Proceedings of the IEEE 2007 Congress on Evolutionary Computation, Singapore, 25-28 September 2007.

Improving Memory's Usage in Evolutionary Algorithms for Changing Environments

A. Simões, E. Costa

Abstract—**When using Evolutionary Algorithms (EAs) in dynamic environments some extensions have been introduced in order to avoid the premature convergence of the population towards a non-optimal point of the search space. One of these improvements consists in adding an explicit memory used for storing good individuals from the search population. When the environment is cyclic and previous environments reappear later memory should maintain EA's performance by keeping individuals' fitness at an acceptable level. But in most situations this purpose is not achieved and the typical behavior of an EA when a change occurs involves the decrease of the best individual's fitness and some time is necessary to readapt to the new conditions. The key problem when using explicit memory is the restriction usually imposed on its size. So, when it is necessary to store a new individual and memory is full we need to replace some individuals. This replacement can lead to the destruction of information that might be useful in the future. In this paper we are interested in the enhancement of memory's usage and we propose two new replacing methods to apply when memory is full. The investigated methods were tested in several memory-based EAs and the results show that memory can be used in a more effective way so algorithms' performance is strongly improved.**

I. INTRODUCTION

EVOLUTIONARY algorithms (EAs) have been widely used in the search of good solutions in changing environments. One of the most used mechanisms is the incorporation of explicit memory, which plays the role of storage for good individuals obtained during the evolutionary process, which can be useful in future situations [1], [5], [7], [9], [14] and [16]. The standard approaches use memory with restricted size and when it is full individuals previously stored have to be replaced. This can destroy information that could be valuable in future contexts.

When the environment is cyclic and prior environments come back later an EA with explicit memory must be able to store and keep relevant information to provide the minimum decrease of the algorithms' performance when a change happens. Generally this is not observed and the typical curve of an EA's performance when dealing with dynamic environments is characterized by the reduction of the best individual's fitness followed by the algorithm's readaptation to the new conditions of the environment [7], [15].

In a previous work we proposed an algorithm called Variable-size Memory Evolutionary Algorithm (VMEA) that includes a search and memory populations with variable size that was used with considerable success in changing environments [9]. In this paper we are interested in studying different schemes to improve the memory's efficiency in an EA. We propose two different mechanisms for storing individuals into the memory (replacing strategies) and will test them against VMEA and other memory-based EAs. Also, we will use a biologically inspired genetic operator called conjugation to control the population's diversity. This genetic operator is compared with uniform crossover and interesting conclusions about the population's diversity may be drawn.

The paper is organized as follows: section 2 presents previous work using the Variable-size Memory Evolutionary Algorithm (VMEA). Section 3 describes several replacement strategies found in literature and introduces two new replacing schemes. Section 4 describes the implementation of the conjugation operator. The experimental setup is explained in section 5. In section 6 we show the obtained results concerning algorithms efficiency, growth of the memory and the population's diversity. Finally, in section 7 we state the main conclusions and ideas for future work.

II. VARIABLE-SIZE MEMORY EVOLUTIONARY ALGORITHM

Simões and Costa [9] proposed an EA called VMEA – Variable-size Memory Evolutionary Algorithm, to deal with dynamic environments. This algorithm uses a memory population, 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 two boundaries. The basic idea behind VMEA is to use the limited resources (total number of individuals) in a flexible way. The size of the populations can change according to the evolutionary process, but the sum of the two populations cannot go beyond a certain limit. The memory is updated from time to time and if the established limits are not reached, the best individual of the

Manuscript received May 28, 2007.

A. Simões is with the Department of Informatics and Systems Engineering, ISEC, Coimbra Polytechnic, 3030 – 390 Coimbra, Portugal (phone: 351-239-790350; e-mail: abs@isec.pt

E. Costa is with CISUC, Department of Informatics Engineering, University of Coimbra, 3030 – 390 Coimbra, Portugal (phone: 351-239-790000; e-mail: ernesto@dei.uc.pt).

current population is stored into the memory. If there is no room to keep this new solution, then the best individual of the current population is introduced replacing one individual of the memory *chosen accordingly with a replacing scheme*. In that paper, an aging-based replacing strategy was used that will be explained in the next section.

The memory is evaluated every generation and a change is detected if at least one individual in the memory changes its fitness. Notice that the environmental changes occur very ***r*** generations but the algorithm has no information about when the next change will occur and so it is necessary to make its discovery every generation.

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 has reached the allowed maximum, the best individual in memory replaces the worst one in the current population. For a complete description of the algorithm see [9] and [10].

