a constant value for population and memory sizes is widely used in memory-based EAs. Memory is always seen as playing a secondary role in the process, and is used always with a smaller dimension.

3 Description of the Implemented Memory-Based EAs

We choose to implement a memory-immigrant based EA (MIGA) and a direct memory EA (MEGA). Both use population and memory with unchanging sizes. An additional algorithm, using a direct memory scheme with changing population and memory sizes was implemented (VMEA), ran and compared with the other two. The three algorithms will be briefly described.

Memory-Immigrants Genetic Algorithm: The Memory-Immigrants Genetic Algorithm (MIGA) uses a direct memory scheme. It was proposed by

([15]) and works in the following way: the algorithm evolves a population of individuals in the standard evolutionary way: selection, crossover and mutation. Additionally, a memory is initialized randomly and used to store the current best individual of the population, replacing one of the initially randomly generated individuals, if it exists, or replacing the most similar in memory if it is better. Every generation the best individual in memory is used to create a set of new individuals, called immigrants that are introduced into the population replacing the worst ones. These new individuals are created mutating the best solution in memory using a chosen mutation rate. When a change is detected it is expected that the diversity introduced in the population by adding these immigrants can help the EA to readapt to the new conditions.

Memory-Enhanced Genetic Algorithm: The Memory-Enhanced Genetic Algorithm (MEGA) ([15]) is an adaptation of Branke's algorithm ([5]) and evolves a population of n individuals through the application of selection, crossover and mutation. Additionally, a memory of size m is used, starting with randomly created individuals. From time to time memory is updated in the following way: if any of the initial random individuals still exist, the current best solution of the population replaces one of them arbitrarily; if not, the most similar memory updating strategy is used to choose which individual to exchange. Memory is evaluated every generation and, when a change is detected, the memory is merged with the best $n - m$ individuals of the current population to form the new population, while memory remains unchanged.

Variable-size Memory Evolutionary Algorithm: Simões and Costa ([9]) proposed a Variable-size Memory Evolutionary Algorithm (VMEA) to deal with dynamic environments. This algorithm uses a population that searches for the optimum and evolves as usual, through selection, crossover and mutation. A memory population is responsible for storing good individuals of the main population at several moments of the search process. The two populations - main and memory - have variable sizes that can change between two boundaries. The basic idea of VMEA is to use the limited resources (total number of individuals) in a flexible way. The size of the two populations can change but the their sum cannot go beyond a certain limit. If an environmental modification is detected, the best individual of the memory is introduced into the population. In the case of either the population 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. The algorithm was compared with other memory-based schemes using the standard dimensions for population and memory and the results validate its effectiveness.

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 ([16]). This generator allows

constructing different dynamic environments from any binary-encoded stationary function using the bitwise exclusive-or (XOR) operator. 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 controls how different is the new environment from the previous one.

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). This group of 16 DOPs was tested using the three different algorithms with two benchmark problems, using ten different combinations os population and memory sizes. A total of 960 different situations were tested in this work. The selected benchmark problems, used to test the different EAs with different parameter settings were the dynamic Knapsack problem and the Onemax problem. 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 (C). We used a Knapsack problem with 100 items using strongly correlated sets of randomly generated data. The **Onemax problem** aims to maximize the number of ones in a binary string. So, the fitness of an individual is equal to the number of ones present in the binary string. This problem has a unique solution. In our experiments we used individuals of length 300. For more details see [11].

4.2 Parameters Setting

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 with probability $p_m = 0.01$. Binary representation was used with chromosomes of size 100 for the Knapsack and 300 for Onemax problem. The probability of mutation and the ratio of immigrants introduced in population used in MIGA were 0.01 and 0.1, respectively, as suggested by [15].

All simulations used a global number of individuals equal to 100. These individuals were divided in main population and memory, in the following way: the main population used a population size between 10 and 90 individuals, in increases of 10 (10, 20,..., 90). Memory size was calculated with the remaining individuals: $M(size) = 100 - P(size)$. The *generational* replacing strategy proposed by [6] was used in all EAs. For each experiment of an algorithm, 30 runs were executed. Each algorithm was run for a number of generations corresponding to 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.

5 Results

In this section we will show partial results concerning: *(a)* the overall performance of the studied algorithms; *(b)* the adaptability of the algorithms along time, *(c)* the diversity of population and memory for the different algorithms and *(d)* the variation of population and memory sizes in VMEA.

