Variable-size Memory Evolutionary Algorithm to Deal with Dynamic Environments: an empirical study

Anabela Simões[1,2], Ernesto Costa[2]

[1]Dept. of Informatics and Systems Engineering
ISEC - Coimbra Polytechnic
Rua Pedro Nunes - Quinta da Nora
3030-199 Coimbra – Portugal

[2]Centre for Informatics and Systems of the University of Coimbra
Pólo II – Pinhal de Marrocos
3030 - 290 Coimbra – Portugal
abs@isec.pt, ernesto@dei.uc.pt

CISUC TECHNICAL REPORT TR 2006/04 - ISSN 0874-338X **NOVEMBER 2006**

Abstract. When dealing with dynamic environments two major aspects must be considered in order to improve the algorithms' adaptability to changes: diversity and memory. Integrating these two mechanisms in Evolutionary Algorithms we can enhance their performance. In this paper we propose and study a new EA 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 lengths allowed and their sizes change according with the stage of the evolutionary process: if a 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 in the biological process of conjugation, is proposed and used 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.

Keywords: Evolutionary algorithms, memory, diversity, dynamic environments

1 Introduction

Evolutionary Algorithms (EA) have been used with success in a wide area of applications. Traditionally, EA are well suited to solve problems were the environment is static. The generational process of evolution often leads the EA to the best solution. However, most of real-world applications are non-stationary 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 EA 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 to the population 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,(e.g. triggered hypermutation [4,14], random immigrants [8], new genetic operators [22],[23] or tag bits [11]), 2) using memory schemes either implicit (e.g. [5], [7], [9], [15], [18] and [19]) or explicit (e.g. [1], [16], [23], [26], [29], [31] and 3) using multi-populations (e.g. [3], [27]).

Recent works simultaneously studied with success the combination of memory schemes and mechanisms for promoting and maintaining diversity [23], [28].

In this paper we are also interested in study 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. The population's size can increase in several ways: until reach its maximum or until reach the total's maximum. If one of these situations occurs, there are possible reactions of the algorithm: replace the worst individuals of population or clean the memory.

Besides this memory framework, we introduce a new way of using the biologically inspired conjugation operator, which improves quality of the solutions. Conjugation will be used as the main recombination operator. This mechanism is applied after the matting pool is selected. The best individuals of the pool will transfer part of its 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.

The paper is organized as follows: section 2 presents an overview about past research using memory schemes and promotion and maintenance of diversity in dynamic environments. Section 3 explains in detail how the population and the memory of the EA change its size moving individuals from the latter to the former. Section 4 describes the implementation of the conjugation operator. The experimental setup is explained in section 5 and, finally, in section 6 we state the main conclusions and ideas for future work.

2 Relevant background

In the following we will give a short overview about different proposals involving memory schemes and/or mechanisms to promote and maintain diversity in the context of dynamic environments.

4.1 Memory schemes

The most known approach using redundant representations is to use diploid instead haploid chromosomes. This idea was suggested by [7] as an extension of the standard GA. Later, other authors investigated the idea of diploidy, particularly its use in the context of dynamic environments (see [5], [9], [15], [18], [19].) The redundant information when using a diploid representation acts as a memory for remembering past solutions and promotes diversity in the population. In diploid chromosomes there are two genes to represent a certain characteristic, but only one of them is expressed in the phenotype. This is controlled by a chosen dominance mechanism.

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 jn one step, which would not be possible with common genetic operators [2].
Different approaches have been proposed in the literature. For instance, [16], introduced an algorithm which stores in a memory the good candidate solutions for a robot controller together with information about the environment. If a similar environment reappears in future then the corresponding stored solution is activated.
[12] proposed an EA applied to scheduling. The idea behind this work is that when a change in the environment occurs, the population is restarted using information of the population last run. [26] investigated an algorithm in which individuals store information about their ancestors. When a change is detected, the current and the past solutions are re-evaluated and the best one is activated. [1] proposed an EA with memory which stores useful information about the environment. As the memory has a fixed size, several replacement strategies were proposed in this paper. [23] introduced an EA inspired in the immune system in which memory is similar to the functioning of B-Cells in human body. When a change is detected (an invasion occurs), a B-Cell with high affinity is retrieved from memory, hypermutated, cloned and introduced in the population in order to help the algorithm to readapt to the new conditions. A memory-enhanced GA was proposed by [31] and uses a fixed size memory to store the best individual of the population. When the environment changes, the memory is merged with the old population and the best individuals are selected to undergo evolution. This author also investigated an associative memory scheme, where information of the environments is kept associated with the best individual. So, when the environment changes this information is used to generate a new chromosome [29].

4.2 Promoting Diversity

Several techniques have been proposed in order to preserve the diversity in a population. Standard EAs because of selective pressure are biased towards a particular region of the search space. This fact implies a premature convergence of the population and must be avoided, especially if we are dealing with dynamic problems. The most popular techniques to promote populations' diversity are hypermutation [4], [14], which consists in changing the mutation rate when a change is detected. Other approaches such as the random immigrants [8], tag bits [11], or alternative genetic operators [22] aim to maintain or increase population's diversity in 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 [23], [28].

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 play 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 is suggested in [1] and [23]) and its updated at time TM=t + rand(5,10), the same way as in [23].

If an environmental modification is detected, the best individual of the memory is introduced in the population. In the case of either the population's size or the sum of the two populations have reached the allowed maximum, the best individual in memory replaces the worst one in the current population.