III. REPLACING STRATEGIES

The idea of adding a memory to an EA has been widely studied by a large number of authors. Memory can be used as a pool of good individuals that can be useful in future situations. For instance, in cyclic environments if the memory correctly updated, when an environment reappears, an individual belonging to the memory can avoid an abrupt decrease on the algorithm's performance when we transfer it to the evolving population. Memory can also be used to maintain diversity in the population, fostering a better readaptation of the EAs. Some questions can be drawn concerning memory: 1) when and which individuals to store in memory, 2) how many, 3) which ones should be replaced when memory is full, 4) which individuals should be selected from memory and introduced in the population when a change happens [1]. In this paper we will try to give an answer to the third question. 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 by ***replacing strategy***. Branke [1] compares a number of replacement strategies for inserting new individuals into the memory. The most popular is called *similar* and selects the most analogous individual currently in memory for replacement, as long as this individual is better. This replacing strategy was already used in several works ([1], [14], [15]).

Simões and Costa [9] proposed a replacing strategy based on the aging of memory individuals: all individuals of the memory start with age equal to zero and every generation their age is increased by one. Besides, if they were selected to feed the population when a change occurs its age is increased by a certain value. In the case a limit age is reached their age is reset to zero. When it is necessary to update the memory the *youngest* element is selected for replacement. The basic idea is to swap one individual with a poor contribution in the evolutionary process, i.e. one that was never selected to the population or that is in the memory for a long time. This approach was called by *age1*. The comparative analysis of *age1* method to the *similar* replacing scheme showed that there was no considerable improvement in the performance of the studied algorithms [10].

In this paper we propose two different replacing strategies to use memory in a more efficient way and thus confer to the EAs a better performance.

The first, will be called by ***age2***, is also based on the age of memory individuals. The age of an individual ***i*** is given by:

$$age_i(t) = age_i\,(t) + 1 + fitness_i * fit\_rate \qquad (1)$$

As suggested by [1], the individuals' age is computed every generation as a linear combination of their actual age and a contribution of their fitness (*fit_rate*).

Moreover memory individuals never die, i.e. their age is not reset to zero. This way, we do not penalize individuals that last long in memory.

Using this method the individual to replace is also the *youngest* one, considering that age was calculated using equation 1.

The second replacing strategy, will be called by ***generational***, and selects the worst individual present in the memory ***since the last change*** to be replaced. For instance, if the last change occurred at generation $\mathbf{t_1}$ and currently the algorithm is in generation $\mathbf{t_2}$, when it is time to insert an individual into the memory we will replace the worst individual that was stored between generation t_1+1 and t_2-1.

When it is time to update the memory, if no individual has been stored since last change, we use the *similar* strategy and substitute the closest individual in terms of Hamming distance, if it is worse than the current best. Therefore, in the case of VMEA, the maximum size of the memory is reached slower, the redundant information is minimized and the best information of a certain period of the evolutionary process is stored. This strategy, in next references, will be called by ***gen***.

IV. BACTERIAL CONJUGATION

It is widely accepted in the Evolutionary Computation community that genetic operators are essential to the efficiency of EAs. They allow creating new individuals, from the genetic information of previous ones, so the algorithm will in general converge to the desired solution. Traditionally crossover is used as the main genetic operator. Nevertheless, other biologically inspired operators have been proposed and tested with some degree of success. These new genetic operators were applied either in static [7], or dynamic environments [8].

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 [6].

Computational conjugation was introduced independently by Harvey and Smith. Smith [12] 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 [3] investigated a tournament-based conjugation: two parents are selected at random, and the winner of the tournament (fittest individual) 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.

Simões and Costa [9] used conjugation in the context of dynamic environments. The authors applied conjugation after selecting the individuals 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 n/2 best individuals become the 'donor' while the remaining becomes the 'recipient' (***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). Two points randomly chosen control this injection. 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**. Fig. 1 shows how conjugation is applied to one pair of individuals of the mating pool.

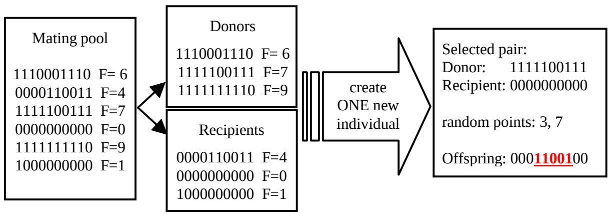


Fig. 1. Creating a new individual using conjugation

The obtained results in dynamic optimization problems using this genetic operator were quite promising [9].

V. EXPERIMENTAL DESIGN

A. Dynamic Test Environments

The dynamic environments used to test our approaches were constructed using Yang's Dynamic Optimization Problems (DOP) generator. 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 $x \oplus M$ 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 $f(x, t+1) = f(x \oplus M, t)$. Using the DOP generator the characteristics of the change are controlled by two parameters: the speed of a change, ***r***, which is the number of generations between two changes, and the magnitude of a change, ρ that consists in the ratio of ones in a temporary template that is created for each environment and integrated to the mask **M**. The more ones in this template, the more severe is the change. The DOP generator 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. More details about Yang's DOP generator can be found in [16].