The global results regarding MIGA and VMEA in the the Knapsack problem under different environments are shown in Figure 1. VMEA's scores are plotted in the upper right corner. As we see, depending on the change period and the change ratio, the choice of population and memory sizes affects the algorithm's performance. For the Knapsack problem, using MIGA, the best choice was population with 60 or 50 individuals (and memory of 40 and 50 respectively), for dynamics changing rapidly ($r = 10$). In environments with slower changes ($r = 50$, $r = 100$, $r = 200$), the best results were achieved with population size of 80 or 70 and memory with 20 or 30 individuals, respectively. Either smaller or larger populations lead to a decrease of the EAs' performance. For the Onemax problem, MIGA obtained best results using population size of 40 individuals using faster changes and lower change ratios ($\rho = 0.1$ and $\rho = 0.2$) and population with 60 individuals for larger change ratios ($\rho = 0.5$ and $\rho = 1.0$). Increasing the change ratio and the change period, larger populations are required to achieve the best results: 70 to 90 individuals. Observing the graphics, it is visible the influence that population and memory sizes can have in the performance of this algorithm. It is evident there must be a tradeoff between memory and population sizes. The usual split of 10% for the memory size and 90% for the population size is not the best choice. In all cases, VMEA outperformed all instances of MIGA and MEGA (not shown here for lack of space), demonstrating robustness and adaptability in the studied problems under different environmental characteristics. As happens in the stationary environments ([2]) it is possible that different sizes of population and memory might be optimal at different stages of the evolutionary process.

The statistical validation of the obtained results used paired one-tailed t-test at a 0.01 level of significance and can be found in [11]. Statistical results support that choosing different sizes for the population and memory affects the algorithm's performance and this difference is statistically significant. In general, VMEA performed significant better than MIGA's and MEGA's best results on most dynamic environments. These results validate our expectation of the impact that a bad choice of population and memory sizes can have in EA's performance and also that dynamically sizing approaches should be further investigated.

Figure 2 shows an example of how the different algorithms evolved through the entire run for the Knapsack problem. These results were obtained using $\rho = 0.5$. For the other cases, the results were analogous. In all cases the difference in the algorithm's performance using a "good choice" and a "bad choice" for the population and memory sizes is considerable, especially in rapidly changing environments. In MIGA's and MEGA's worst results, the evolution is slower and the best performance is only achieved when $r = 200$ and at the end of the process, since more time between changes is given to the algorithms. VMEA's performance is

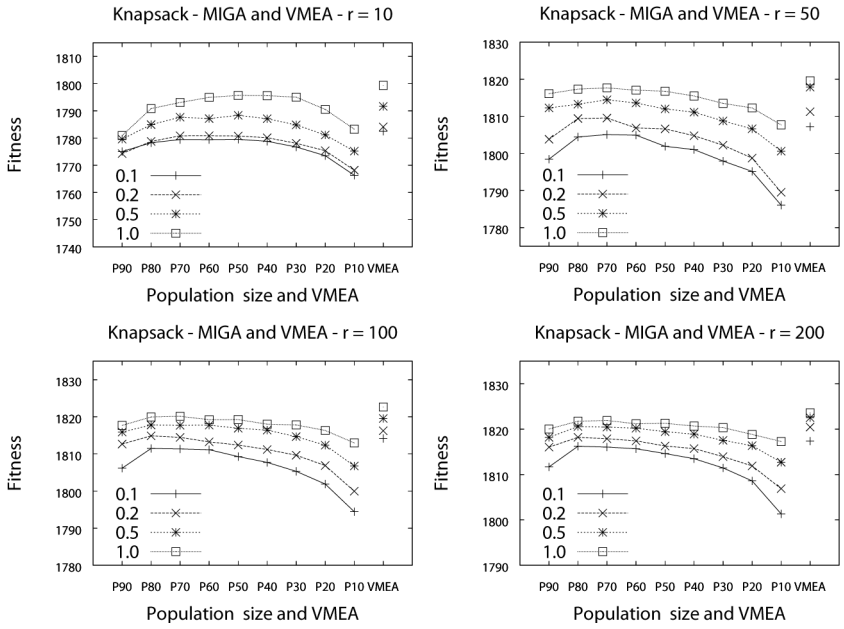

Fig. 1. Global results obtained in the dynamic **Knapsack** problem using MIGA (with different population and memory sizes) and VMEA

