Published in M. Giacobini et al. (Eds.): EvoWorkshops 2007, Applications of Evolutionary Computing, LNCS 4448, pp. 617-626, Springer Verlag, 2007.

Variable-size Memory Evolutionary Algorithm to Deal with Dynamic Environments

Anabela Simões[1,2], Ernesto Costa[2]

[1] Dept. of Informatics Engineering, ISEC – Coimbra Polytechnic, R. Pedro Nunes Quinta da Nora, 3030-199 Coimbra - Portugal
[2] CISUC, University of Coimbra, Polo II, 3030-290 Coimbra - Portugal
abs@isec.pt, ernesto@dei.uc.pt

Abstract. When dealing with dynamic environments two major aspects must be considered in order to improve the algorithms' adaptability to changes: diversity and memory. In this paper we propose and study a new evolutionary algorithm that combines two populations, one playing the role of memory, with a biological inspired recombination operator to promote and maintain diversity. The size of the memory mechanism may vary along time. The size of the (usual) search population may also change in such a way that the sum of the individuals in the two populations does not exceed an established limit. The two populations have minimum and maximum sizes allowed that change according to the stage of the evolutionary process: if an alteration is detected in the environment, the search population increases its size in order to readapt quickly to the new conditions. When it is time to update memory, its size is increased if necessary. A genetic operator, inspired by the biological process of conjugation, is proposed and combined with this memory scheme. Our ideas were tested under different dynamics and compared with other approaches on two benchmark problems. The obtained results show the efficacy, efficiency and robustness of the investigated algorithm.

1 Introduction

Evolutionary Algorithms (EAs) have been used with success in a wide area of applications. Traditionally, EAs are well suited to solve problems where the environment is static. The generational process of evolution often leads the EA to the best solution. However, most of real-world applications are dynamic and the algorithms used to solve them must be able to adapt to the new circumstances.

For this type of optimization, an effective EA must be able to deal with the changes, detecting and reacting rapidly when they occur. Classical EAs are not suited for this kind of problems, since they have the tendency to prematurely converge to a solution and, when the conditions of the environment change, the population has all individuals usually concentrated in a specific point of the search space. So, it takes some time for the population to readapt and move towards the new solution.

To deal with these limitations, some improvements have been proposed as extensions of the classical EA. These improvements include 1) maintaining diversity using several strategies [4, 6, 8, 10, 15], 2) using memory schemes either implicit [5,

11, 13] or explicit [1, 15, 20, 22] and 3) using multi-populations [3, 18]. Recent works successfully studied the combination of memory schemes and mechanisms for promoting and maintaining diversity [15, 19].

In this paper we are also interested in studying the combination of these two important issues. We propose an EA with a search population and a memory population whose size may change. Both search and memory populations have minimum and maximum values and the global number of individuals in the two populations cannot go beyond an established value. The memory is updated from time to time and its size changes whenever it is necessary and possible. If the maximum size is reached, the memory is cleaned to make room for new individuals. When a change occurs, the best individuals from the memory are selected and inserted in the population.

Besides this memory framework, we introduce a new way of using the biologically inspired conjugation operator, which improves the quality of the solutions. Conjugation will be used as the main recombination operator. This mechanism is applied after the mating pool is selected. The best individuals of the pool will transfer part of their genetic material to the worst individuals of the pool. After that, the new individuals are merged with the old population to form the next population.

2 Memory and Diversity: Relevant Background

The most known approach using redundant representations is to use diploid instead of haploid chromosomes. This idea was suggested by [5] as an extension of the standard GA. Later, other authors investigated the idea of diploidy, particularly its use in the context of dynamic environments [11, 13] .

When using explicit memory the main goal is to store useful information (good solutions) about the current environment and reuse them when a change occurs. It can also permit the population to move to a different area in the landscape in one step, which would not be possible with common genetic operators [2]. Different approaches have been proposed in the literature. For instance, [1, 15, 20, 22].

Several techniques have been proposed in order to preserve the diversity in a population. The most popular techniques to promote populations' diversity are hypermutation [4, 10], random immigrants [6], tag bits [8] or alternative genetic operators [15]. Their aim is to maintain or increase population's diversity in an EA allowing a quick reaction when modifications are detected. There are some works where these two issues – diversity and memory – are combined within the same algorithm claiming improved performance [15, 19].