In this paper we constructed 20 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, 1.0 (severe change). In order to study the behavior of the algorithm in randomly changing environments we also set ρ to a uniformly randomly generated value in the interval [0.01, 0.99] (called by *rnd*).

To test and compare the several replacing schemes combined with two different genetic operators, we selected two benchmark problems: the knapsack problem (100 items) and OneMax problem (300 bits).

1) 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 [4]. 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. Details about the implemented knapsack, including the penalty function can be found in [10].

2) 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.

B. Experimental Setup

1) Algorithms' parameters

Previous work compared VMEA using the ***age1*** replacing strategy with other algorithms: the random immigrants' algorithm (RIGA) [2] and the memory-enhanced GA (MEGA) studied in [13]. The results proved that VMEA, using either crossover (VMEA Cx) or conjugation (VMEA Cj), generally outperformed the other approaches. A typical comparative graphic of the behavior of VMEA and the other studied algorithms is shown in Fig. 2. To see more results consult [10].

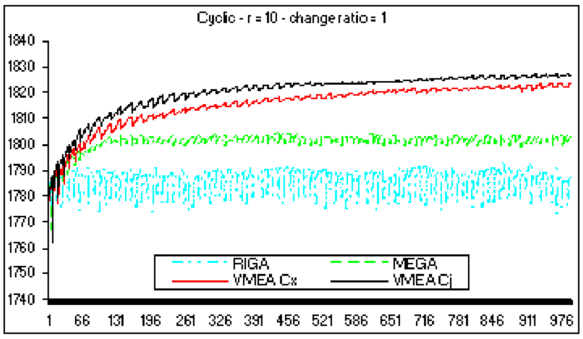


Fig. 2. VMEA *vs* MEGA and RIGA. Typical curve of the algorithms in the dynamic Knapsack problem

In this paper we will be focused in the performance of memory-based EAs using the proposed replacing schemes. Besides the above mentioned algorithms we will also introduce the Memory-based Immigrants Genetic Algorithm (MIGA) proposed in [14]. We will compare the algorithms' efficiency using the following methods: similar (*sim*), proposed by Branke in [1], *age2* and *gen*. We will also combine the conjugation operator with the replacing methods to make some conclusions about the efficiency of this genetic operator in changing problems and study its impact in the population's diversity. The comparison of the VMEA using the *similar* method and the originally proposed in [9] called *age1* can be found in [10]. The different VMEAs will be designated by; ***VMEA-age2Cx***, ***VMEA-age2Cj***, ***VMEA-genCx***, ***VMEA-genCj***, ***VMEA-simCx*** and ***VMEAsimC***j. The originally proposed ***MIGA*** and ***MEGA*** will be referred the same way and use the similar replacing method and uniform crossover [14], [13]. The other variants will be called using the same notation used in VMEAs: ***MIGA-age2Cx, MIGA-age2Cj, MIGA-genCx, MIGA-genCj***. Same designations will be used for MEGA: ***MEGA, MEGA-age2Cx, MEGA-age2Cj, MEGA-genCx, MEGA-genCj***.

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 pc=0.7 (the same probability was used with conjugation) and mutation applied with probability pm=0.01. In VMEA, the search population starts with 100 individuals, the memory starts with 10 individuals. Their sizes are variable, changing along time, but the sum of the two populations cannot surpass 120. The *age2* strategy, was computed according to equation (1), with *fit_rate* set to 0.1 (value chosen after some preliminary experimentation). MIGA and MEGA use populations of 110 individuals and a memory of 10 individuals. For MIGA, the ratio of immigrants was set to 0.1 and the mutation rate to create the immigrants was 0.01. For each experiment of an algorithm, 30 runs were executed and the number of environmental changes was 200 with r=10 (2000 generations), 80 with r=50 (4000 generations) and 40 with r=100 and 200 (4000 and 8000 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:

$$Foverall = \frac{1}{G}\sum_{i=1}^{G}\left[\frac{1}{N}\sum_{j=1}^{N}F_{bestij}\right] \qquad (2)$$

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

VI. Experimental Results

The amount of data, graphics and tables produced in this comparative study was enormous and in this paper we can only show part of the results, using representative graphics of each analyzed situation. All information concerning this work can be consulted in [11]. The statistical validation of the results was made using paired one-tailed t-test at a 0.01 level of significance. The validation tables comparing all the approaches are also available in the referred document. In the next sections we show the results concerning: (1) the efficiency of the algorithms using the different replacing strategies, (2) the diversity levels measured in population and memory and (3) the observed results in the growth of the population and memory sizes using the different schemes for VMEA.

