Improving Memory-based Evolutionary Algorithms in Changing Environments

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 2007/004 - ISSN 0874-338X **March 2007**

Abstract. When using Evolutionary Algorithms (EAs) in no stationary problems some extensions have been introduced in order to avoid the convergence of the population towards a point of the search space. One of these improvements consists in the use of explicit memory responsible for storing good individuals from the search population. When the environment is cyclic and previous environments reappear later, memory should allow continuous progression of the EA's performance with the least decline of the individuals' fitness. But in most situations this purpose is not achieved, and the typical behavior of an EA when a change happens is the best-fitness decrease and some time is necessary to readapting to the new conditions. The key problem when using explicit memory is the size's restrictions usually imposed. So, when it is necessary to store a new individual and memory is full, it is necessary to replace individuals. This replacement can lead to the destruction of information that might be useful in the future. In this work 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 are tested in several memory-based EAs and the obtained results show that memory can be used in a more effective way such that the algorithms' performance is strongly improved.

Keywords: Evolutionary Algorithms, Dynamic Environments, Memory, Diversity, Replacing Strategies

1 Introduction

Evolutionary Algorithms (EAs) have been widely used in the search of good solutions in changing environments. One of the most used mechanisms in EA is the incorporation of explicit memory which has the role to store good individuals during the evolutionary process, which can be useful in future conditions [1], [5], [7], [9], [13] and [14]. The common 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 situations.

It is our conviction that 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 a decline of the best individual's fitness followed by the algorithm's readaptation to the new conditions of the environment [7], [14].

Earlier work investigated an effective EA concerned in the improvement of the populations' resources, namely the information stored in memory. This algorithm called Variable-size Memory Evolutionary Algorithm (VMEA) uses a search and memory populations with variable sizes and was used with considerable success in changing environments [9].

In this work we are interested in studying different schemes to improve the memory's efficiency in an EA. We propose two different and efficient mechanisms of storing individuals into the memory (replacing strategies) and will test them in VMEA and in 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 report 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. In section 7 we state the main conclusions and ideas for future work. Finally, in section 10, tables and graphics concerning the algorithms efficiency can be consulted.

2. 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 ideal of 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 current population is stored in the memory. If there is no room to keep this new solution, then the best individual of the current population is introduced replacing a memory individual chosen accordingly to the replacing scheme. In that paper, an aging-based replacing strategy was used. This 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, besides the environmental changes occur very r generations, the algorithm has no information about when next change will arise 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. Fig. 1 details the algorithm.

3. Replacing Strategies

The idea of adding 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 is well used, when an environment reappears, a good memory individual can avoid an abrupt decrease on the algorithm's performance. Memory can also be used to maintain diversity in the population, allowing 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 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 work we will try to 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 current in memory to be replaced, as long as the new one is better. This replacing strategy was already used in several works ([1], [13], [14]).

Simões and Costa [9] proposed a replacing strategy based in 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 the population when a change occurs, its age is increased by a certain value and finally if a limit age is reached, their age is reset to zero. When it's necessary to update memory, the *youngest* one is selected to be replaced. The basic idea is to swap one individual with a poor contribution in the evolutionary process: one that was never selected to the population or that is in the memory for a long time. This approach was called by *age1*.

In this work **we propose two different replacing strategie**s. The first, called by ***age2***, is also based on the age of memory individuals. The age of an individual ***i*** is given by:

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

As suggested by [1], the individuals' age is computed every generation as a linear combination of their actual age and a contribution of its fitness (*fit_rate*). Besides, 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, denominated by ***generational***, selects the worst individual present in the memory ***since the last change***. For instance, if 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 its time to update 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 will be called by ***gen***.

```
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 individual of memory according the replacing scheme
       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
```

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

4. 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, using the genetic information of previous ones, allowing the algorithm to 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 [11] 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] 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.

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', the remaining become 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). 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 individuals of the mating pool.

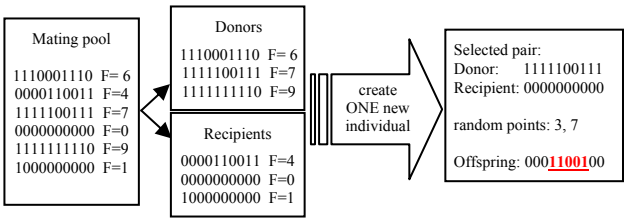


Fig. 2 - Creating a new individual using conjugation

The results obtained using this operator in an EA to solve dynamic optimization problems were quite promising [9].

5. Experimental Design

5.1 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⊕M where ⊕ 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 ⊕ M, t). Using the DOP generator the characteristics of the change are controlled by two parameters: the speed of the change, ***r***, which is the number of generations between two changes, and the magnitude of the change, ρ, that 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 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 [15].

In this work we constructed 16 cyclic DOPs, setting the parameter **r** to 10, 50, 100 and 200. The ratio ρ was set to different values in order to test different levels of change: 0.1 (a light shifting) 0.2, 0.5, 1.0 (severe change). 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).

5.1.1. Knapsack Problem

The benchmark used was the 0/1 dynamic knapsack. 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:

$$\max v(x) = \sum_{i=1}^{m} v_i x_i \qquad \textbf{(2)}$$

subject to the weight constraint:

$$\sum_{i=1}^{m} w_i x_i < C \qquad \textbf{(3)}$$

We used a knapsack problem with 100 items using strongly correlated sets of randomly generated data constructed in the following way ([4]):

w_i = uniformly random integer [1, 50] **(4)**

$v_i = w_i$ + uniformly random integer [1, 5] **(5)**

$$C = 0.6 \times \sum_{i=1}^{100} w_i \qquad \textbf{(6)}$$

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:

$$f(x) = \begin{cases} \sum_{i=1}^{100} v_i w_i, if \sum_{i=1}^{100} v_i w_i \le C \\ 10^{-10} \times \left[\sum_{i=1}^{100} w_i - \sum_{i=1}^{100} w_i x_i \right], otherwise \end{cases} \qquad \textbf{(7)}$$

5.1.2. 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.

5.2. Experimental Setup

5.2.1. Algorithms' parameters

Previous work compared VMEA using the ***age1*** replacing strategy with other algorithms: the random immigrants' algorithm [2] and the memory-enhanced GA (MEGA) studied in [12]. 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. 3. To see more results consult [9] and [10].

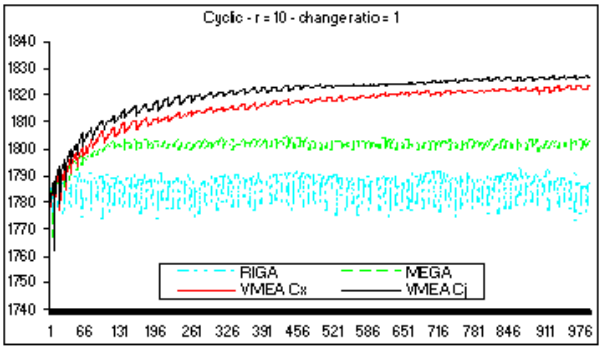


Fig. 3. VMEA *vs* MEGA and RIGA. Typical curve of the algorithms in the dynamic Knapsack problem

In this work we will be focused in the performance of memory-based EAs using the proposed replacing schemes. 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 [13], [12]. 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 through the time, but the sum of the two populations can not 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 population of 110 individuals and 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:

$$Foverall = \frac{1}{G}\sum_{i=1}^{G}\left[\frac{1}{N}\sum_{j=1}^{N}F_{best_{ij}}\right] \qquad \textbf{(8)}$$

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

6. Experimental Results

The statistical results comparing the algorithms are reported section 10. We used paired one-tailed t-test at a 0.01 level of significance. The notation used in the tables with t-test results, 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. In the next sections we will describe the obtained 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. All graphics and tables can be consulted in section 10.

6.1. Comparing Replacing Strategies

6.1.1. Analyzing VMEAs' results

The results obtained by VMEAs in the studied benchmarks are reported in section 10.1. Analyzing the results on both problems, we can see that for the VMEAs, the proposed *gen* replacing scheme generally 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, in this case, we replace the most *similar*. For smaller change periods, the *age2* replacing strategy was globally the best option. 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 combined with conjugation. These results confirm the difficulty of finding a tradeoff between the fitness and age contributions, as suggested in [1].

The proposed conjugation operator proved to be helpful in VMEAs when the changes are more severe and happen in environments with smaller change periods.

6.1.2. Analyzing MEGAs' results

The results obtained by MEGAs in the studied benchmarks are reported in section 10.2. In MEGAs the ***gen*** method substantially improves the performance in environments of small periods and with severe changes. Observing Fig. 10 to 13 we confirm that using the proposed *gen scheme* in MEGA considerably improved its performance. The information stored in memory allows a continuously adaptation of the analyzed EAs, contrary to the remaining replacing strategies. When the changes are lighter and more different states can reappear, the limitation imposed to the memory size, decreases the algorithms' performance.

The ***age2*** method was not as effective as the ***gen*** scheme. In fact, as reported in table 5, this method only improves MEGAs' performance in few situations.