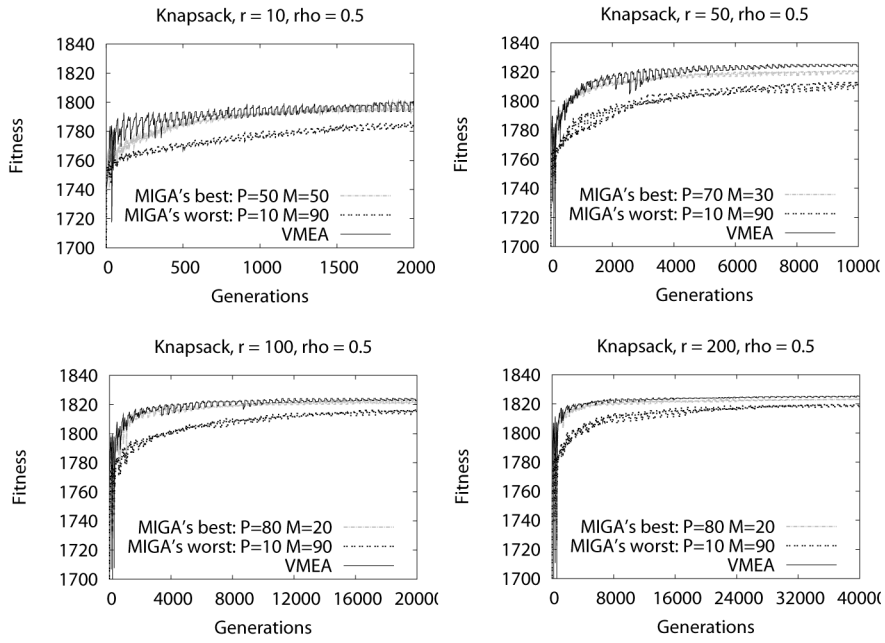

Fig. 2. Behavior of MIGA and VMEA in the **Knapsack problem**, $r = 10$, $r = 50$, $r = 100$ and $r = 200$, $\rho = 0.5$

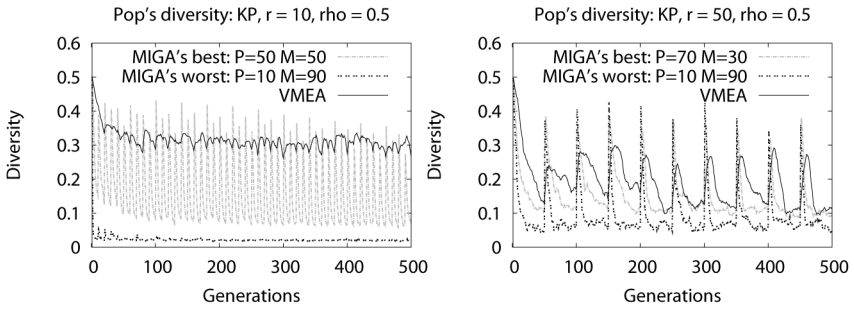

Fig. 3. Population's Diversity using MIGA and VMEA in the Knapsack problem, $r = 10$ and $r = 50$, $\rho = 0.5$

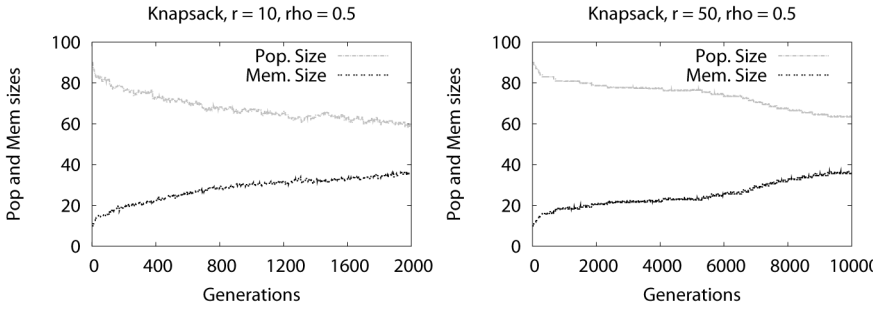

Fig. 4. Changes on Population and Memory Sizes using VMEA in Knapsack, $r = 10$ and $r = 50$, $\rho = 0.5$

always superior when compared with best performances of the other algorithms. To understand the previously shown results, we analyzed the diversity for different sizes of population and memory. Figure 3 shows the population's diversity using MIGA (best and worst results) and VMEA for the Knapsack problem. As we can see the worst performance of MIGA corresponds to the lower population's diversity and higher memory diversity. VMEA was the algorithm that maintains higher diversity in the population. In MIGA this observation was generalized to all situations. For MEGA the results are not clear and there are some cases where the best results were achieved by the population and memory sizes that maintained lower diversity. Thus, it's not straightforward that diversity is the reason for best or worst performances. Figure 4 shows the evolution of population and memory sizes during time using VMEA for different change periods in the Knapsack problem. The evolution of population and memory sizes has a typical behavior. In general memory tends to increase and population to decrease. As the number of generations increase this increase/decrease is slower. At the end of the process, population and memory sizes stabilize around similar values: population of 60-70 individuals and memory of 40-30 individuals, corresponding

to the set of values that often lead to best scores using MIGA and MEGA. The adjustment of population and memory sizes performed by VMEA can be considered *"blind"* since its only restrictions are the fixed boundaries and the global number of individuals. Nevertheless, superior results were attained.