A. Comparing Replacing Strategies

The global results for the different VMEAs are plotted in Fig. 3 (Knapsack) and Fig. 4 (OneMax) and for the different MIGAs and MEGAs in Fig 5 and Fig. 6 (Knapsack). Analyzing the results on both problems, we can see that for the VMEAs, the proposed *gen* replacing scheme always obtained the best results, except when r=10. This happens because the change period is small and when a change occurs, most of the times, no individual of that period was already stored. So, we replace the most *similar*. In this case, the best replacing strategy was the proposed *age2*. The *age2* scheme compared with the remaining was not so effective. In fact, the results show that, *age2* performs better in small change periods and when combined with conjugation. These results confirm the difficulty of finding a tradeoff between the fitness and age contributions, as suggested in [1]. In MIGAs and MEGAs ***gen*** methods substantially improves the performance in environments of small periods and with severe changes. The ***age2*** method was also superior in most of the times, but the enhancement introduced is not as visible as in the ***gen*** scheme. Conjugation improves the performance in the analyzed VMEAs just as observed before [9], but has no influence in the studied MIGAs and MEGAs. Observing Fig. 8 to 11 we confirm that using the proposed *gen scheme* the different algorithms considerably improved their performances. The information stored in memory allows a continuously adaptation of the analyzed EAs, contrary to the remaining replacing strategies. The use of population and memory with variable sizes combined with the ***gen*** replacing method and conjugation (VMEA-genCj) was the algorithm that globally achieved the best performance. MIGAs and MEGAs, due to the restriction of memory's size, even using the ***gen*** scheme, have a degradation of its performance mainly in environments with larger change periods, since the replacing is more often and the destruction of good individuals occurs frequently.

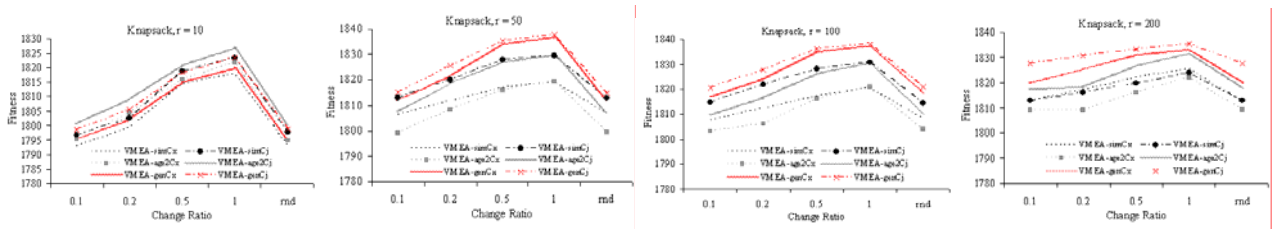


Fig. 3. VMEA: Results on dynamic Knapsack problem, with different change ratios (ρ) and r=10, 50, 100 and 200

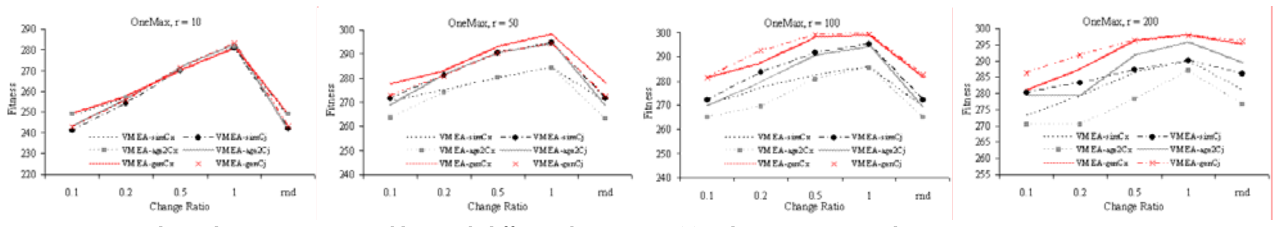


Fig. 4. VMEA: Results on dynamic OneMax problem, with different change ratios (ρ) and r = 10, 50, 100 and 200

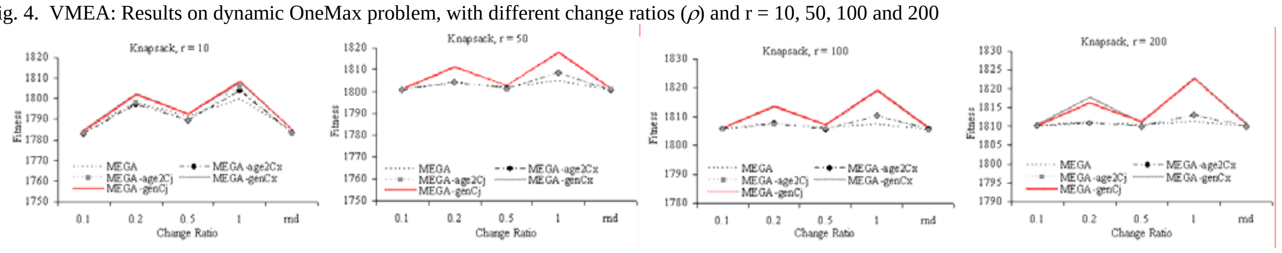


Fig. 5. MEGA: Results on dynamic Knapsack problem, with different change ratios (ρ) and r = 10, 50, 100 and 200