For both studied problems, the conjugation operator has no influence in the obtained results.

6.1.3. Analyzing MIGAs' results

The results obtained by MIGAs in the studied benchmarks are reported in section 10.3. In MIGAs, the *age2* replacing scheme improves the performance in environments with larger change periods (100, 200), especially when the change ratio is lower (0.1). Table 9 shows the results that support this observation. In these cases, the memory size doesn't allow to keep storing useful information from past environments.

Th proposed ***gen*** method once again substantially improved the MIGAs' performance. This was more evident in the Onemax problem and in the Knapsack problem dealing with higher change ratios.

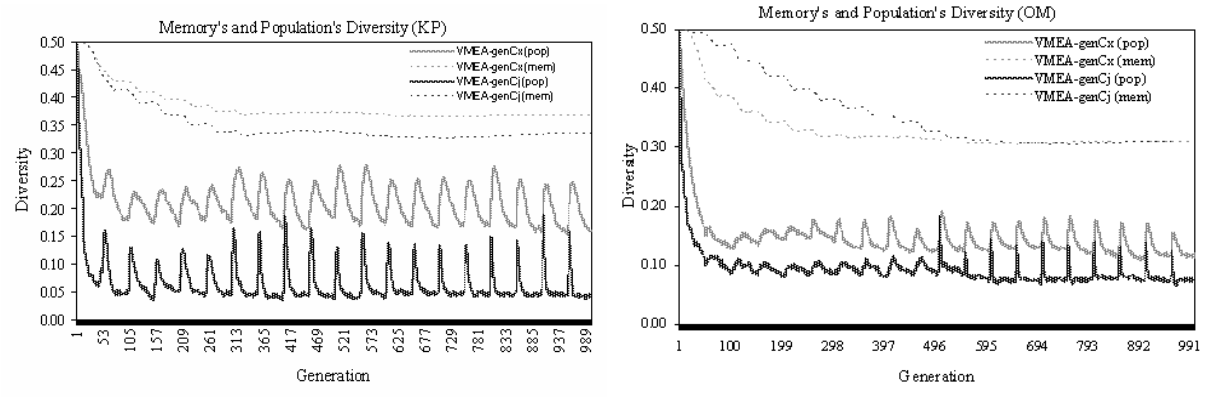
For both studied problems, the conjugation operator has no influence in the obtained results.

6.2. Population's and Memory's Diversity

In order to study the impact of the genetic operators in the diversity dynamics, we stored the population's and memory's diversity at every generation using the standard measure based on the Hamming distance. At generation t, the diversity is given by:

$$Div(t) = \frac{1}{N}\sum_{k=1}^{N}\left[\frac{1}{l.n(l-1)}\sum_{i=1}^{n}\sum_{j\neq i}^{n} HDij(k,t)\right] \qquad \textbf{(9)}$$

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, frequently, the best results are achieved using 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. 4, 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.



(a) (b)

Fig. 4. Population's and Memory's diversity for the first 20 environmental changes: (a) Knapsack; (b) Onemax

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

6.3. 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. 5 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.

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

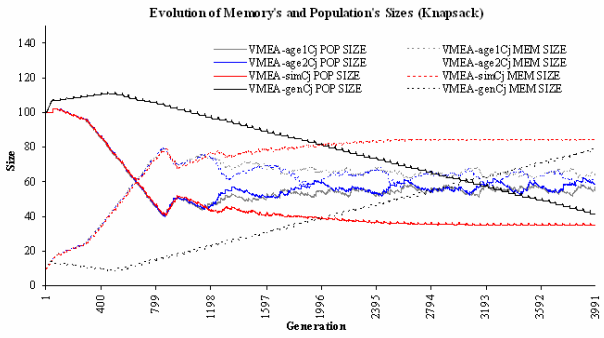


Fig. 5. Population's and Memory's sizes on dynamic Knapsack

7. Conclusions

This report detailed describes two new replacing strategies and tested its efficiency in the 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 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. This result was observed in almost versions of VMEA. In MIGA and MEGAs this is also true in environments with higher change ratios. When the ratios are lower and more different states are possible, the limited size of the memory of these two algorithms is a serious weakness. 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, the proposed approaches of replacing the individuals proved to be excellent choices. Also, the size of the populations (memory and search) is also a critical issue to consider. Using fixed sizes is not a good option.

To fundament our conclusions we are currently testing the proposed ideas in other algorithms, using different problem benchmarks and in environments with noise. We are also detailed studying the influence of the population's and memory's size in the VMEA's efficiency.

8. Acknowledgment

The work described in this report was partially financed by the PhD Grant BD/39293/2006 of the Foundation for Science and Technology of the Portuguese Ministry of Science and Technology and High Education.

9. 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] P. Smith. Conjugation: A Bacterially Inspired Form of Genetic. Late Breaking Papers at the Genetic Programming 1996 Conf., Stanford Univ., July 28-31, 1996.
[12] S. Yang. Experimental Study on Population-Based Incremental Learning Algorithms for Dynamic Optimization problems. Soft Computing, vol. 9, nº 11, pp. 815-834, 2005.
[13] 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.
[14] 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.
[15] S. Yang. Associative Memory Scheme for Genetic Algorithms in Dynamic Environments. Applications of Evolutionary Computing, LNCS 3097, pp. 788-799, Springer Verlag Berlin, 2006.

10. Tables and graphics

10.1. VMEA RESULTS

Table 1. T-test results comparing VMEAs using the *age2* strategy with *age1*, *similar* and *gen*

		KP				OneMax			
T-test results	r $\rho\rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
VMEA-age2Cx -- VMEA-age1Cx		+	+	+	+	-	+	+	-
VMEA-age2Cx -- VMEA-age1Cj		-	-	--	--	++	+	--	--
VMEA-age2Cx -- VMEA-simCx		+	++	+	+	+	+	++	+
VMEA-age2Cx -- VMEA-simCj		-	+	-	--	++	+	--	--
VMEA-age2Cx -- VMEA-genCx		+	+	+	+	+	+	++	++
VMEA-age2Cx -- VMEA-genCj		-	+	-	--	++	-	--	--
VMEA-age2Cj -- VMEA-age1Cx	10	++	++	++	++	--	+	++	++
VMEA-age2Cj -- VMEA-age1Cj		+	+	+	+	-	+	+	+
VMEA-age2Cj -- VMEA-simCx		++	++	++	++	--	+	++	++
VMEA-age2Cj -- VMEA-simCj		++	++	+	+	+	+	+	++
VMEA-age2Cj -- VMEA-genCx		++	++	++	++	--	+	++	++
VMEA-age2Cj -- VMEA-genCj		++	++	+	+	+	+	+	++
VMEA-age2Cj -- VMEA-age2Cx		++	+	+	++	--	++	++	++
VMEA-age2Cx -- VMEA-age1Cx		-	--	+	+	+	+	+	+
VMEA-age2Cx -- VMEA-age1Cj		+	-	-	-	++	+	-	--
VMEA-age2Cx -- VMEA-simCx		--	--	+	+	--	--	++	++
VMEA-age2Cx -- VMEA-simCj		--	--	-	+	+	--	--	-
VMEA-age2Cx -- VMEA-genCx		--	--	-	+	--	--	++	+
VMEA-age2Cx -- VMEA-genCj		--	--	--	-	--	--	--	--
VMEA-age2Cj -- VMEA-age1Cx	50	-	-	++	++	--	-	++	++
VMEA-age2Cj -- VMEA-age1Cj		+	-	+	-	++	-	+	+
VMEA-age2Cj -- VMEA-simCx		--	--	++	++	--	--	++	++
VMEA-age2Cj -- VMEA-simCj		--	--	+	++	-	--	++	++
VMEA-age2Cj -- VMEA-genCx		--	--	+	++	--	--	++	++
VMEA-age2Cj -- VMEA-genCj		--	--	--	+	--	--	-	++
VMEA-age2Cj -- VMEA-age2Cx		-	+	++	+	--	-	++	++
VMEA-age2Cx -- VMEA-age1Cx		+	-	-	--	+	-	++	-
VMEA-age2Cx -- VMEA-age1Cj		-	++	-	--	++	++	++	-
VMEA-age2Cx -- VMEA-simCx		--	--	--	--	--	--	-	+
VMEA-age2Cx -- VMEA-simCj		--	--	--	--	-	-	-	-
VMEA-age2Cx -- VMEA-genCx		--	--	--	--	--	--	-	+
VMEA-age2Cx -- VMEA-genCj		--	--	--	--	--	--	--	-
VMEA-age2Cj -- VMEA-age1Cx	100	-	--	+	--	+	-	+	++
VMEA-age2Cj -- VMEA-age1Cj		-	+	+	--	++	++	+	-
VMEA-age2Cj -- VMEA-simCx		-	--	--	-	--	--	-	+
VMEA-age2Cj -- VMEA-simCj		-	--	--	--	-	-	--	+
VMEA-age2Cj -- VMEA-genCx		--	--	--	-	--	--	--	++
VMEA-age2Cj -- VMEA-genCj		--	--	--	--	--	--	--	+
VMEA-age2Cj -- VMEA-age2Cx		-	--	+	-	+	+	-	+
VMEA-age2Cx -- VMEA-age1Cx		-	+	-	--	+	-	++	++
VMEA-age2Cx -- VMEA-age1Cj		+	+	--	--	--	++	+	++
VMEA-age2Cx -- VMEA-simCx		-	--	--	--	--	--	--	++
VMEA-age2Cx -- VMEA-simCj		-	--	--	--	--	--	--	++
VMEA-age2Cx -- VMEA-genCx		--	--	--	--	--	--	--	++
VMEA-age2Cx -- VMEA-genCj		--	--	--	--	--	--	--	-
VMEA-age2Cj -- VMEA-age1Cx	200	+	-	+	--	++	-	++	++
VMEA-age2Cj -- VMEA-age1Cj		+	+	+	--	-	+	+	++
VMEA-age2Cj -- VMEA-simCx		-	--	--	--	+	--	--	++
VMEA-age2Cj -- VMEA-simCj		-	--	--	--	--	--	--	+
VMEA-age2Cj -- VMEA-genCx		--	--	--	--	--	--	--	+
VMEA-age2Cj -- VMEA-genCj		--	--	--	--	--	--	--	-
VMEA-age2Cj -- VMEA-age2Cx		+	-	++	-	++	-	-	+