3 Variable-size Memory Evolutionary Algorithm

We propose a new EA called VMEA – Variable-size Memory Evolutionary Algorithm, comprising two populations: the main population searches the best solution and evolves as usual through selection, crossover and mutation. The main difference is that its size can change between two bounds: POP_MIN and POP_MAX.

The second population plays the role of a memory, where the best individuals of the population in several points of the generational process are stored. Its size also changes between off-line established limits MEM_MIN and MEM_MAX. The sum of the two populations cannot go beyond a certain limit (TOTAL_MAX). The individuals of the memory are aged: their age starts at zero, being increased by one at every generation. If an individual is selected to the population when a change is detected, an extra value is added to its age. Oldest individuals are those who stay longer in memory and/or contributed to adaptability of the population in an environmental change. If individuals reach a LIMIT_AGE their ages are reset to zero. The age of the individuals in the memory is used to select which individual to choose to withdraw when memory is full (or the sum of the size of the two populations is equal to the permitted limit). 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. If there is no room to save this new solution, we first clean the memory removing the individuals with the same genotype. If no individual was deleted through this process of cleaning, then the best individual of the current population, if is fittest than the one with lowest age present in the memory, replaces it.

The memory is evaluated every generation and a change is detected if at least one individual in the memory changes its fitness (as it was suggested in [1] and [22]) and it is updated at time $T_M=t + rand(5,10)$, the same way as in [22].

If an environmental modification is detected, the best individual of the memory is introduced into the population. In the case of either the population's size or the sum of the two populations reach the allowed maximum, the best individual in memory replaces the worst one in the current population. A complete description of the algorithm can be consulted in [16].

4 Promoting Diversity in the Search Population

Traditionally EAs use crossover as the main genetic operator. In the past other biologically inspired operators have been proposed and tested with some degree of success. These new genetic operators were applied either in static [7, 14], or dynamic environments [15, 19].

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 [12].

Computational conjugation was introduced independently by Harvey and Smith. Smith [17] proposed an implementation of this operator, called simple conjugation: the donor and the recipient were chosen randomly, transferring the genetic material between two random points. Harvey [7] introduced a tournament based conjugation: two parents are selected on a random basis, and then the winner of the tournament becomes the donor and the loser the recipient of the genetic material. That way, the conjugation operator can be applied repeatedly by different donors to a single recipient.

In this paper conjugation is applied differently. We perform conjugation involving the individuals *selected to the mating pool*, using the idea of donor-recipient genetic transfer. As it happens in biology, the donor individuals give genetic material to the recipient ones. After selecting the individuals to mate, using the established selection method, they are divided into two groups: the $\boldsymbol{n}/2$ best individuals become the 'donor', the remaining become the 'recipient' ($\boldsymbol{n}$ is the current size of the population). Then, the i^{th} donor transfers part of its genetic material to the i^{th} recipient (i=1, …n/2). This injection is controlled by two points randomly chosen. The donor remains unchanged. Following that, all offspring created by this process are joined with the donor individuals and they become the next population of size **n**. A complete explanation of conjugation is given in [16].

5 Experimental Study

5.1. Dynamic Test Environments

We selected two benchmark problems to test our VMEA so we can compare it with other algorithms easier: the knapsack problem (100 items) and *oneMax* problem (300 bits). In this paper, due to the restrictions of space, we only show partial results. A detailed set of results can be consulted in [16].

Knapsack 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. We used a knapsack problem with 100 items using strongly correlated sets of randomly generated data [9, 21]. 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.

OneMax problem
The *OneMax* problem aims to maximize the number of ones in a binary string. So the fitness of an individual consists in the number of ones present in the binary string. This problem has a unique solution. In our experiments we used individuals of length 300.

Three kinds of environments were created for each of these two base problems, using Yang's Dynamic Optimization Problems (DOP) generator: cyclic, cyclic with noise and random. The environment was changed every $\boldsymbol{r}$ generations ($\boldsymbol{r}$ = 10, 50, 100 and 200) and the ratio $\boldsymbol{\rho}$ was set to different values in order to test different levels of change: 0.1 (a light shifting) 0.2, 0.5, 1.0 (severe change). In order to study the behaviour of the algorithms in randomly changing environments we also set $\boldsymbol{\rho}$ to a uniformly randomly generated value in the interval [0.01 and 0.99] (called by ***rnd***).