Fig. 1, in the next page, describes in pseudo-code the general functioning of VMEA.

4 Promoting Diversity in the Search Population

Traditionally, EAs use crossover as the main genetic operator. In the past other biological inspired operators have been proposed and tested with some degree of success . These new genetic operators were applied either in static [6], [10], [20], [21] or dynamic environments [22], [23], [28].

When dealing with dynamic environments, the main purpose of using different biological operators is to maintain the population's diversity in order to allow a better performance of the EA when a change is detected.

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 it cannot be considered actually sexual, as it does not involve the fusing of gametes and the creation of a zygote. It is merely the transfer of genetic information from a donor cell to a recipient. In order to perform conjugation, one of the bacteria, the donor, is the host to a conjugative plasmid [17].

```
pop_size = POP_MAX
mem_size = MEM_MIN
Initialize Memory Randomly
Initialize Population Randomly
T_M = rand(5,10)
t=0;
repeat
        Evaluate memory
        Evaluate population
        Select mating pool
        Recombination
        Mutation
        if is time to update memory then
          T_M = t + rand(5,10)
           Select best individual of the population
           if there is room to one more individual in memory then
                Store best individual in memory
                Increase mem_size
          else
                if cleanMemory is successful then
                     Update mem_size
                     Store best individual in memory
                else
                     Replace youngest individual of memory (if best)
        if change is detected then
           Select best individual from memory
           if there is room to one more individual in population then
                Store individual in population
                Increase pop_size
           else
                if cleanMemory is successful then
                     Update mem_size
                     Store individual in population
                else
                     Replace worst individual of population
until stop_condition
```

Fig. 1. Pseudo-code of the Variable Memory Evolutionary Algorithm (VMEA)

Computational conjugation was introduced independently by Harvey and Smith (see [10], [24], [25]). Smith used bacterial conjugation to neighbouring individuals placed in a square matrix in the context of solving hard satisfiability problems. In simple conjugation two individuals of the matting pool are randomly chosen and who is the donor and the recipient is also decided in a random manner. Then, two points are defined and the donor gives the genes bounded by those points to the recipient [25]. Harvey 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 [10]. 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 in two groups: the $\boldsymbol{n}/2$ best individuals become 'donor', the remaining, becomes '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**. Fig. 2 shows how conjugation is applied to one pair of the mating pool.

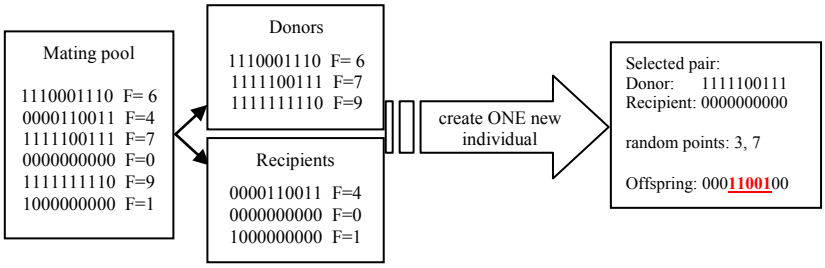

Fig. 2. Create a new individual using conjugation

5 Experimental study

5.1. Dynamic test environments

To test the performance of the proposed EA we constructed several dynamic environments using Yang's DOP generator [29, 32]. 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 $f(x, t+1) = f(x \oplus M)$ [30])

For instance, suppose we have the individual **x** = 11111 which in the *OneMax* problem has the maximum fitness. But if the environment changes and the mask **M** = 11111 is applied to **x**, then it becomes **x** = 00000 with fitness zero.

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 [32].

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. This allows for characteristics of 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 whenever is time to change the environment. [29].

5.2. Experimental Setup

We selected two benchmark problems to test our VMEA to make comparisons with other algorithms easier: the knapsack problem (100 items) and *oneMax* problem (300 bits).

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:

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

subject to the weight constraint:

$$\sum_{i=1}^{m} w_i x_i \leq C \quad (2)$$

We used a knapsack problem with 100 items using strongly correlated sets of randomly generated data constructed in the following way [13, 30]:

$$w_i = \text{uniformly random integer } [1, 50] \quad (3)$$

$$v_i = w_i + \text{uniformly random integer } [1, 5] \quad (4)$$

$$C = 0.6 \times \sum_{i=1}^{100} w_i \quad (5)$$

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

$$f(x) = \begin{cases} \sum_{i=1}^{100} v_i x_i, if \sum_{i=1}^{100} w_i x_i \leq C \\ 10^{-10} \times \left[\sum_{i=1}^{100} w_i - \sum_{i=1}^{100} w_i x_i \right], else \end{cases} \quad (6)$$

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.

Algorithms' Parameters

Three kinds of environments were created from each of these two base problems, using Yang's 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***).

To compare our approach we used two other algorithms: the random immigrants' algorithm [8] and the MEGA algorithm proposed by [31]. VMEA was tested using conjugation (VMEA-Cj) and uniform crossover (VMEA-Cx), in order to conclude about the efficiency of the proposed genetic operator in non-stationary 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 was set to 100 individuals. This value was used as the maximum approved size for the population in VMEA. In MEGA, the memory was used with size 10, and updated according the description given in [31]. 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. But the total of individuals in the two populations could not surpass 120. The age limit for the individuals in memory was set to ***G/2***, where $\boldsymbol{G}$ is the total number of generations.