Table 2. T-test results comparing VMEAs using the *gen* strategy with *age2*, *age1* and *similar*

		KP				OneMax			
T-test results	*r* $\rho\rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
VMEA-genCx -- VMEA-age1Cx		-	-	-	-	--	+	--	--
VMEA-genC x-- VMEA-age1Cj		-	-	--	--	++	-	--	--
VMEA-genCx -- VMEA-age2Cx		-	-	-	-	-	-	--	--
VMEA-genCx -- VMEA-age2Cj		--	--	--	--	++	-	--	--
VMEA-genCx -- VMEA-simCx		-	-	-	-	-	-	-	-
VMEA-genCx -- VMEA-simCj		-	--	--	--	++	+	--	--
VMEA-genCj -- VMEA-age1Cx	10	+	-	+	+	--	+	++	++
VMEA-genCj -- VMEA-age1Cj		+	-	--	-	-	+	-	--
VMEA-genCj -- VMEA-age2Cx		+	-	+	++	--	+	++	++
VMEA-genCj -- VMEA-age2Cj		--	--	-	-	-	-	-	--
VMEA-genCj -- VMEA-simCx		+	+	+	+	--	+	++	++
VMEA-genCj -- VMEA-simCj		+	-	-	-	-	+	-	-
VMEA-genCj - VMEA-gen Cx		+	+	++	++	--	+	++	++
VMEA-genCx -- VMEA-age1Cx		++	++	+	-	++	++	--	-
VMEA-genC x-- VMEA-age1Cj		++	++	+	-	++	++	--	--
VMEA-genCx -- VMEA-age2Cx		++	++	+	-	++	++	--	-
VMEA-genCx -- VMEA-age2Cj		++	++	-	--	++	++	--	--
VMEA-genCx -- VMEA-simCx		++	+	+	-	++	+	++	+
VMEA-genCx -- VMEA-simCj		++	+	-	-	++	+	--	-
VMEA-genCj -- VMEA-age1Cx	50	++	++	++	++	++	++	++	++
VMEA-genCj -- VMEA-age1Cj		++	++	++	-	++	++	++	+
VMEA-genCj -- VMEA-age2Cx		++	++	++	+	++	++	++	++
VMEA-genCj -- VMEA-age2Cj		++	++	++	-	++	++	+	--
VMEA-genCj -- VMEA-simCx		++	+	++	+	-	-	++	++
VMEA-genCj -- VMEA-simCj		+	++	++	+	++	+	+	++
VMEA-genCj -- VMEA-gen Cx		+	+	+	++	--	--	++	++
VMEA-genCx -- VMEA-age1Cx		++	++	++	+	++	++	++	-
VMEA-genC x-- VMEA-age1Cj		++	++	++	-	++	++	++	--
VMEA-genCx -- VMEA-age2Cx		++	++	++	++	++	++	+	-
VMEA-genCx -- VMEA-age2Cj		++	++	++	+	++	++	++	--
VMEA-genCx -- VMEA-simCx		++	++	+	+	++	++	+	-
VMEA-genCx -- VMEA-simCj		++	++	+	+	++	++	--	-
VMEA-genCj -- VMEA-age1Cx	100	++	++	++	+	++	++	++	+
VMEA-genCj -- VMEA-age1Cj		++	++	++	+	++	++	++	-
VMEA-genCj -- VMEA-age2Cx		++	++	++	++	++	++	++	+
VMEA-genCj -- VMEA-age2Cj		++	++	++	++	++	++	++	-
VMEA-genCj -- VMEA-simCx		++	++	+	++	++	++	++	+
VMEA-genCj -- VMEA-simCj		++	++	+	+	++	++	++	+
VMEA-genCj -- VMEA-gen Cx		+	+	+	+	--	--	++	++
VMEA-genCx -- VMEA-age1Cx		++	++	++	+	++	++	++	++
VMEA-genC x-- VMEA-age1Cj		++	++	++	-	++	++	++	++
VMEA-genCx -- VMEA-age2Cx		++	++	++	++	++	++	++	--
VMEA-genCx -- VMEA-age2Cj		++	++	++	++	++	++	++	-
VMEA-genCx -- VMEA-simCx		++	++	+	+	++	++	+	+
VMEA-genCx -- VMEA-simCj		++	++	+	+	++	++	--	+
VMEA-genCj -- VMEA-age1Cx	200	++	++	++	+	++	++	++	++
VMEA-genCj -- VMEA-age1Cj		++	++	++	-	++	++	++	++
VMEA-genCj -- VMEA-age2Cx		++	++	++	++	++	++	++	+
VMEA-genCj -- VMEA-age2Cj		++	++	++	++	++	++	++	+
VMEA-genCj -- VMEA-simCx		++	++	+	+	++	++	++	++
VMEA-genCj -- VMEA-simCj		++	++	+	+	++	++	++	++
VMEA-genCj -- VMEA-gen Cx		++	+	+	+	++	++	++	++

Table 3. Global results obtained by VMEAs in the Dynamic Knapsack Problem

Change Ratio	Change Speed	VMEA Cx AGE 1	VMEA Cj AGE 1	VMEA Cx AGE2	VMEA Cj AGE2	VMEA Cx SIMILAR	VMEA Cj SIMILAR	VMEA Cx GEN	VMEA Cj GEN
0.1	10	1794.7	1796.4	1794.3	**1798.3**	1792.6	1794.6	1792.2	1795.7
	50	1804.8	1801.5	1802.3	1802.3	1808.8	1808.6	1812.8	**1814.0**
	100	1812.2	1811.9	1812.2	1811.9	1814.4	1814.0	1817.5	**1819.3**
	200	1820.0	1819.6	1819.8	1820.6	1821.0	1821.2	1824.0	**1826.0**
0.2	10	1792.7	1794.5	1793.7	**1797.2**	1790.5	1793.6	1790.9	1791.9
	50	1807.1	1803.8	1802.1	1803.4	1812.6	1812.9	1814.4	**1815.3**
	100	1803.6	1801.0	1803.2	1801.5	1814.6	1812.8	1818.7	**1820.1**
	200	1812.5	1812.1	1812.7	1811.6	1819.8	1819.1	1825.4	**1827.2**
0.5	10	1792.7	**1801.3**	1793.7	1801.2	1791.7	1799.6	1790.5	1796.9
	50	1815.6	1818.6	1816.5	1819.6	1817.1	1819.7	1819.0	**1822.2**
	100	1811.7	1812.1	1811.1	1813.1	1821.9	1821.6	1824.0	**1825.1**
	200	1805.0	1805.9	1803.6	1806.4	1827.7	1827.3	1829.7	**1830.7**
1	10	1794.0	1808.0	1794.6	**1810.2**	1793.5	1805.5	1790.5	1802.1
	50	1825.9	**1829.1**	1826.4	1828.9	1824.7	1826.6	1825.2	1828.3
	100	1828.5	1828.8	1823.1	1822.4	1827.9	1828.4	1828.3	**1830.4**
	200	1832.8	1833.0	1823.6	1822.9	1831.5	1831.1	**1833.1**	**1833.1**