Details about Yang's DOP generator can be found in [20]. With this generator it is possible to construct different dynamic environments from any binary-encoded stationary function using the bitwise exclusive-or (XOR) operator. The basic idea of the generator can be described as follows: when evaluating an individual **x** in the population, first we perform the operation $\mathbf{x} \oplus \mathbf{M}$ 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 $f(x, t+1) = f(x \oplus M)$. Using the DOP generator the characteristics of the change are controlled by two parameters: the speed of the change, $\boldsymbol{r}$, that is the number of generations between two changes, and the magnitude of the change, $\boldsymbol{\rho}$, 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 $\boldsymbol{\rho}$ 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.

5.2. Experimental Setup

Algorithms' parameters

To compare our approach we used two other algorithms: the random immigrants algorithm [6] and the memory-enhanced GA (MEGA) studied in [22]. VMEA was tested using conjugation (VMEA-Cj) and uniform crossover (VMEA-Cx), in order to make conclusions about the efficiency of the proposed genetic operator in changing problems. For all the algorithms, 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 (the same probability was used with conjugation) and mutation applied with probability p_m=0.01. The population size for VMEA, RIGA and MEGA was set to 120 individuals. In MEGA, 10 individuals were used as memory, which is updated according the description given in [22]. The ratio of immigrants used in RIGA was 0.1. The mutation ratio used for noisy environments was 0.05. In VMEA the memory size varied between 10 and 50 individuals. However, the total of individuals in the two populations could not surpass 120. The age limit for the individuals in memory was set to $\boldsymbol{G/2}$, where $\boldsymbol{G}$ is the total number of generations.

For each experiment of an algorithm, 30 runs were executed and the number of environmental changes was 100 with $\boldsymbol{r}$ =10 (1000 generations), 40 with $\boldsymbol{r}$ =50 (2000 generations) and 20 with $\boldsymbol{r}$=100 and 200 (2000 and 4000 generations, respectively). 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:

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

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

5.3 Experimental Results

The experimental results carried out to assess the efficiency of our algorithm shows that VMEA outperformed the other two approaches. 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. Fig. 1 plots the average of the best-of-generation fitness obtained in the knapsack problem. Fig. 2 and 3 show some examples of the algorithms' behaviour during the generations.

Table 1. The t-test results of comparing the different algorithms (*knapsack* problem).

		CYCLIC					CYCLIC WITH NOISE					NON CYCLIC				
Statistical significance	r, ρ→	0.1	0.2	0.5	1.0	rnd	0.1	0.2	0.5	1.0	rnd	0.1	0.2	0.5	1.0	rnd
VMEA Cx - RIGA		++	++	++	++	++	--	++	++	++	--	++	-	--	++	--
VMEA Cj - RIGA		++	++	++	++	++	--	++	++	++	--	++	--	--	++	--
VMEA Cx - MEGA	10	++	++	++	++	++	+	++	++	++	++	++	+	++	++	++
VMEA Cj - MEGA		++	++	++	++	++	+	++	++	++	-	++	--	--	++	++
VMEA Cj - VMEA Cx		++	++	++	++	++	-	-	++	++	-	++	--	--	++	--
VMEA Cx - RIGA		++	++	++	++	++	++	++	++	++	++	++	++	+	++	++
VMEA Cj - RIGA		++	++	++	++	++	++	++	++	++	++	++	++	-	++	++
VMEA Cx - MEGA	50	++	++	++	++	++	++	++	++	++	++	++	++	++	++	++
VMEA Cj - MEGA		++	++	++	++	++	++	++	++	++	++	++	++	++	++	++
VMEA Cj - VMEA Cx		++	++	++	++	++	++	++	++	++	++	++	-	--	++	++
VMEA Cx - RIGA		++	++	++	++	++	++	++	++	++	++	++	++	++	++	++
VMEA Cj - RIGA		++	++	++	++	++	++	++	++	++	++	++	++	++	++	++
VMEA Cx - MEGA	100	++	++	++	++	++	++	++	++	++	++	++	++	++	++	++
VMEA Cj - MEGA		++	++	++	++	++	++	++	++	++	++	++	++	++	++	++
VMEA Cj - VMEA Cx		++	++	++	++	++	++	++	++	++	++	++	++	--	++	++
VMEA Cx - RIGA		++	++	++	++	++	++	++	++	++	++	++	++	++	++	++
VMEA Cj - RIGA		++	++	++	++	++	++	++	++	++	++	++	++	++	++	++
VMEA Cx - MEGA	200	++	++	++	++	++	++	++	++	++	++	++	++	++	++	++
VMEA Cj - MEGA		++	++	++	++	++	++	++	++	++	++	++	++	++	++	++
VMEA Cj - VMEA Cx		++	++	++	++	++	++	++	++	++	++	++	++	+	++	++