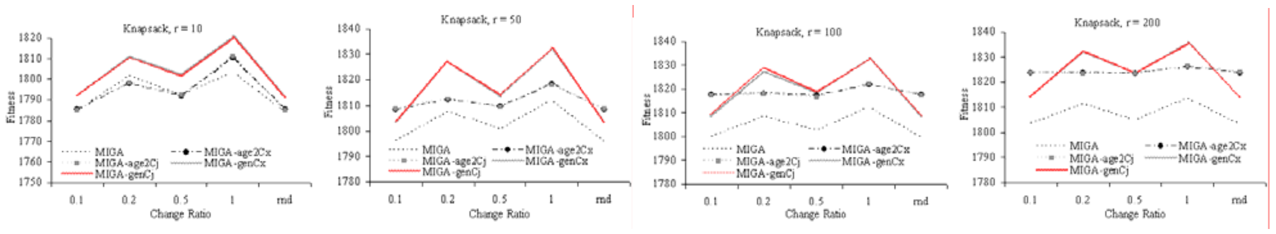


Fig. 6. MIGA: Results on dynamic Knapsack problem, with different change ratios (ρ) and r = 10, 50, 100 and 200

B. Population's and Memory's Diversity

In order to study the impact of the genetic operators in the diversity dynamics, we stored the population and memory diversity at every generation using the standard measure based on the Hamming distance. At generation *t*, the diversity is given by:

$$Div(t)=\frac{1}{N}\sum_{k=1}^{N}\left[\frac{1}{l.n(l-1)}\sum_{i=1}^{n}\sum_{j\neq i}^{n}HDij(k,t)\right] \qquad (3)$$

where **N** is the number of runs, ***l*** is the length of the chromosomes, ***n*** is the population (or memory) size at generation *t* and $\boldsymbol{HD_{ij}(k,t)}$ is the Hamming distance between individuals *i* and *j* at generation *t*, in the k^{th} run. Analyzing the recorded values for population's and memory's diversity, we conclude that the conjugation operator always kept lower diversity levels in the population. Nevertheless, very often the best results are achieved using the conjugation operator! As an example, we show the diversity of memory and population stored on the studied benchmarks, with *r*=50 and ρ=0.2 using the *gen* replacing scheme. As we can see in Fig. 7, the algorithm using the *gen* scheme and conjugation obtained the best results. From the analysis of the results it follows that the population's and memory's diversity levels *is inferior when using conjugation*. This is also true for the remaining strategies and for different values of *r* and ρ.

These results show that the usual idea that in dynamic environments it is crucial to maintain high diversity in the population to improve the adaptability of algorithm is not always true

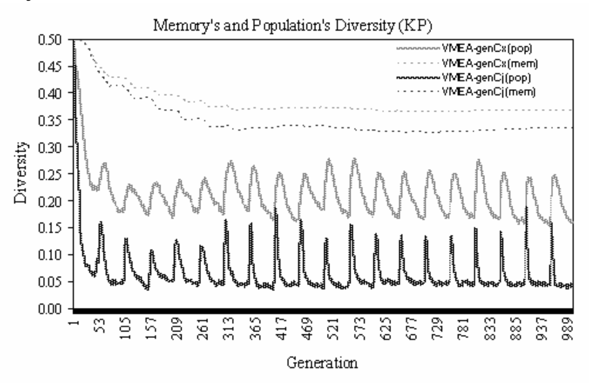


Fig. 7. Population's and Memory's diversity for the first 20 environmental changes in dynamic Knapsack

One reason to this fact maybe because conjugation is less disruptive, since the information about the best individuals is always preserved and that appear to be positive in the studied problems. We are making more experiments in order to provide more consistent conclusions.