For each experiment of an algorithm, were executed 30 runs 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). To compare the algorithms we stored the best-of-generation fitness every generation.

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:

$$Foverall = \frac{1}{G}\sum_{i=1}^{G}\left[\frac{1}{N}\sum_{j=1}^{N}F_{bestij}\right] \quad (7)$$

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, most of the times, 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. 3 and 4 plots the average of the best-of-generation fitness obtained in the knapsack and in onemax problem, respectively.

		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 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	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		++	++	++	++	++	--	--	--	--	--	++	--	--	+	--

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

In *cyclic environments*, our approach obtained 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 is cyclic environments. When the ratio is 1, the memory allows the algorithm to continue evolving when a change occurs. Also, conjugation shows best 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.

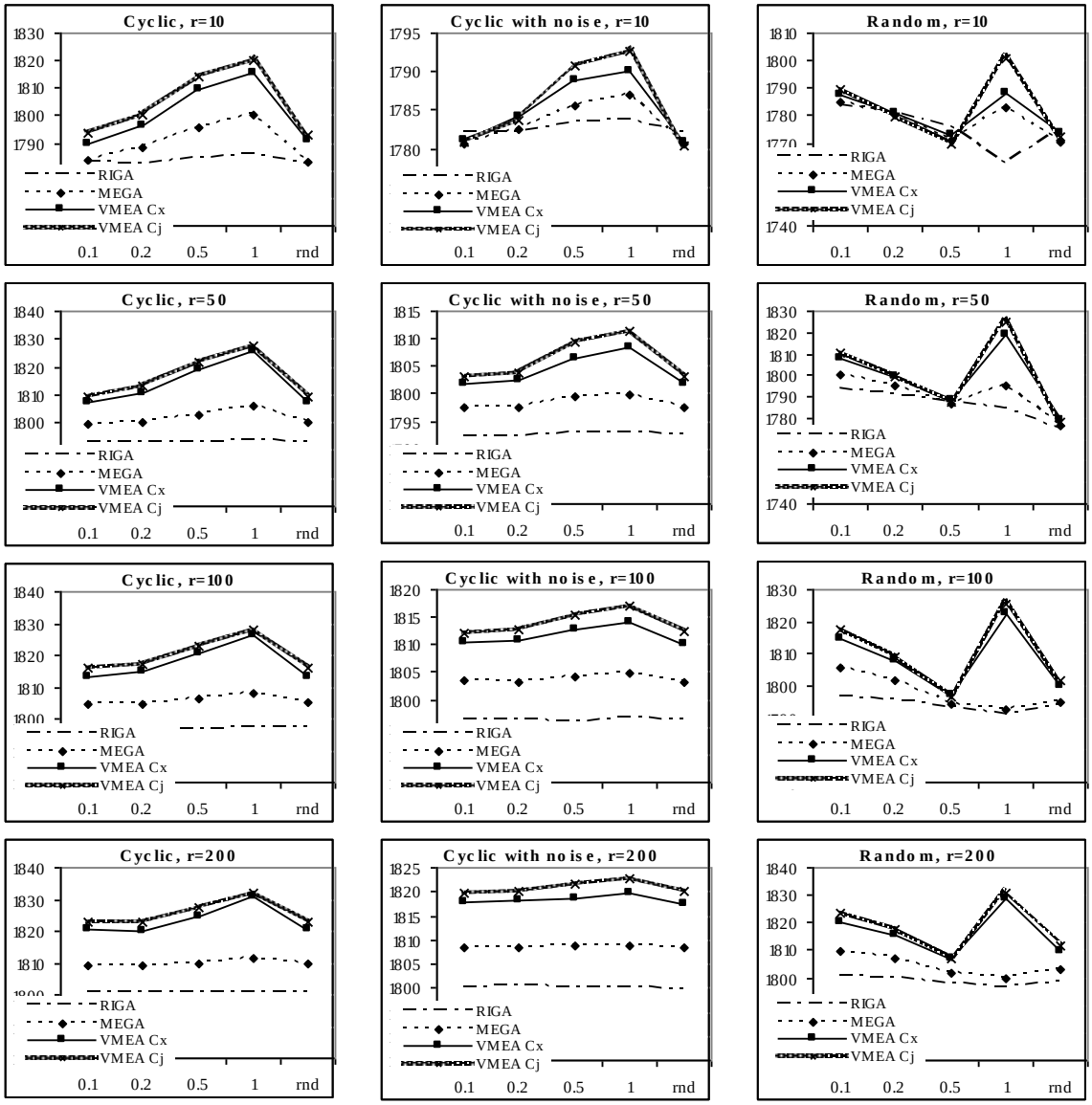

Fig. 3. Global results for the knapsack problem.

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. 5 shows the evolutionary behavior of the algorithms in cyclic environments for ρ=1, 0.5, 0.2 and 0.1, with r=10 and r=200, in the knapsack problem. Fig. 6 show equivalent results in the onemax problem..

The VMEA algorithm combined with the conjugation operator performed better in the knapsack problem. In fact, VMEA-Cj was, normally, the best algorithm. The same was not observed in the onemax problem. In these cases, VMEA-Cj performed better in cyclic environments and with larger change periods.