Table 2. The t-test results of comparing the different algorithms (*OneMax* problem).

		CYCLIC					CYCLIC WITH NOISE					NON CYCLIC				
Statistical significance	r, ρ→	0.1	0.2	0.5	1	rnd	0.1	0.2	0.5	1	rnd	0.1	0.2	0.5	1	rnd
VMEA Cx - RIGA		++	++	++	++	++	++	++	++	++	++	++	++	+	++	+
VMEA Cj - RIGA		++	++	++	++	++	--	--	--	--	--	--	--	--	++	--
VMEA Cx - MEGA	10	++	++	++	++	++	++	++	++	++	++	++	++	++	++	++
VMEA Cj - MEGA		++	++	++	++	++	--	--	--	--	--	--	--	--	++	--
VMEA Cj - VMEA Cx		--	--	--	--	--	--	--	--	--	--	--	--	--	++	--
VMEA Cx - RIGA		++	++	++	++	++	++	++	++	++	++	++	++	++	++	++
VMEA Cj - RIGA		++	++	++	++	++	++	++	++	++	+	++	--	--	++	--
VMEA Cx - MEGA	50	++	++	++	++	++	++	++	++	++	++	++	++	++	++	++
VMEA Cj - MEGA		--	++	++	++	--	--	--	--	--	--	--	--	--	++	--
VMEA Cj - VMEA Cx		--	--	++	++	--	--	--	--	--	--	--	--	--	++	--
VMEA Cx - RIGA		++	++	++	++	++	++	++	++	++	++	++	++	++	++	++
VMEA Cj - RIGA		++	++	++	++	++	++	++	++	++	++	++	++	--	++	++
VMEA Cx - MEGA	100	++	++	++	++	++	++	++	++	++	++	++	++	++	++	++
VMEA Cj - MEGA		--	++	++	++	--	--	--	--	-	--	--	--	--	++	--
VMEA Cj - VMEA Cx		--	--	++	++	--	--	--	--	--	--	--	--	--	++	--
VMEA Cx - RIGA		++	++	++	++	++	++	++	++	++	++	++	++	++	++	++
VMEA Cj - RIGA		++	++	++	++	++	++	++	++	++	++	++	++	++	++	++
VMEA Cx - MEGA	200	++	++	++	++	++	++	++	++	++	++	++	++	++	++	++
VMEA Cj - MEGA		++	++	++	++	++	++	++	++	++	++	++	-	--	++	-
VMEA Cj - VMEA Cx		++	++	++	++	++	--	--	--	--	--	++	--	--	+	--

In *cyclic environments*, our approach obtained at all times the best solutions. As expected, RIGA had the worst performance, obviously because it doesn't use any memory mechanism. Comparing VMEA and MEGA, we can conclude that the mechanism we introduced performs very well in cyclic environments. Also,

conjugation shows a better performance in cyclic environment, obtaining almost all the times the best results. Decreasing the ratio of change, the effect of memory is not so visible. In fact, both memory algorithms, VMEA and MEGA, need some time to readapt when a change happens. This is because with small changes in the XOR mask, when a repeated state reappears, memory has already lost the useful information previously stored.

For *cyclic with noise and random environments* the results were in general very poor. The four algorithms didn't achieve high results as in the case of cyclic environments. Nevertheless, VMEA obtained, in most cases, the best results, as we can see in Tables 1 and 2.