Table 4. Global results obtained by VMEAs in the Dynamic Onemax Problem

Change Ratio	Change Speed	VMEA Cx AGE 1	VMEA Cj AGE 1	VMEA Cx AGE2	VMEA Cj AGE2	VMEA Cx similar	VMEA Cj similar	VMEA Cx GEN	VMEA Cj GEN
0.1	10	**249.5**	241.5	248.8	240.4	247.5	240.6	247.7	240.0
	50	264.4	259.4	264.9	261.8	269.4	263.6	**274.0**	269.2
	100	270.2	269.1	270.4	270.3	272.9	270.6	**277.8**	276.1
	200	277.5	279.7	277.3	279.5	279.0	280.6	283.6	**286.3**
0.2	10	236.0	238.2	236.7	**242.9**	237.2	238.4	236.5	240.3
	50	266.2	266.7	267.1	263.8	274.5	273.7	**276.1**	274.6
	100	264.1	260.8	264.6	264.3	274.6	267.6	**281.2**	279.8
	200	273.9	271.7	273.7	272.9	279.9	277.7	287.7	**289.6**
0.5	10	220.7	245.7	222.2	**247.0**	214.7	243.0	214.4	243.2
	50	281.7	283.8	282.1	285.3	277.4	284.2	279.4	**285.7**
	100	278.5	279.4	283.5	281.5	284.8	286.3	285.0	**287.6**
	200	264.7	267.4	271.0	271.2	291.8	293.5	292.7	**294.1**
1	10	232.2	263.7	230.9	**264.0**	220.2	254.7	216.5	250.9
	50	288.7	292.0	288.2	**293.6**	283.2	289.5	286.0	292.9
	100	292.9	**294.8**	292.9	294.4	291.4	293.1	291.1	293.7
	200	293.2	291.3	296.7	296.8	294.3	295.9	296.0	**297.0**