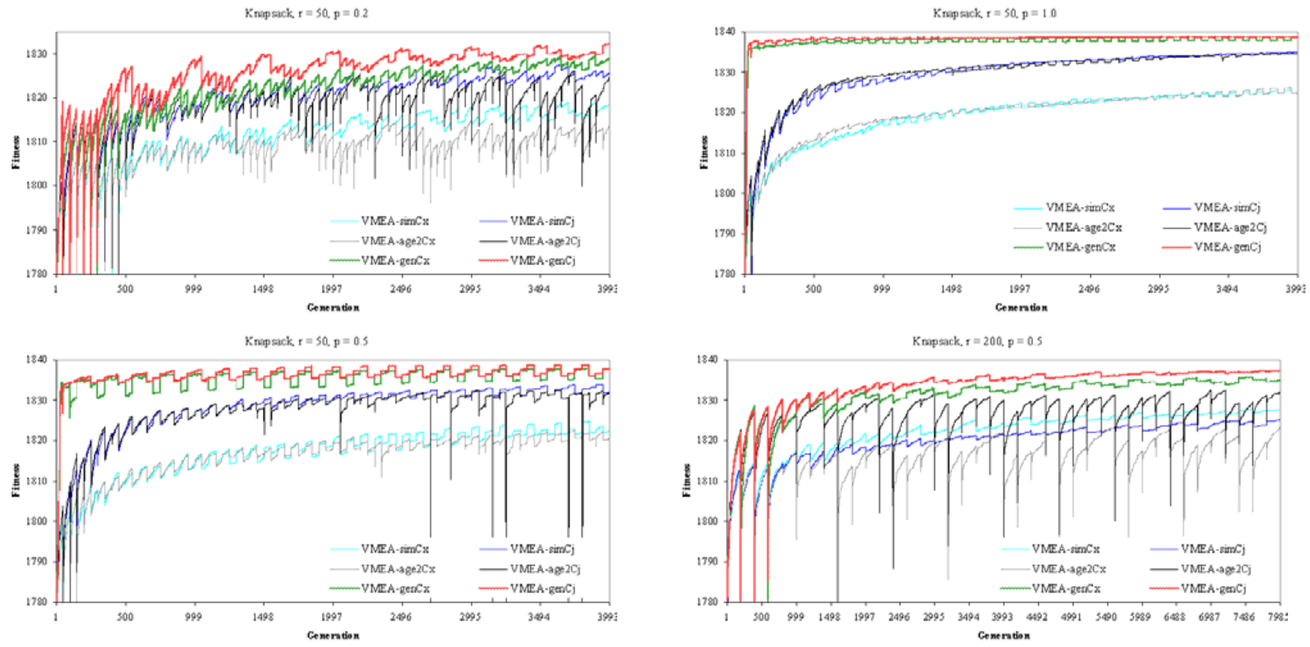
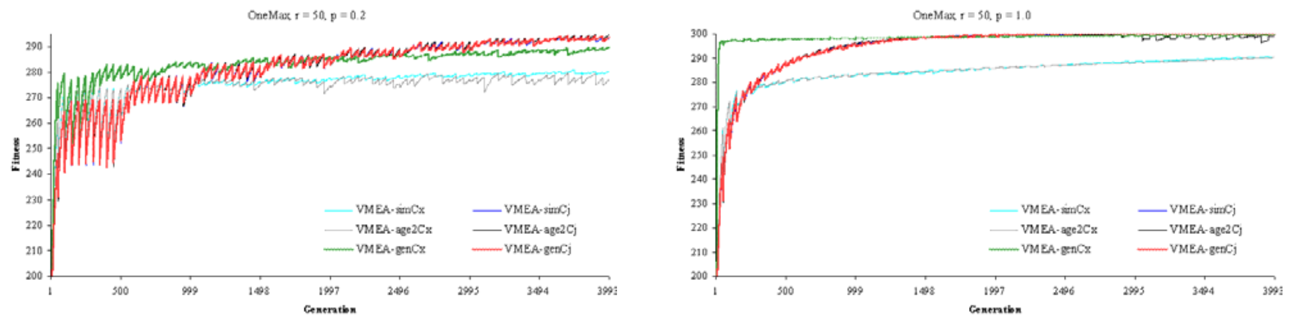


Fig. 8. Dynamic behavior of VMEAs on dynamic Knapsack, with r = 50 and r = 200, when ρ = 0.5, ρ = 1.0



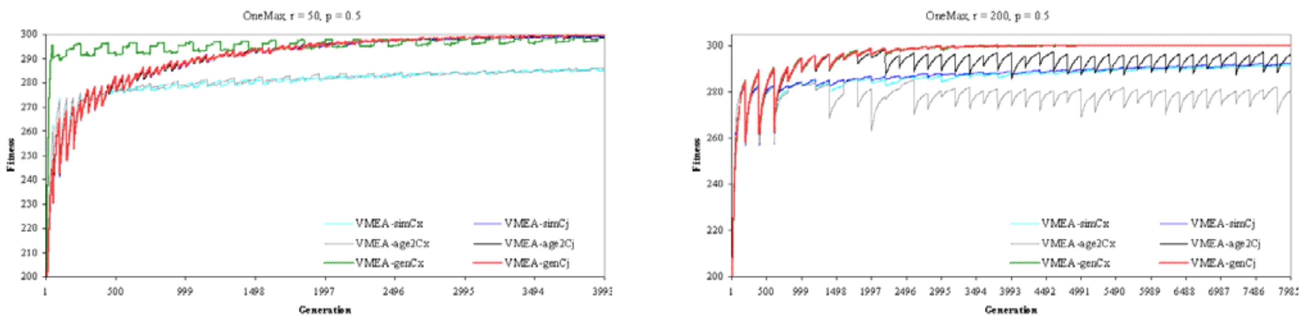


Fig. 9. Dynamic behavior of VMEAs on dynamic OneMax, with r = 50 and r = 200, when ρ = 0.5, ρ = 1.0