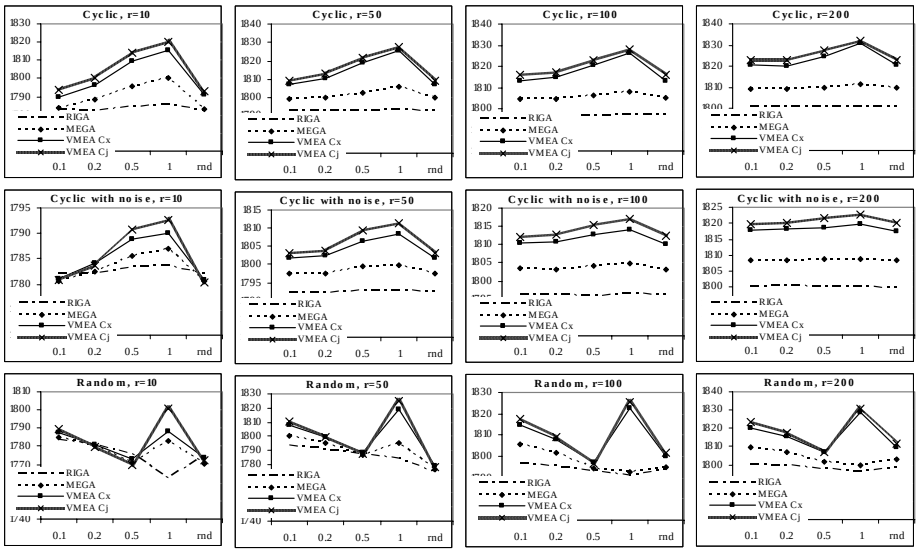

Fig. 1. Global results for the *knapsack* problem.

In *cyclic with noise environments*, algorithms behave in similar way: after a change they need some time to readapt and find a best solution. Memory improves the algorithm (VMEA-Cx achieves, in general, the highest scores), but its effect is not as obvious as in cyclic environment. Fig. 2 shows the evolutionary behavior of the algorithms for ρ=1 and ρ=0.1, with r=10 and r=200.

In *random environments*, with high ratio changes (ρ=1), VMEA achieved very good results. The memory allows the algorithm to continuously improve its performance. The good performance reduced as we decrease the change ratio. The new environment is slightly different from the previous one, but repeated states appear after a long time on and so memory has already lost the related information. Fig. 3 shows the behavior of the algorithms in noisy and random environments with ρ=1. For random change ratio (ρ = *rnd)* we observed a degradation of the results. In this case, RIGA and the other two memory-based algorithms performed in a very similar way: after a change in the environment the best-of generation falls for lower values and it is required some time for the algorithms to start evolving again. Even so, VMEA, typically arise the best marks.

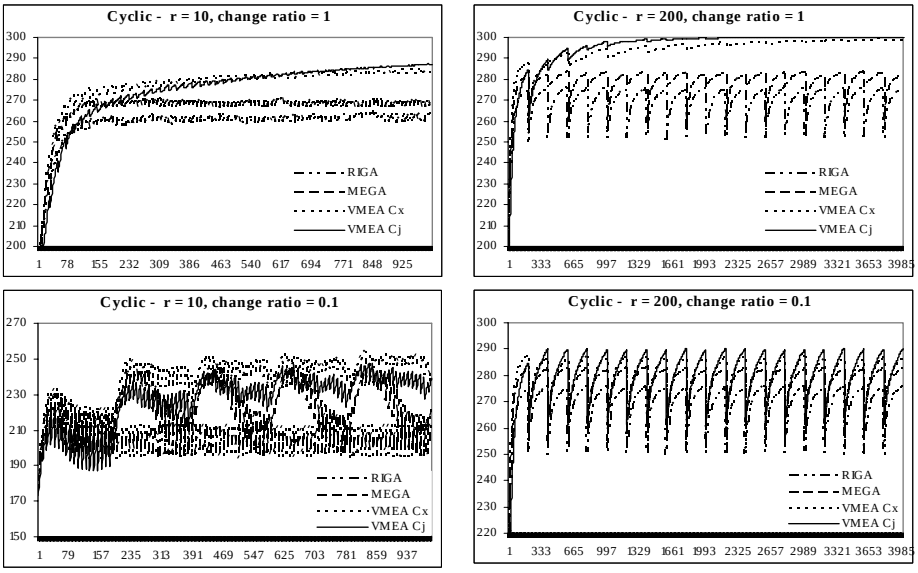

Fig. 2. Dynamic behavior of the algorithms in cyclic environments (*OneMax* problem)