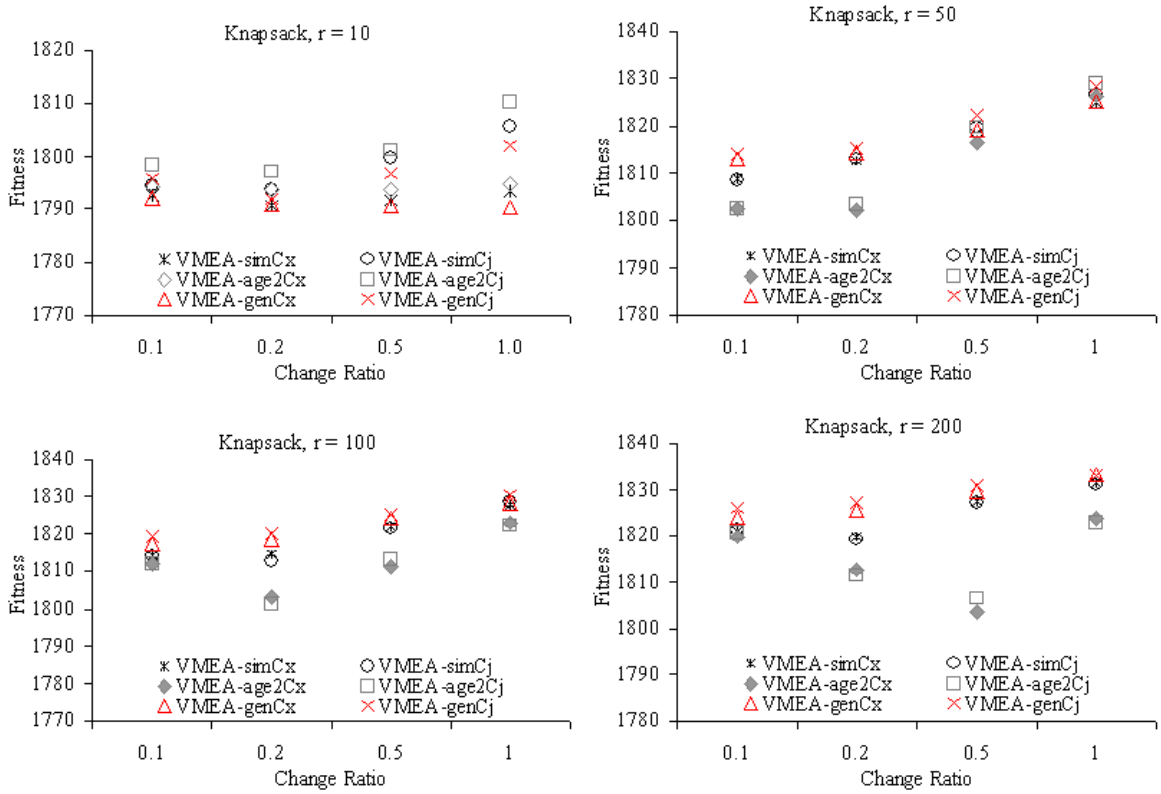


Fig. 6. Global results obtained by VMEAs in the Dynamic Knapsack Problem

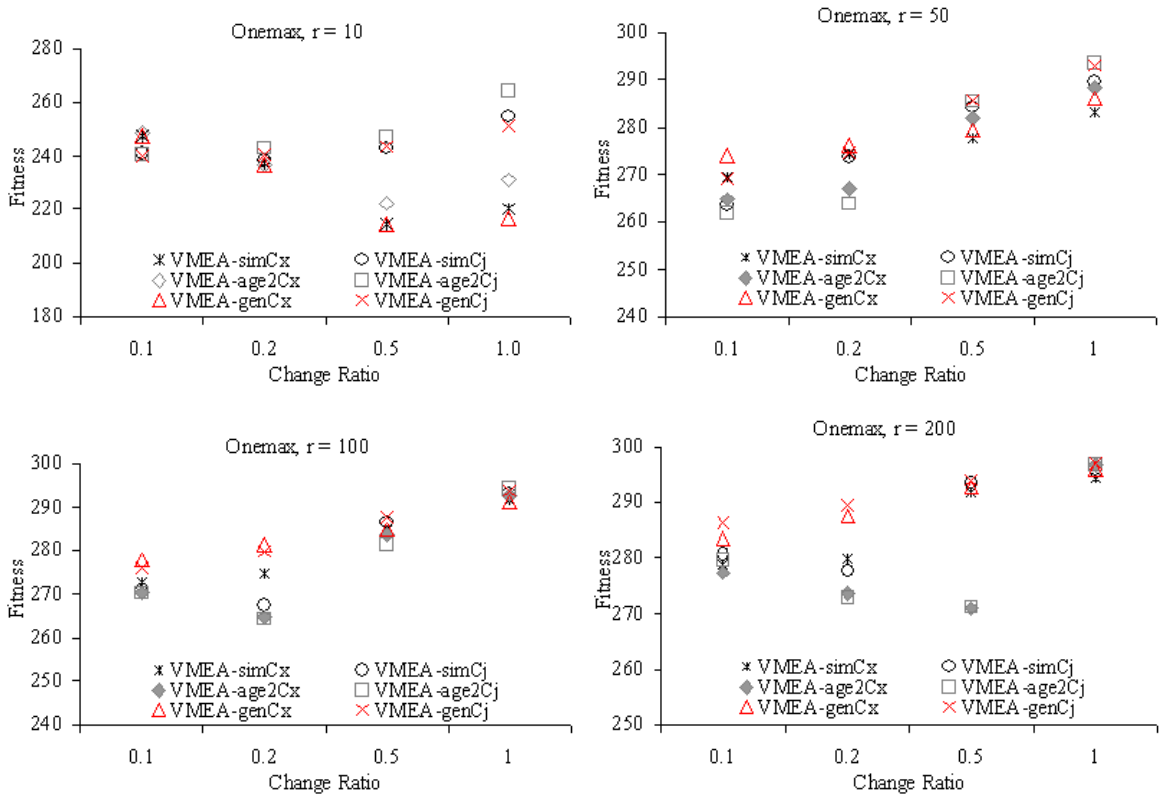


Fig. 7. Global results obtained by VMEAs in the Dynamic Onemax Problem

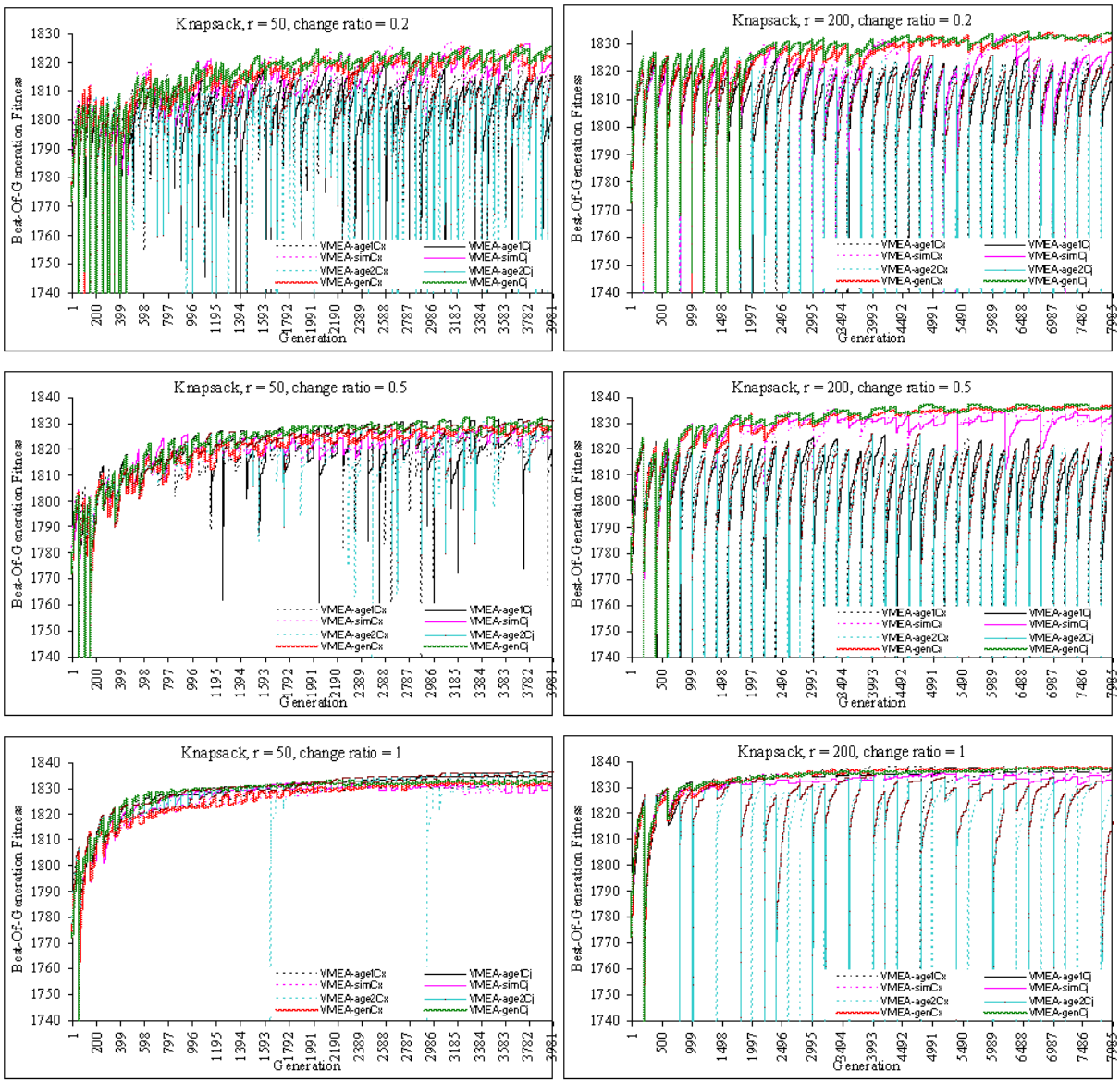


Fig. 8. Comparing VMEAs on dynamic Knapsack, with r = 50 and r = 200, when ρ = 0.2, ρ = 0.5, ρ = 1.0

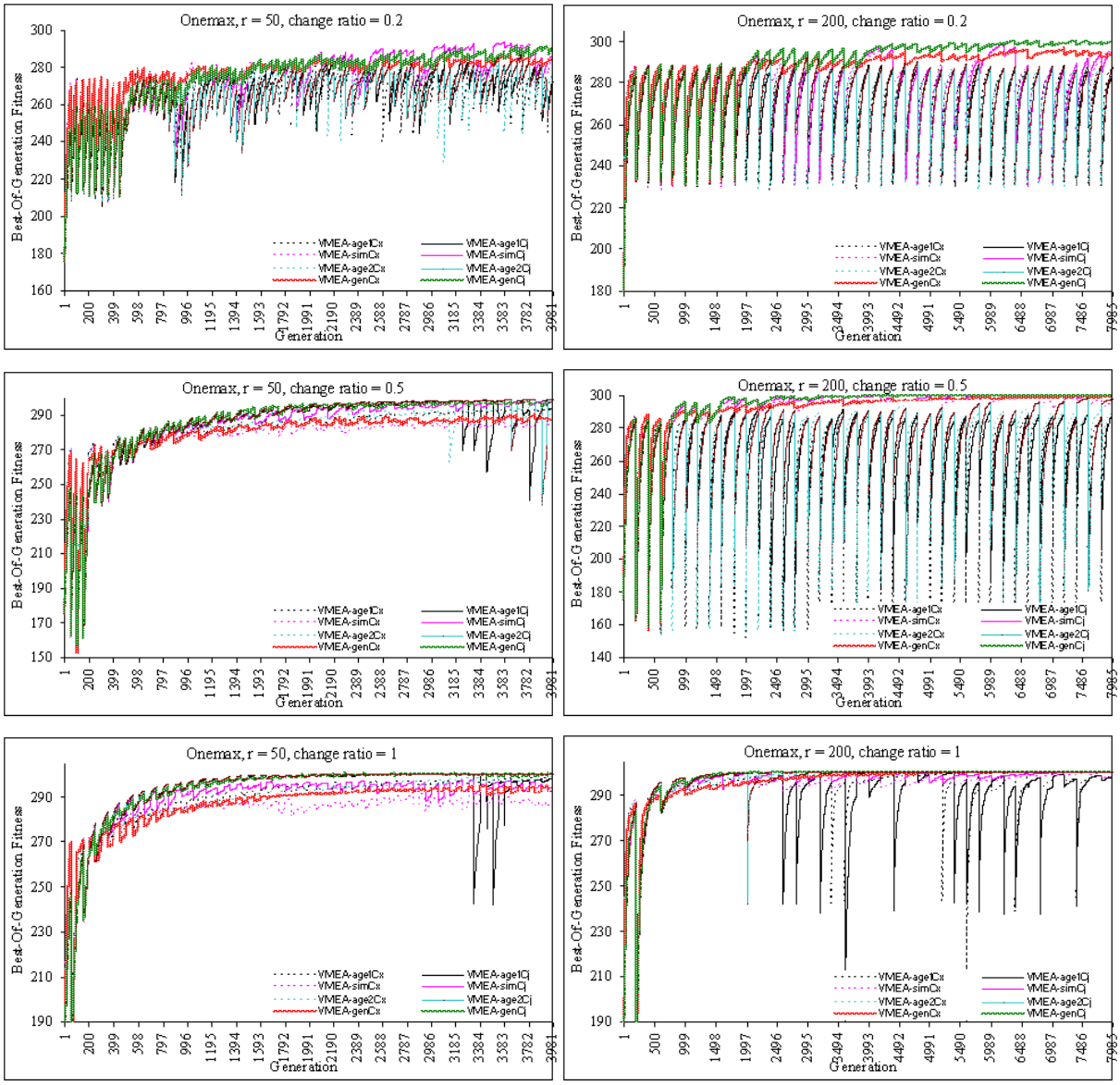


Fig. 9. Comparing VMEAs on dynamic Onemax, with r = 50 and r = 200, when ρ = 0.2, ρ = 0.5, ρ = 1.0

10.2. MEGA RESULTS

Table 5. T-test results comparing MEGAs using the *age2* strategy with *similar* and *gen*

		KP				OneMax			
T-test results	r $\rho\rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
MEGA-age2Cx -- MEGA-simCx	10	-	--	--	-	--	--	--	-
MEGA-age2Cx -- MEGA-simCj		-	--	--	-	--	--	--	-
MEGA-age2Cx -- MEGA-genCx		-	--	--	-	--	--	--	--
MEGA-age2Cx -- MEGA-genCj		-	--	--	-	--	--	--	--
MEGA-age2Cj -- MEGA-simCx		-	--	--	-	--	--	-	-
MEGA-age2Cj -- MEGA-simCj		-	--	--	-	--	--	-	+
MEGA-age2Cj -- MEGA-genCx		+	--	--	--	--	--	--	--
MEGA-age2Cj -- MEGA-genCj		-	--	--	--	--	--	--	--
MEGA-age2Cj -- MEGA-age2Cx		+	-	+	-	+	-	+	+
MEGA-age2Cx -- MEGA-simCx	50	-	+	-	--	-	--	--	--
MEGA-age2Cx -- MEGA-simCj		+	-	-	--	-	--	--	--
MEGA-age2Cx -- MEGA-genCx		+	-	--	--	--	--	--	--
MEGA-age2Cx -- MEGA-genCj		-	-	--	--	--	--	--	--
MEGA-age2Cj -- MEGA-simCx		+	+	-	-	-	--	--	--
MEGA-age2Cj -- MEGA-simCj		+	-	-	-	-	-	--	--
MEGA-age2Cj -- MEGA-genCx		+	-	--	--	--	--	--	--
MEGA-age2Cj -- MEGA-genCj		-	--	--	--	--	--	--	--
MEGA-age2Cj -- MEGA-age2Cx		+	-	+	+	-	+	-	-
MEGA-age2Cx -- MEGA-simCx	100	+	-	-	-	-	+	+	+
MEGA-age2Cx -- MEGA-simCj		+	-	-	-	-	+	+	++
MEGA-age2Cx -- MEGA-genCx		+	-	--	--	-	--	--	--
MEGA-age2Cx -- MEGA-genCj		+	--	--	--	--	--	--	--
MEGA-age2Cj -- MEGA-simCx		+	+	-	-	-	+	+	+
MEGA-age2Cj -- MEGA-simCj		+	-	-	+	-	+	+	+
MEGA-age2Cj -- MEGA-genCx		-	-	--	--	-	--	--	--
MEGA-age2Cj -- MEGA-genCj		-	-	--	--	--	--	--	--
MEGA-age2Cj -- MEGA-age2Cx		-	+	-	+	+	+	-	+
MEGA-age2Cx -- MEGA-simCx	200	-	--	+	+	+	+	+	++
MEGA-age2Cx -- MEGA-simCj		-	-	-	+	+	-	+	++
MEGA-age2Cx -- MEGA-genCx		-	-	--	--	--	--	--	--
MEGA-age2Cx -- MEGA-genCj		-	-	--	--	--	--	--	--
MEGA-age2Cj -- MEGA-simCx		-	-	++	+	+	+	+	++
MEGA-age2Cj -- MEGA-simCj		-	-	+	+	-	-	-	++
MEGA-age2Cj -- MEGA-genCx		-	-	--	--	--	--	--	--
MEGA-age2Cj -- MEGA-genCj		-	-	--	--	--	--	--	--
MEGA-age2Cj -- MEGA-age2Cx		-	+	+	+	-	+	-	+