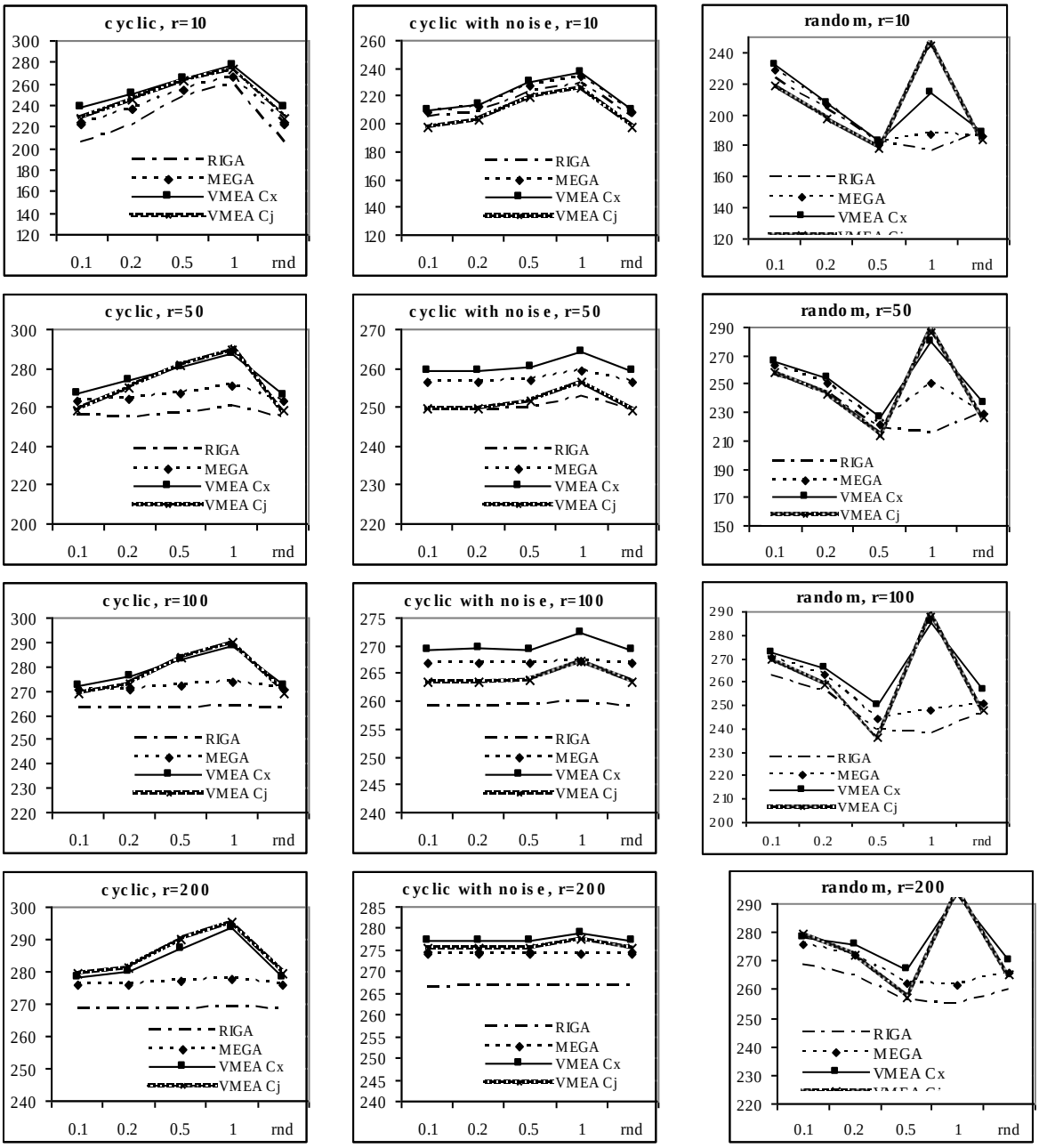

Fig. 4. Global results for the onemax problem

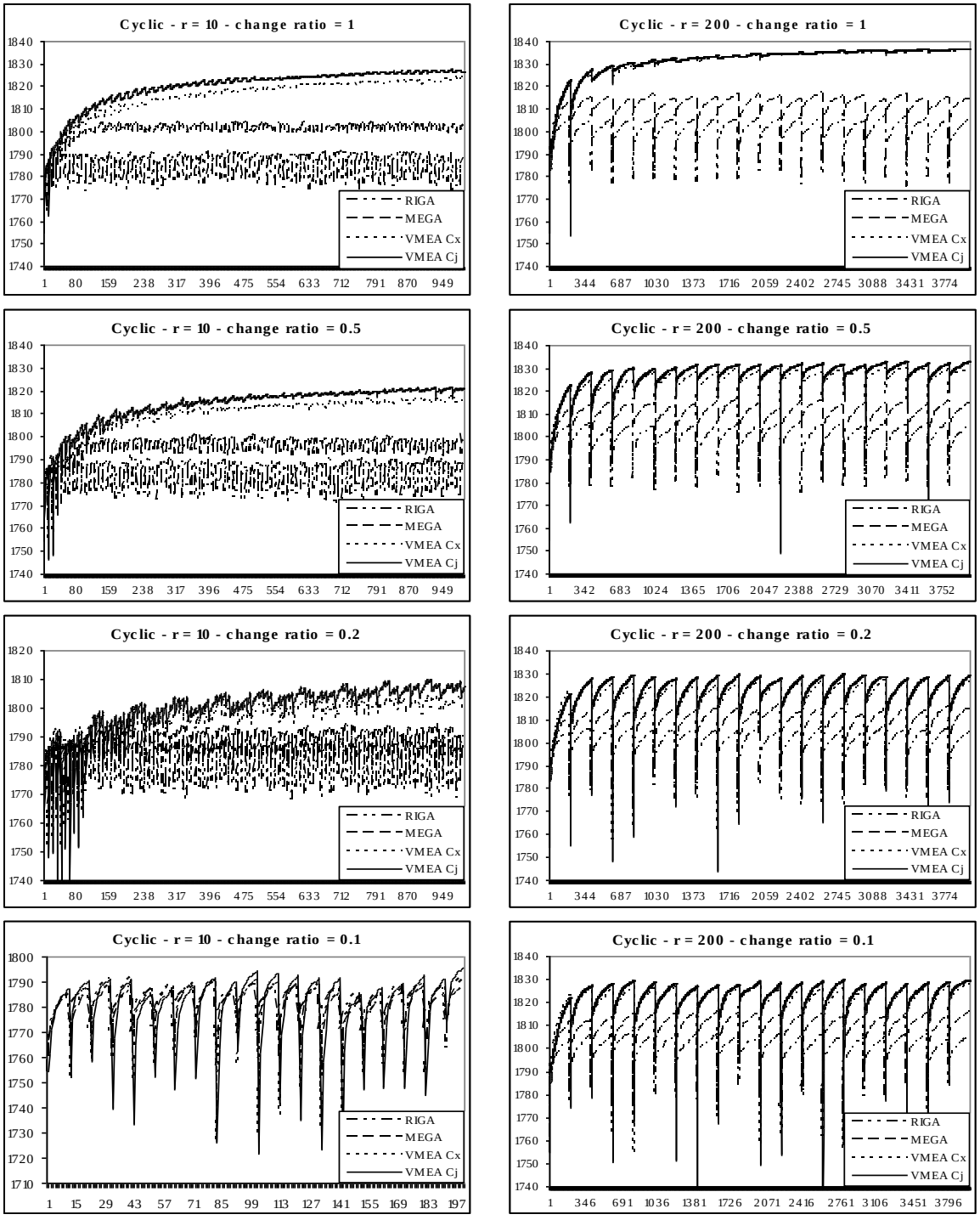

Fig. 5. Dynamic behavior of the algorithms in cyclic environments, for the *knapsack problem*