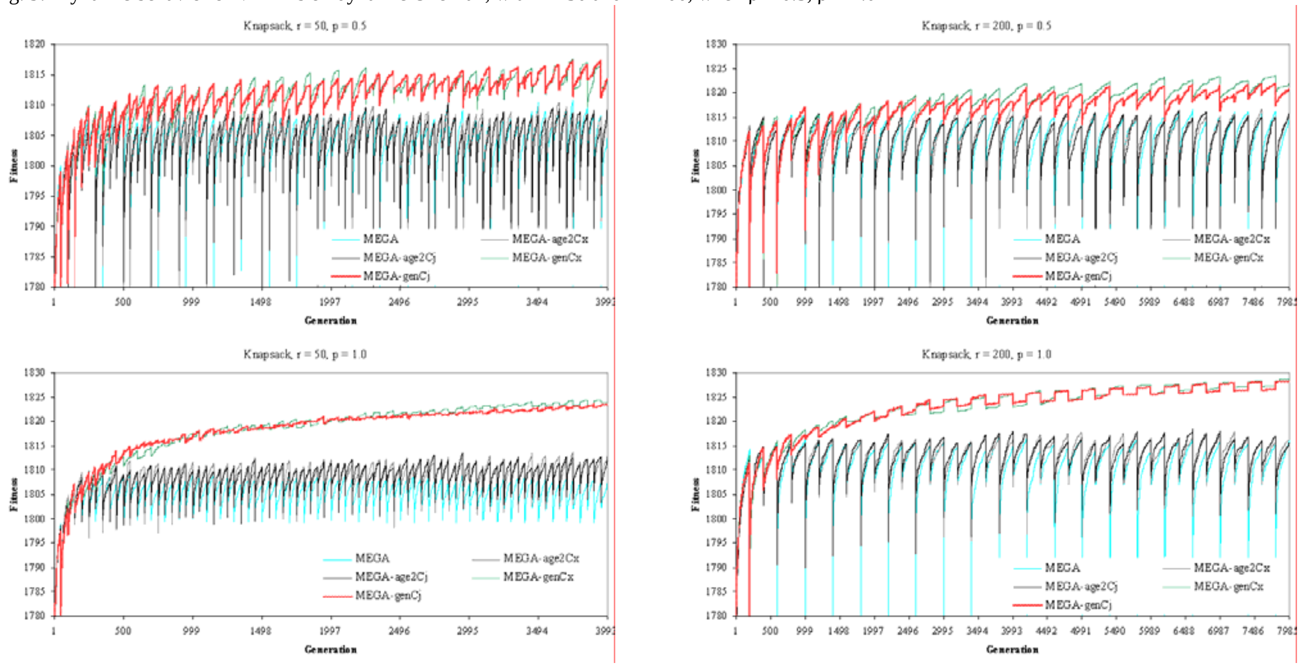


Fig. 10. Comparing MEGAs on dynamic Knapsack, with r = 50 and r = 200, when ρ = 0.5, ρ = 1.0