6 Conclusions and Future Work

Usually, memory-based EAs dealing with dynamic environments use population and memory of constant sizes. Typically, memory size is set as a small percentage of population size. In this work we tried to understand if different population and memory sizes can have considerable influence in the performance of memory-based EAs dealing with different dynamic environments. Two direct memory EAs were ran with different values for population and memory sizes in two benchmark problems. A third algorithm, using dynamically adjusting population and memory sizes was also tested and compared with the other two algorithms. The obtained results show that the traditionally used values for population and memory sizes do not allow the best performance of the implemented EAs. It is clear that the tuning of the population and memory sizes has significant influence in the efficacy and convergence of the EAs. The best choice of values depends on the environmental characteristics, the problem to solve or the used algorithm. So, this choice is not linear or easy and trying to tune the population size before running the algorithm is practically impossible, since the combinations are huge and the process is time consuming. A preferred solution is the use of an EA capable of controlling the populations sizes during the run. In this work we used a simple EA which dynamically adjusts the population and memory dimensions and the results demonstrate its effectiveness and robustness to all environments and problems tested.

As future work we plan to investigate other sizing mechanisms in EAs to cope with dynamic environments. These approaches should take into account other information besides the limited number of the individuals that can exist, like, for example, the average fitness of population or aging mechanisms.

Acknowledgments

The work of the first author described in this paper 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. Gould, J.L., Keeton, W.T.: Biological Science. W. W. Norton & Company (1996)
2. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer, Heidelberg (2003)

3. Cobb, H.G.: An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Technical Report TR AIC-90-001, Naval Research Laboratory (1990)
4. Yang, S.: Genetic algorithms with elitism-based immigrants for changing optimization problems. In: Giacobini, M., et al. (eds.) EvoWorkshops 2007. LNCS, vol. 4448, pp. 627–636. Springer, Heidelberg (2007)
5. Branke, J.: Memory enhanced evolutionary algorithms for changing optimization problems. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999), pp. 1875–1882. IEEE Press, Los Alamitos (1999)
6. 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), pp. 276–283. IEEE Press, Los Alamitos (2007)
7. Branke, J., Kaußler, T., Schmidt, C.: A multi-population approach to dynamic optimization problems. In: Parmee, I. (ed.) Proceedings of Adaptive Computing in Design and Manufacture (ACDM 2000), pp. 299–308. Springer, Heidelberg (2000)
8. Schönemann, L.: The impact of population sizes and diversity on the adaptability of evolution strategies in dynamic environments. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2004), vol. 2, pp. 1270–1277. IEEE Press, Los Alamitos (2004)
9. Simões, A., Costa, E.: Variable-size memory evolutionary algorithm to deal with dynamic environments. In: Giacobini, M., et al. (eds.) EvoWorkshops 2007. LNCS, vol. 4448, pp. 617–626. Springer, Heidelberg (2007)
10. Richter, H., Yang, S.: Memory based on abstraction for dynamic fitness functions. In: Giacobini, M., et al. (eds.) EvoWorkshops 2008. LNCS, vol. 4974, pp. 596–605. Springer, Heidelberg (2008)
11. Simões, A., Costa, E.: The influence of population and memory sizes on the evolutionary algorithm's performance for dynamic environments. Technical Report TR 2008/02, CISUC (2008)
12. Branke, J.: Evolutionary Optimization in Dynamic Environments. Kluwer Academic Publishers, Dordrecht (2002)
13. Uyar, A.S., Harmanci, A.E.: A new population based adaptive dominance change mechanism for diploid genetic algorithms in dynamic environments. Soft Computing 9(11), 803–814 (2005)
14. Yang, S., Yao, X.: Experimental study on population-based incremental learning algorithms for dynamic optimization problems. Soft Computing 9(11), 815–834 (2005)
15. Yang, S.: Memory-based immigrants for genetic algorithms in dynamic environments. In: Beyer, H.-G. (ed.) Proceedings of the Seventh International Genetic and Evolutionary Computation Conference (GECCO 2005), vol. 2, pp. 1115–1122. ACM Press, New York (2005)
16. Yang, S.: Associative memory scheme for genetic algorithms in dynamic environments. In: Rothlauf, F., et al. (eds.) EvoWorkshops 2006. LNCS, vol. 3907, pp. 788–799. Springer, Heidelberg (2006)