Table 6. T-test results comparing MEGAs using the *gen* strategy with *similar* and *age2*

		KP				OneMax			
T-test results	*r* $\rho\rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
MEGA-genCx -- MEGA-age2Cx	10	+	++	++	+	++	++	++	++
MEGA-genCx -- MEGA-age2Cj		-	++	++	++	++	++	++	++
MEGA-genCx -- MEGA-simCx		-	-	+	++	++	++	++	++
MEGA-genCx -- MEGA-simCj		-	+	++	++	++	++	++	++
MEGA-genCj -- MEGA-age2Cx		+	++	++	+	++	++	++	++
MEGA-genCj -- MEGA-age2Cj		+	++	++	++	++	++	++	++
MEGA-genCj -- MEGA-simCx		+	-	++	++	++	++	++	++
MEGA-genCj -- MEGA-simCj		-	-	++	+	++	++	++	++
MEGA-genCj -- MEGA-gen Cx		+	-	+	+	+	-	-	+
MEGA-genCx -- MEGA-age2Cx	50	-	+	++	++	++	++	++	++
MEGA-genCx -- MEGA-age2Cj		-	+	++	++	++	++	++	++
MEGA-genCx -- MEGA-simCx		-	+	++	++	++	++	++	++
MEGA-genCx -- MEGA-simCj		-	+	++	++	++	++	++	++
MEGA-genCj -- MEGA-age2Cx		+	+	++	++	++	++	++	++
MEGA-genCj -- MEGA-age2Cj		+	++	++	++	++	++	++	++
MEGA-genCj -- MEGA-simCx		+	++	++	++	++	++	++	++
MEGA-genCj -- MEGA-simCj		+	+	++	++	++	++	++	++
MEGA-genCj -- MEGA-gen Cx		+	-	+	+	+	-	+	+
MEGA-genCx -- MEGA-age2Cx	100	-	+	++	++	+	++	++	++
MEGA-genCx -- MEGA-age2Cj		+	+	++	++	+	++	++	++
MEGA-genCx -- MEGA-simCx		+	++	++	++	++	++	++	++
MEGA-genCx -- MEGA-simCj		+	+	++	++	+	++	++	++
MEGA-genCj -- MEGA-age2Cx		-	++	++	++	++	++	++	++
MEGA-genCj -- MEGA-age2Cj		+	+	++	++	++	++	++	++
MEGA-genCj -- MEGA-simCx		+	+	++	++	++	++	++	++
MEGA-genCj -- MEGA-simCj		+	+	++	++	++	++	++	++
MEGA-genCj -- MEGA-gen Cx		+	-	-	+	+	+	-	+
MEGA-genCx -- MEGA-age2Cx	200	+	+	++	++	++	++	++	++
MEGA-genCx -- MEGA-age2Cj		+	+	++	++	++	++	++	++
MEGA-genCx -- MEGA-simCx		-	+	++	++	++	++	++	++
MEGA-genCx -- MEGA-simCj		+	+	++	++	++	++	++	++
MEGA-genCj -- MEGA-age2Cx		+	+	++	++	++	++	++	++
MEGA-genCj -- MEGA-age2Cj		+	+	++	++	++	++	++	++
MEGA-genCj -- MEGA-simCx		-	+	++	++	++	++	++	++
MEGA-genCj -- MEGA-simCj		+	+	++	++	++	++	++	++
MEGA-genCj -- MEGA-gen Cx		+	-	-	+	+	-	+	+

Table 7. Global results obtained by MEGAs in the Dynamic Knapsack Problem

Change Ratio	Change Speed	MEGA	MEGA-age2Cx	MEGA-age2Cj	MEGA-genCx	MEGA-genCj
0.1	10	1784.0	1783.2	1783.9	1783.3	**1784.1**
	50	1800.6	1800.6	1800.7	1800.2	**1800.8**
	100	1805.4	**1806.5**	1805.8	1805.9	1806.0
	200	**1810.3**	1809.9	1809.7	1809.9	**1810.3**
0.2	10	1782.3	**1776.5**	1776.4	1781.6	1781.1
	50	1795.1	1795.9	1795.7	**1797.2**	1797.0
	100	1801.4	1801.1	1801.6	**1802.9**	**1802.9**
	200	1807.7	1806.9	1807.3	**1808.7**	1807.9
0.5	10	1781.3	1765.6	1766.7	1788.5	**1791.2**
	50	1791.4	1788.5	1789.3	1804.8	**1806.4**
	100	1796.5	1796.2	1795.7	**1810.4**	1810.1
	200	1802.4	1802.6	1804.0	**1816.5**	1815.8
1	10	1784.8	1774.2	1766.0	1793.4	**1793.9**
	50	1796.9	1786.8	1789.1	1808.7	**1810.7**
	100	1795.7	1793.4	1795.6	1815.5	**1815.6**
	200	1800.6	1800.8	1800.9	1820.4	**1820.8**

Table 8. Global results obtained by MEGAs in the Onemax Knapsack Problem

Change Ratio	Change Speed	MEGA	MEGA-age2Cx	MEGA-age2Cj	MEGA-genCx	MEGA-genCj
0.1	10	223.7	219.3	219.5	232.9	**233.9**
	50	264.5	264.3	264.2	266.5	**266.7**
	100	271.7	271.6	271.7	272.7	**272.8**
	200	276.6	276.8	276.6	277.2	**277.7**
0.2	10	206.7	196.3	196.1	**217.9**	217.3
	50	253.3	252.3	252.5	**266.3**	265.7
	100	265.4	265.4	265.5	271.0	**271.2**
	200	273.5	273.5	273.5	**277.6**	277.4
0.5	10	189.8	180.1	182.0	**210.6**	209.0
	50	237.4	229.5	229.2	270.1	**270.6**
	100	251.5	253.1	252.8	**277.3**	276.9
	200	266.2	267.2	266.3	283.3	**283.6**
1	10	188.0	186.3	186.8	217.1	**217.4**
	50	252.0	235.8	233.9	276.0	**276.1**
	100	248.8	255.2	255.7	282.5	**283.1**
	200	262.1	268.8	269.2	288.5	**288.8**