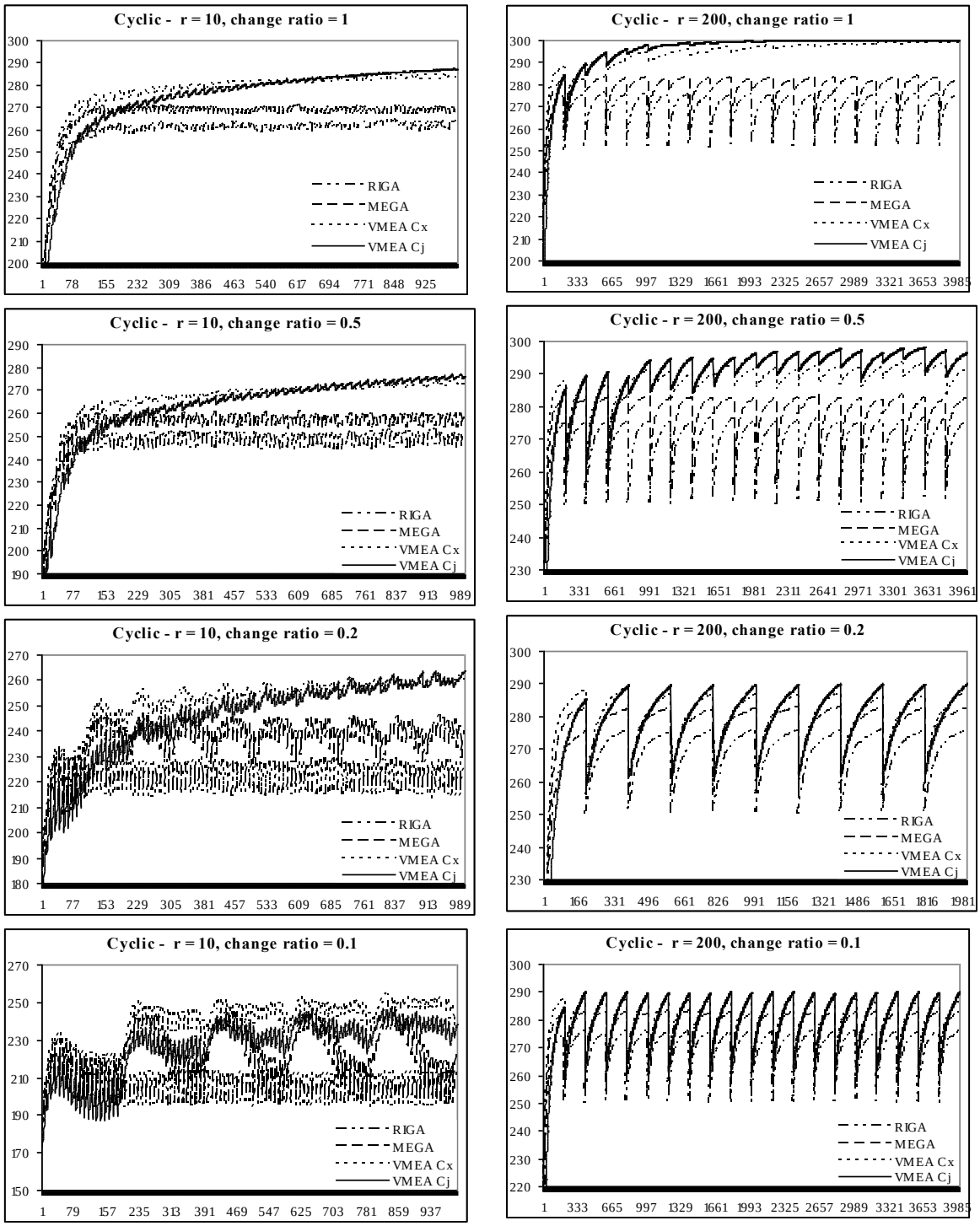

Fig. 6. Dynamic behavior of the algorithms in cyclic environments, for the *onemax problem*

In *random environments*, with high ratio changes (ρ=1), VMEA achieved very good results. The memory allows the algorithm to continuously improve its performance. This good performance is not as good as we decrease the change ratio. The new environment is slightly different from previous one, but repeated states appear later on and so memory has already lost the related information. Fig. 7 and 8 show some results obtained in random environments in both studied problems.

For random change ratio (ρ = *rnd)* we observed a degradation of the results. In this case, RIGA and the other two memory-based algorithms perform in a very similar way: after a change in the environment the

best-of generation fall for lower values and it is required some time to the algorithms start evolving again. Even so, VMEA, typically arise the best marks.

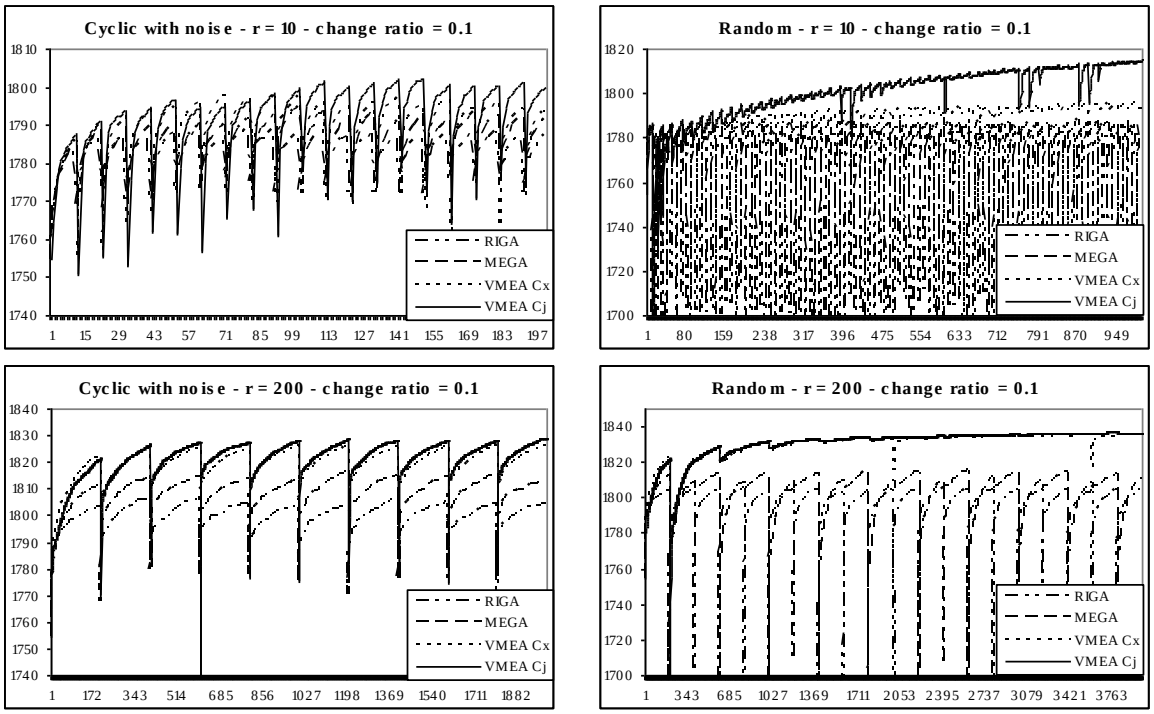

Fig. 7. Dynamic behavior of the algorithms in cyclic with noise and random environments (knapsack).