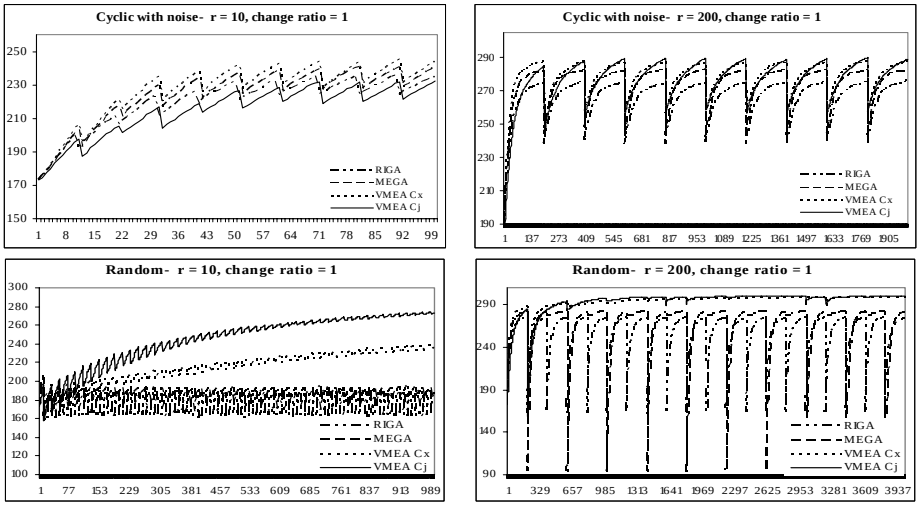

Fig. 3. Dynamic behavior in cyclic with noise and random environments (*OneMax* problem).

The VMEA algorithm combined with the conjugation operator performed better in the knapsack problem. In fact, VMEA-Cj was, usually, the best algorithm. The same was not observed in the *OneMax* problem. In this benchmark problem, VMEA-Cx obtained the highest marks and VMEA with conjugation performed better in cyclic environments with larger change periods.

5.4 Memory and Population Sizes

The restrictions we impose when we increase the size of the memory and search populations imply that memory tends to grow until its maximum is achieved and so population is 'penalized' because we run out of resources. This happens because, when the established limits are attained, we only increase the population size when there is room for at least one more individual and this is possible only if some individuals of the memory have been cleaned. After the maximum value for the memory is reached, the process of deleting individuals with the same genotype from the memory allows periodical increases in the population size. Fig. 4 shows a representative graphic of the evolution of the populations' size.

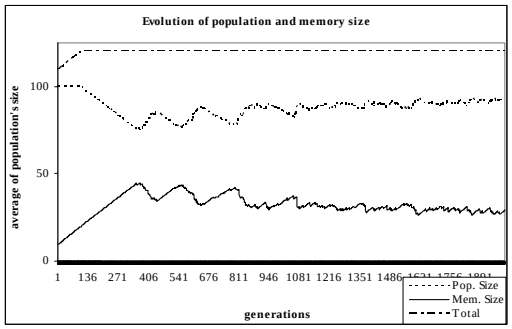

Fig. 4. Evolution of the population's and memory's size.

6 Conclusions

In this paper we proposed an EA with memory of variable size to deal with dynamic environments. Additionally, we introduced a different biological operator to test its efficiency in the promotion of diversity. The investigated algorithm, called VMEA, was tested and compared with other approaches in different dynamic environments: cyclic, cyclic with noise and random. From the obtained results we can conclude that VMEA is very efficient. The best results were observed in cyclic environments: the greater the change ratio, the better the performance. For small change ratios, besides the change is not so severe, there are more different states reappearing in the environment. We can also conclude that the combination of the variable memory scheme and the conjugation operator increases the performance of the algorithm, mainly in cyclic environments. Finally, we must stress that for the implemented and compared algorithms, VMEA predominantly achieved the best results.