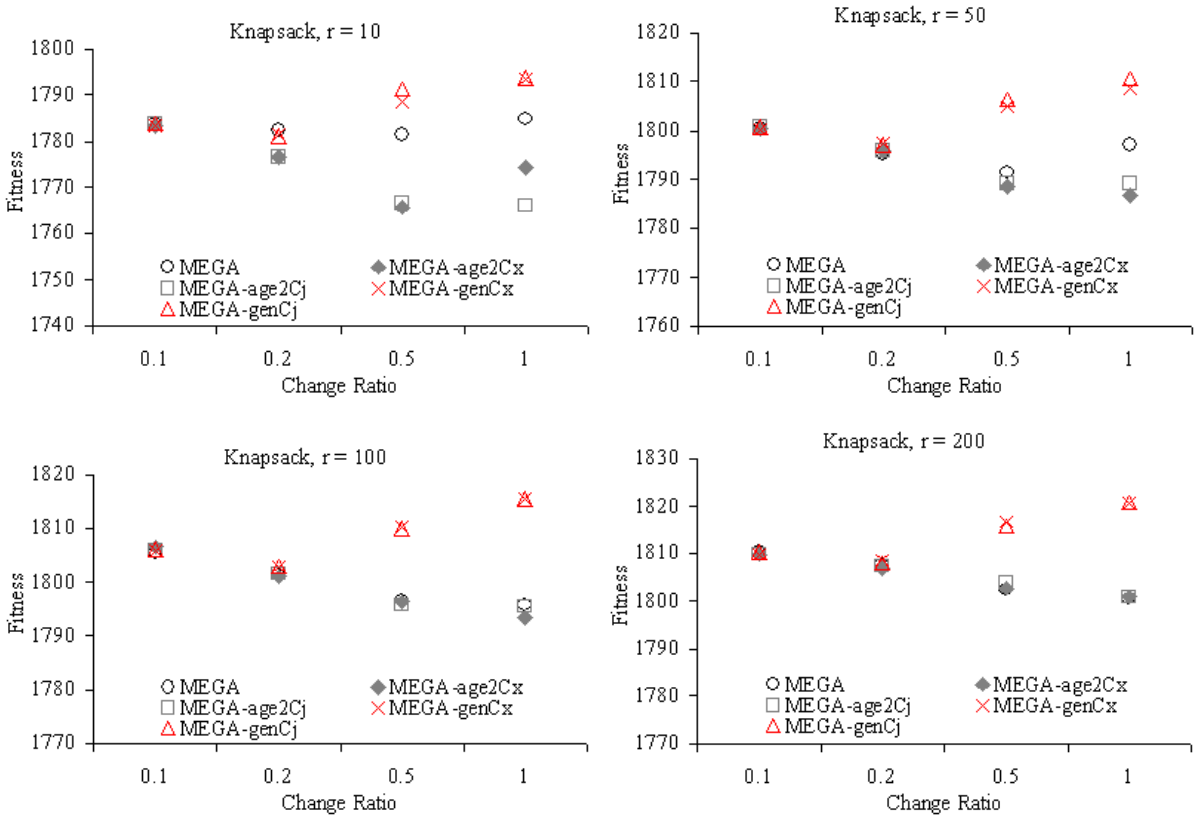


Fig. 10. Global results obtained by MEGAs in the Dynamic Knapsack Problem

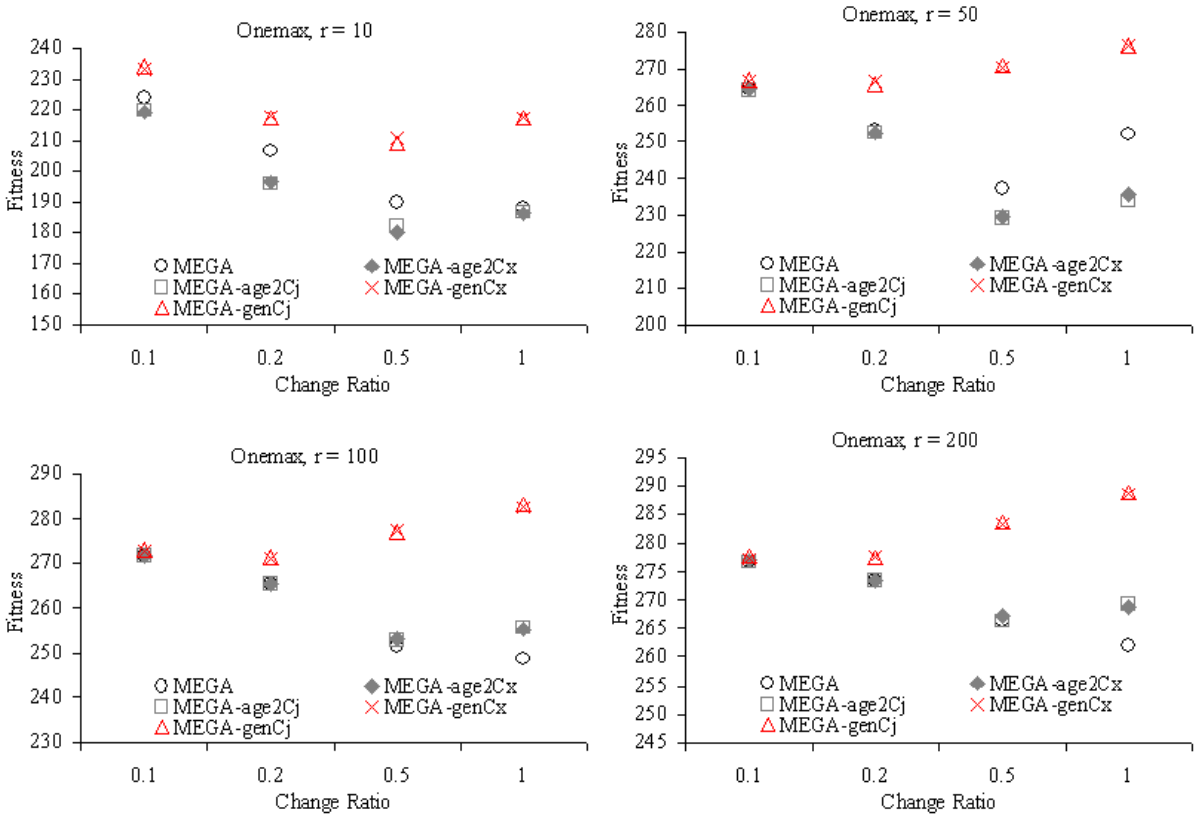


Fig. 11. Global results obtained by MEGAs in the Dynamic Onemax Problem

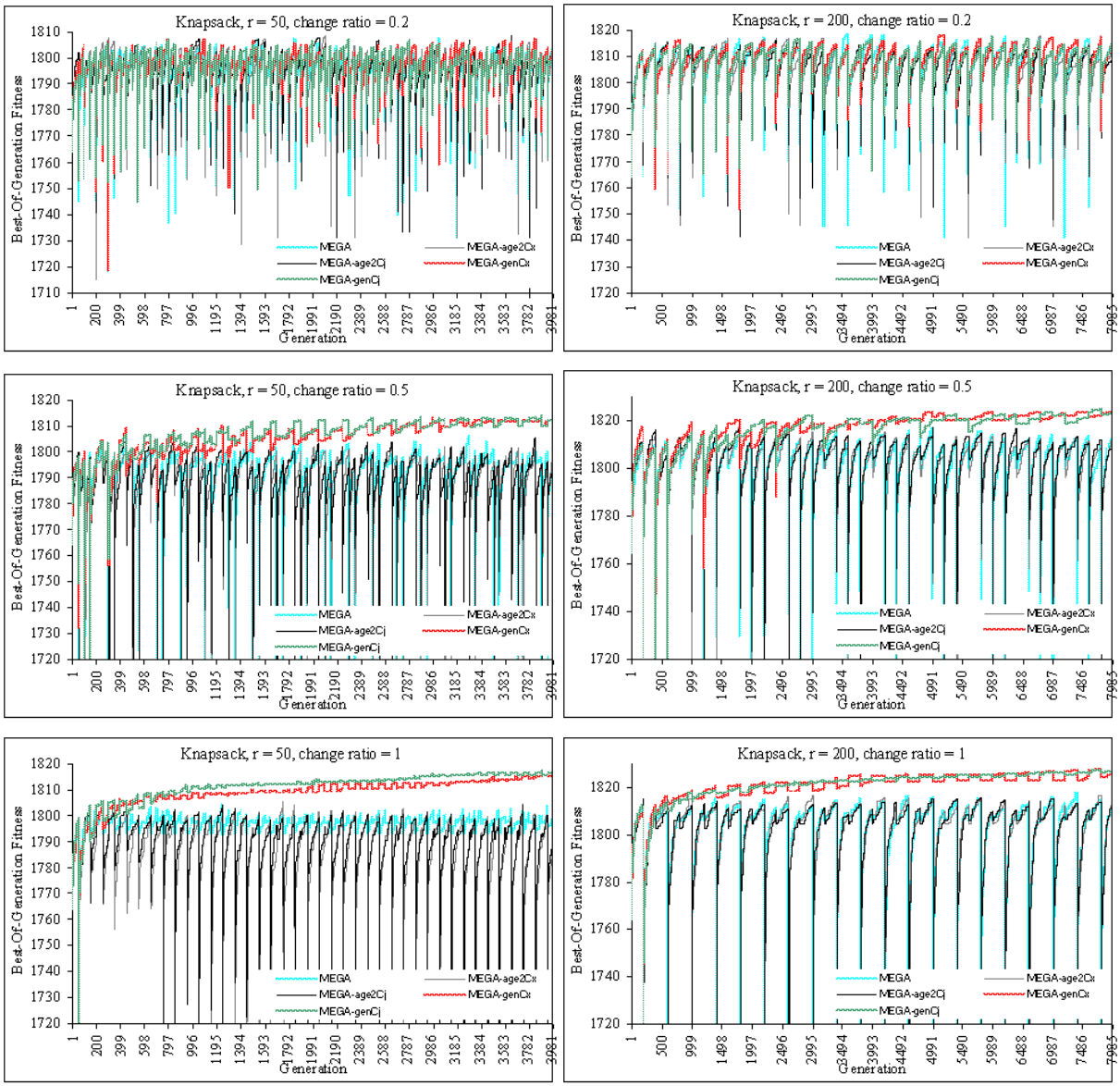


Fig. 12. Comparing MEGAs on dynamic Knapsack, with r = 50 and r = 200, when ρ = 0.2, ρ = 0.5, ρ = 1.0