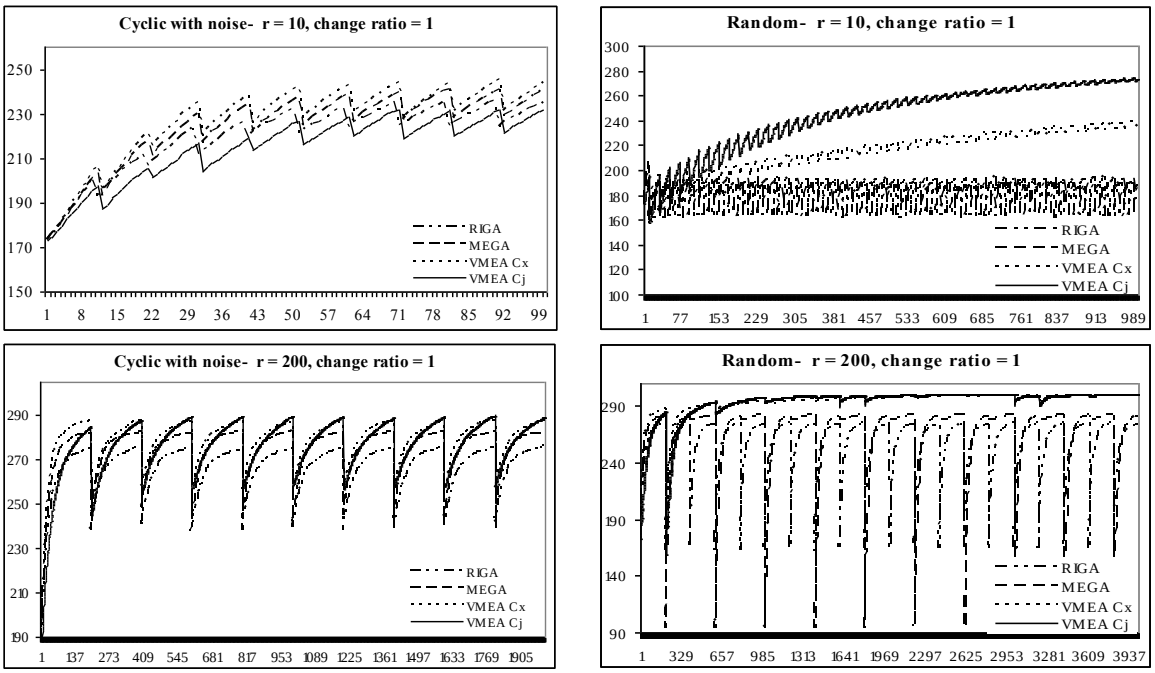

Fig. 8. Dynamic behavior of the algorithms in cyclic with noise and random environments (onemax).

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 its 'penalized' because we ran 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 were cleaned. After the maximum value for the memory is reached, the process of deleting individuals with same genotype from memory allows an increase in the population size. Fig. 9 shows a representative graphic of the evolution of the populations' size.

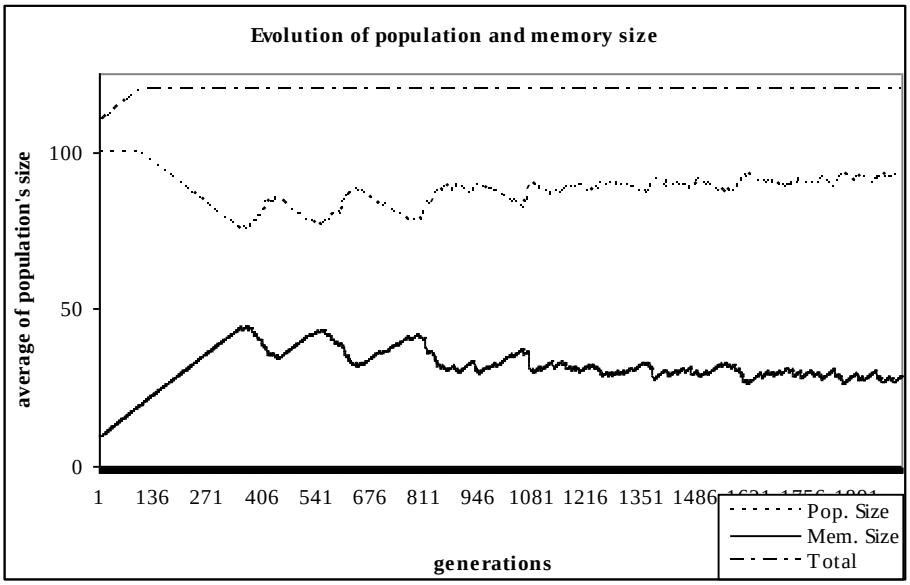

Fig. 9. Evolution of the population's and memory's size.

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 diversity's promotion. 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 is 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. This happens because memory discards to soon the information of the environments that reappear later. In the presence of the noise the solutions found were the worst and, just like before, as we increase the value of ρ the time needed for the EA to react after a change, also increases. The same conclusions can be drawn for random environments.

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 in general achieved the best results.

Currently we are introducing some modifications in VMEA in order to improve its performance in noisy and random environments. We are also testing a different type of aging and an alternative process of controlling the sizes of memory and population. Other memory-based algorithms, proposed by other authors are being implemented to use in the future as peer algorithms.