References

1. J. Branke. Memory Enhanced Evolutionary Algorithms for Changing Optimization Problems. *Proc. of Congress on Evol. Computat. 1999*, pp. 1875-1882, IEEE, 1999.

2. J. Branke. *Evolutionary Optimization in Dynamic Environments*. Norwell MA: Kluwer, 2001.
3. J. Branke, T. Kaußler, C. Schmidt and H. Schmeck. A Multi-Population Approach to Dynamic Optimization Problems. *Adaptive Computing in Design and Manufacture (ACDM 2000*), pp. 299-308, Springer, 2000.
4. H. Cobb. An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having Continuous, Time-Dependent Non-Stationary Environments. *Technical Report* AIC-90-001, 1990.
5. D. E. Goldberg and R. E. Smith. Nonstationary Function Optimization using Genetic Algorithms with Dominance and Diploidy. *Proc. of the 2nd Int. Conf. on Genetic Algorithms*, pp. 59-68. Laurence Erlbaum Associates, 1987.
6. 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.
7. I. Harvey. The Microbial Genetic Algorithm. *Unpublished*, 1996.
8. W. Liles and K. De Jong. The Usefulness of Tag Bits in Changing Environments. *Proceedings of Congress on Evol. Computat. 1999*, pp. 2054-2060, IEEE, 1999.
9. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. 3rd Edition Springer-Verlag, 1999.
10. R. W. Morrison and K. De Jong. Triggered Hypermutation Revisited. *Proc. of Congress on Evol. Computat. 1999*, pp. 1025-1032, IEEE, 1999.
11. K. P. Ng and K. C. Wong. A New Diploid Scheme and Dominance Change Mechanism for Non-stationary Function Optimization. *Proc. of the 6th Int. Conf. on Genetic Algorithms*, pp. 159-166, Morgan Kaufmann, 1995.
12. P. J. Russell. *Genetics*. 5th edition, Addison-Wesley, 1998.
13. A. Sima Uyar and A. Emre Harmanci. A New Population Based Adaptive Domination Change Mechanism for Diploid Genetic Algorithms in Dynamic Environments. *Soft Computing*, vol. 9, pp. 803-814, 2005.
14. A. Simões and E. Costa. Transposition: A Biologically Inspired Mechanism to Use with Genetic Algorithms. *Proc. of the 4th Int. Conf. on Artificial Neural Networks*, pp. 612- 19, Springer-Verlag 1999.
15. 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.
16. A. Simões and E. Costa. Variable-size Memory Evolutionary Algorithm to Deal with Dynamic Environments: an empirical study. *CISUC Technical Report*, TR 2006/004 - ISSN 0874-338X, November 2006.
17. P. Smith. Conjugation: A Bacterially Inspired Form of Genetic. *Late Breaking Papers at the Genetic Programming 1996 Conf.*, Stanford Univ., July 28-31, 1996.
18. M. Wineberg and F. Oppacher. Enhancing GA's Ability to Cope with Dynamic Environments. *Proc. of the 2000 Genetic and Evol. Comp. Conf.*, pp. 3-10, Las Vegas, USA, 8-12 July, San Francisco, CA, 2000.
19. S. Yang. A Comparative Study of Immune System Based Genetic Algorithms in Dynamic Environments. *Proc. of the 2006 Genetic and Evol. Comp. Conf*, pp. 1377-1384, ACM Press, 2006.
20. S. Yang. Associative Memory Scheme for Genetic Algorithms in Dynamic Environments. *Proc. of the EvoWorkshops 2006*, LNCS 3097, pp. 788-799, Springer-Verlag, 2006.
21. S. Yang. Experimental Study on Population-Based Incremental Learning Algorithms for Dynamic Optimization problems. *Soft Computing*, vol. 9, nº 11, pp. 815-834, 2005.
22. S. Yang. Memory-Based Immigrants for Genetic Algorithms in Dynamic Environments. *Proc. of the 2005 Genetic and Evol. Comp. Conf*, Vol. 2, pp. 1115-1122, ACM Press, 2005.