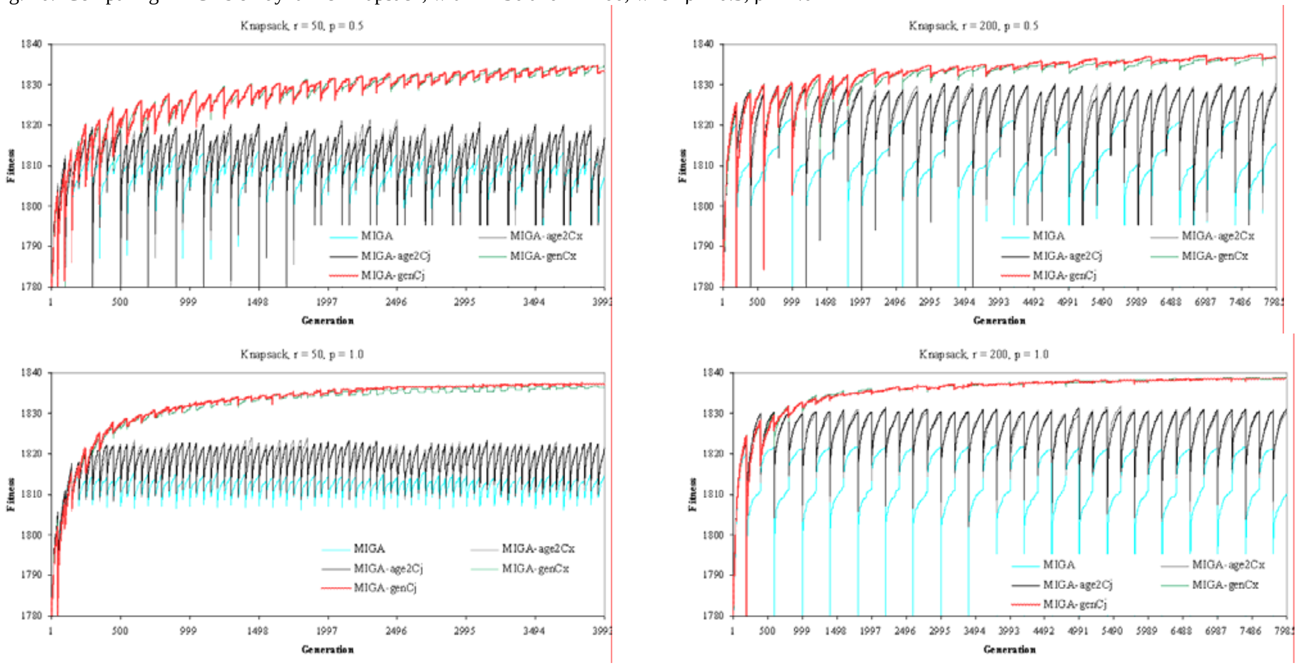


Fig. 11. Comparing MIGAs on dynamic Knapsack, with r = 50 and r = 200, when ρ = 0.5, ρ = 1.0

C. Memory and Populations Sizes in VMEA

The results concerning the growth of memory's and population's sizes confirm our conjecture that using the *gen* replacing scheme the memory should increase slower. Actually, this was observed in all studied DOPs. Fig. 12 shows a typical example of how population and memory grow using the different replacing schemes. We can see that, with *gen* replacing strategy (black lines), the memory's size increased gradually and its maximum is only attained at the end of the evolutionary process.

By replacing the worst individual memorized since the last environmental change we save space (in the case of VMEA) for good individuals that may appear in future generations, minimizing the substitution of helpful information. The results confirm our initial assumption that the memory's usage can be improved if the replacing scheme is chosen carefully.

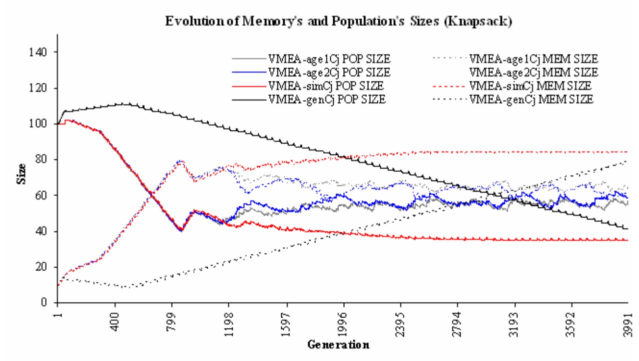


Fig. 12. Population's and Memory's sizes on dynamic Knapsack

VII. CONCLUSIONS

This paper proposed two new replacing strategies and tested its efficiency against several memory-based EAs. A genetic operator called conjugation was used combined with the proposed schemes. An experimental studied was carried out using different dynamic optimization problems based in two benchmark problems. From the obtained results several conclusions can be drawn: First, the *age2* replacing scheme obtained better results in dynamics with small period changes if combined with conjugation. For the remaining cases, the ***gen*** replacing scheme was superior. Second, in the studied VMEAs, conjugation typically outperformed uniform crossover (few exceptions were found). Third, the proposed *gen* replacing scheme provided excellent improvement in the studied algorithms, minimizing (or even abolishing) the decrease of individual's fitness after a change. These results prove that when dealing with dynamic environments the information stored in memory is crucial to the effectiveness of the algorithms. Since the memory's capacity usually has limited size, this approach of replacing the individuals proved to be an excellent choice. To fundament our conclusions we are currently testing the proposed ideas in other algorithms, using different problem benchmarks and including environments with noise.

ACKNOWLEDGMENT

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 High Education.

REFERENCES

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

[2] 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.

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

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

[5] 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.

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

[7] A. Simões and E. Costa. On Biologically Inspired Genetic Operators: Transformation in the Standard Genetic Algorithm. In Spector, L., E. Goodman, A. Wu, W.B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke, (eds). 2001 Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001. pp. 584-591, San Francisco, USA, 7-11 July, CA: Morgan Kaufmann Publishers, 2001

[8] 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.

[9] 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.

[10] A. Simões and E. Costa. VMEA: Studies of the impact of different replacing strategies in the algorithm's performance an in the population's diversity when dealing with dynamic environments. CISUC Technical Report TR2007/001, ISSN 0874-338X, February 2007.

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

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

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

[14] 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.

[15] S. Yang. A Comparative Study of Immune System Based Genetic Algorithms in Dynamic Environments. Proc. of the 2006 Genetic and Evolutionary Computation Conference, pp. 1377-1384, ACM Press, 2006.

[16] S. Yang. Associative Memory Scheme for Genetic Algorithms in Dynamic Environments. Applications of Evolutionary Computing, LNCS 3097, pp. 788-799, Springer Verlag Berlin, 2006.