References

1. Branke, J.: Memory Enhanced Evolutionary Algorithms for Changing Optimization Problems. Proceedings of the Congress of Evolutionary Computation (CEC 1999). IEEE, pp. 1875-1882, 1999.
2. Branke J.: Evolutionary Optimization in Dynamic Environments. Norwell MA: Kluwer, 2001.
3. Branke, J., Kaußler T., Schmidt, C. Schmeck, H.: A Multi-Population Approach to Dynamic Optimization Problems. Adaptive Computing in Design and Manufacture (ACDM 2000), pp. 299-308, Springer, 2000.
4. Cobb, H.: 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. Collingwood, E., Corne, D., Ross, P.: Useful Diversity via Multiploidy. Proceedings of the AISB Workshop on Evolutionary Computation, 1996.
6. De Falco, I., Iazzetta, A., Tarantino, E., Della Cioppa, A.: On Biologically Inspired Mutations: The Translocation. Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference (GECCO' 2000), pp. 70-77, Las Vegas, USA, 8-12 July 2000.
7. Goldberg, D. E., Smith, R. E.: Nonstationary Function Optimization using Genetic Algorithms with Dominance and Diploidy. In J. J. Grefenstette (ed.), Proceedings of the Second International Conference on Genetic Algorithms, pp. 59-68. Laurence Erlbaum Associates, 1987.
8. Grefenstette, J. J: Genetic Algorithms for Changing Environments. Proceedings of Parallel Problem Solving from Nature 2, pp. 137-144, North-Holland, 1992.
9. Hadad, B., Eick, C.: Supporting Poliploidy in Genetic Algorithms using Dominance Vectors. In P. Angeline, R. G. Reynolds, J. R. McDonnell and R. Eberhart (eds.) Proceedings of the Sixth International Conference on Evolutionary Programming, vol. 1213 of LNCS. Springer, 1997.
10. Harvey, I.: The Microbial Genetic Algorithm. Unpublished, 1996.
11. Liles, W., De Jong, K.: The Usefulness of Tag Bits in Changing Environments. Proceedings of the Congress on Evolutionary Computation (CEC 99), pp. 2054-2060, IEEE 1999.
12. Louis, S., Xu, Z.: Genetic Algorithms for Open Shop Scheduling and Re-Scheduling. Proceedings of Conference on Computers and their Applications. 1996.
13. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. 3rd Edition Springer-Verlag, 1999.
14. Morrison R. W., De Jong, K.: Triggered Hypermutation Revisited. Proceedings of the Congress on Evolutionary Computation (CEC99), pp. 1025-1032, IEEE 1999.
15. Ng, K. P., Wong, K. C.: A New Diploid Scheme and Dominance Change Mechanism for Non-stationary Function Optimization. Proceedings of the Sixth International Conference on Genetic Algorithms, pp. 159-166. Morgan Kaufmann, 1995.
16. Ramsey, C. L., Greffenstette, J. J.: Case-based Initialization of Genetic Algorithms. In S. Forrest (ed.), Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 84-91. Morgan Kaufmann, 1993.
17. Russell, P. J.: Genetics. 5th edition, Addison-Wesley, 1998.
18. Ryan, C.: The Degree of Oneness. Proceedings of the ECAI Workshop on Genetic Algorithms. Springer-Verlag 1996.
19. Sima Uyar, A., Emre Harmanci, A.: A New Population Based Adaptive Domination Change Mechanism for Diploid Genetic Algorithms in Dynamic Environments. Soft Computing, 9: 803-814, 2005.
20. Simões, A., Costa, E.: Transposition: A Biologically Inspired Mechanism to Use with Genetic Algorithms. Proceedings of the Fourth International Conference on Neural Networks and Genetic Algorithms (ICANNGA'99), pp. 612- 19. Springer-Verlag 1999.
21. Simões, A., Costa, E.: On Biologically Inspired Genetic Operators: Transformation in the Standard Genetic Algorithm. Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001. pp. 584-591, San Francisco, USA, 7-11 July, CA: Morgan Kaufmann Publishers, 2001.
22. Simões, A., Costa, E.: A Comparative Study Using Genetic Algorithms to Deal with Dynamic Environments. Proceedings of the Sixth International Conference on Neural Networks and Genetic Algorithms (ICANNGA'03), pp. 203-209, Roanne, France, 23-25 April, Springer, 2003.
23. Simões, A., Costa, E.: An Immune System-Based Genetic Algorithm to Deal with Dynamic Environments: Diversity and Memory. Proceedings of the Sixth International Conference on Neural Networks and Genetic Algorithms (ICANNGA'03), pp. 168-174, Roanne, France, 23-25 April, Springer, 2003.
24. Smith, P.: Conjugation: A Bacterially Inspired Form of Genetic. Late Breaking Papers at the Genetic Programming 1996 Conference, Stanford University July 28-31, 1996.

25.Smith, P.: Finding Hard Satisfiability Problems using Bacterial Conjugation. AISB Workshop on Evolutionary Computing, pp. 236 – 244, University of Sussex, April 2006.
26.Trojanowski, S., Michalewicz, Z.: Searching for Optima in Non-Stationary Environments. Proceedings of the Congress on Evolutionary Computation (CEC99), pp. 1843-1850, IEEE 1999.
27.Wineberg, M., Oppacher, F.: Enhancing GA's Ability to Cope with Dynamic Environments Proceedings of the Genetic and Evolutionary Computation Conference (GECCO' 2000), pp. 3-10, Las Vegas, USA, 8-12 July, San Francisco, CA: Morgan Kaufmann, 2000.
28.Yang, S.: A Comparative Study of Immune System Based Genetic Algorithms in Dynamic Environments. Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1377-1384, Washington, 2006.
29.Yang, S.: Associative Memory Scheme for Genetic Algorithms in Dynamic Environments. Proceedings of the EvoWorkshops 2006, LNCS 3097, pp. 788-799, Springer-Verlag, 2006.
30.Yang, S.: Experimental Study on Population-Based Incremental Learning Algorithms for Dynamic Optimization problems. Soft Computing, 9 (11), pp. 815-834, 2005.
31.Yang, S.: Memory-Based Immigrants for Genetic Algorithms in Dynamic Environments. Proceedings of the Genetic and Evolutionary Computation Conference, 2005.
32.Yang, S.: Non-Stationary Problem Optimization Using Primal-Dual Genetic Algorithm. Proceedings of the Congress on Evolutionary Computation (CEC 2003), vol. 3, pp. 2246-2253, IEEE, 2003