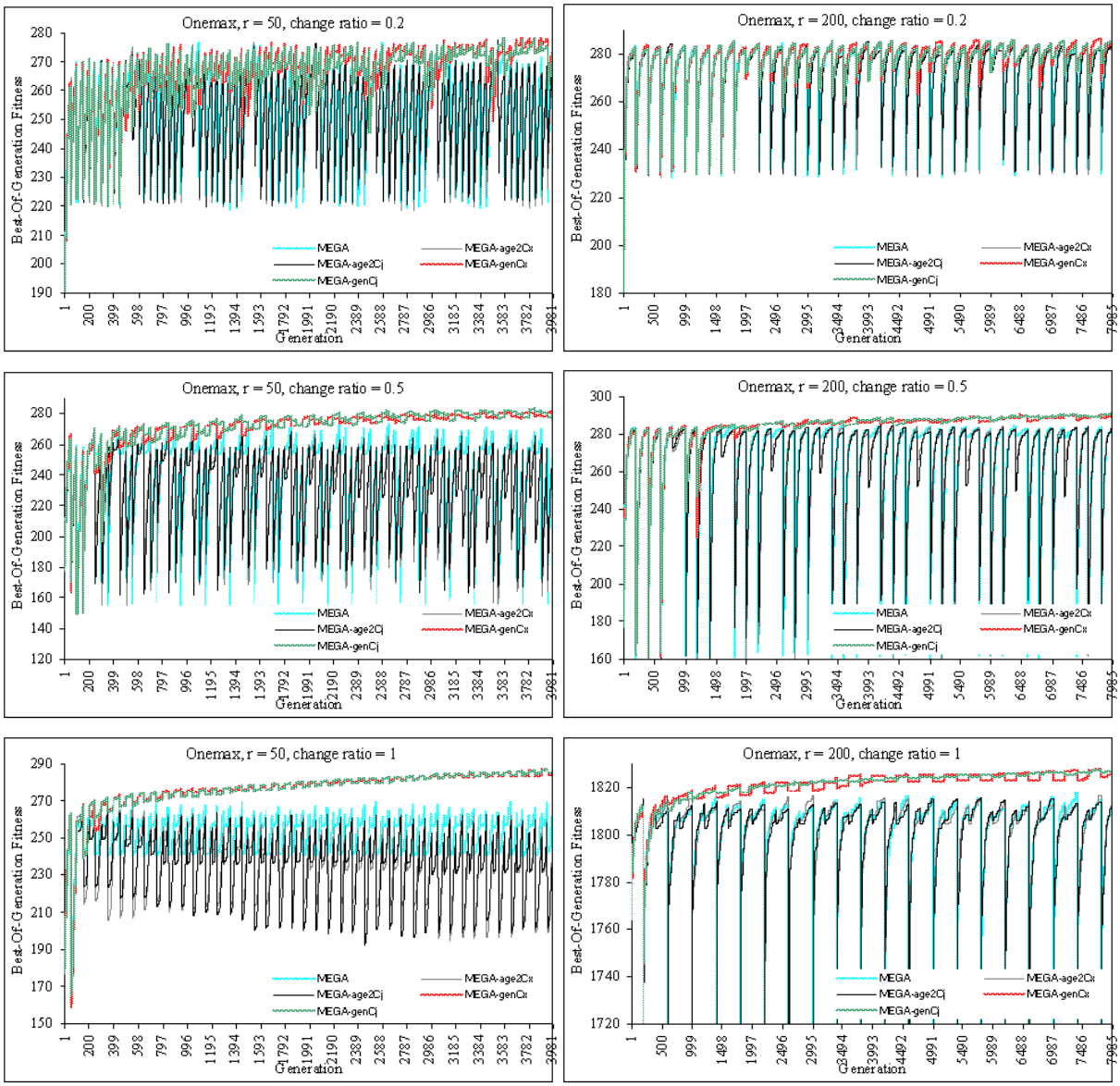


Fig. 13. Comparing MEGAs on dynamic Onemax, with r = 50 and r = 200, when ρ = 0.2, ρ = 0.5, ρ = 1.0

10.3. MIGA RESULTS

Table 9. T-test results comparing MIGAs using the *age2* strategy with *similar* and *gen*

		KP				OneMax			
T-test results	*r* $\rho\rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
MIGA-age2Cx -- MIGA-simCx		+	-	-	-	-	-	+	-
MIGA-age2Cx -- MIGA-simCj		+	-	+	-	-	-	-	-
MIGA-age2Cx -- MIGA-genCx		--	--	--	--	--	--	--	--
MIGA-age2Cx -- MIGA-genCj		--	--	--	--	--	--	--	--
MIGA-age2Cj -- MIGA-simCx	10	++	+	-	-	--	-	+	-
MIGA-age2Cj -- MIGA-simCj		+	+	+	-	-	-	+	-
MIGA-age2Cj -- MIGA-genCx		--	--	--	--	--	--	--	--
MIGA-age2Cj -- MIGA-genCj		--	--	--	--	--	--	--	--
MIGA-age2Cj -- MIGA-age2Cx		+	+	+	-	-	-	+	-
MIGA-age2Cx -- MIGA-simCx		++	++	+	+	--	--	--	-
MIGA-age2Cx -- MIGA-simCj		++	++	-	+	--	--	-	-
MIGA-age2Cx -- MIGA-genCx		++	--	--	--	--	--	--	--
MIGA-age2Cx -- MIGA-genCj		++	--	--	--	--	--	--	--
MIGA-age2Cj -- MIGA-simCx	50	++	++	-	+	--	--	-	-
MIGA-age2Cj -- MIGA-simCj		++	++	-	+	--	--	-	+
MIGA-age2Cj -- MIGA-genCx		++	--	--	--	--	--	--	--
MIGA-age2Cj -- MIGA-genCj		++	--	--	--	--	--	--	--
MIGA-age2Cj -- MIGA-age2Cx		-	+	-	+	-	-	+	+
MIGA-age2Cx -- MIGA-simCx		++	++	++	-	-	--	--	+
MIGA-age2Cx -- MIGA-simCj		++	++	++	-	-	--	-	+
MIGA-age2Cx -- MIGA-genCx		++	+	--	--	-	--	--	--
MIGA-age2Cx -- MIGA-genCj		++	-	--	--	-	--	--	--
MIGA-age2Cj -- MIGA-simCx	100	++	++	++	-	+	--	-	-
MIGA-age2Cj -- MIGA-simCj		++	++	++	-	+	--	-	+
MIGA-age2Cj -- MIGA-genCx		++	+	--	--	-	--	--	--
MIGA-age2Cj -- MIGA-genCj		++	+	--	--	-	--	--	--
MIGA-age2Cj -- MIGA-age2Cx		+	+	+	+	+	+	+	-
MIGA-age2Cx -- MIGA-simCx		++	++	++	++	++	++	+	++
MIGA-age2Cx -- MIGA-simCj		++	++	++	+	++	++	+	+
MIGA-age2Cx -- MIGA-genCx		++	+	--	--	++	--	--	--
MIGA-age2Cx -- MIGA-genCj		++	++	--	--	++	--	--	--
MIGA-age2Cj -- MIGA-simCx	200	++	++	++	++	++	++	+	++
MIGA-age2Cj -- MIGA-simCj		++	++	++	+	++	++	+	+
MIGA-age2Cj -- MIGA-genCx		++	+	--	--	++	--	--	--
MIGA-age2Cj -- MIGA-genCj		++	++	--	--	++	--	--	--
MIGA-age2Cj -- MIGA-age2Cx		-	+	-	+	+	+	-	+

Table 10. T-test results comparing MIGAs using the *gen* strategy with *similar* and *age2*

		KP				OneMax			
T-test results	r $\rho\rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
MIGA-genCx -- MIGA-age2Cx	10	+	++	++	+	++	++	++	++
MIGA-genCx -- MIGA-age2Cj		+	++	++	++	++	++	++	++
MIGA-genCx -- MIGA-simCx		+	+	+	++	++	++	++	++
MIGA-genCx -- MIGA-simCj		+	+	++	++	++	++	++	++
MIGA-genCj -- MIGA-age2Cx		+	++	++	+	++	++	++	++
MIGA-genCj -- MIGA-age2Cj		+	++	++	++	++	++	++	++
MIGA-genCj -- MIGA-simCx		+	+	++	++	++	++	++	++
MIGA-genCj -- MIGA-simCj		+	+	++	+	++	++	++	++
MIGA-genCj -- MIGA-gen Cx		-	+	+	-	+	-	+	+
MIGA-genCx -- MIGA-age2Cx	50	-	+	++	++	++	++	++	++
MIGA-genCx -- MIGA-age2Cj		-	+	++	++	++	++	++	++
MIGA-genCx -- MIGA-simCx		+	+	++	++	++	++	++	++
MIGA-genCx -- MIGA-simCj		+	+	++	++	++	++	++	++
MIGA-genCj -- MIGA-age2Cx		-	+	++	++	++	++	++	++
MIGA-genCj -- MIGA-age2Cj		-	++	++	++	++	++	++	++
MIGA-genCj -- MIGA-simCx		+	++	++	++	++	++	++	++
MIGA-genCj -- MIGA-simCj		+	+	++	++	++	++	++	++
MIGA-genCj -- MIGA-gen Cx		-	-	-	-	-	-	-	-
MIGA-genCx -- MIGA-age2Cx	100	-	-	++	++	+	++	++	++
MIGA-genCx -- MIGA-age2Cj		-	-	++	++	+	++	++	++
MIGA-genCx -- MIGA-simCx		+	++	++	++	++	++	++	++
MIGA-genCx -- MIGA-simCj		+	+	++	++	+	++	++	++
MIGA-genCj -- MIGA-age2Cx		-	++	++	++	++	++	++	++
MIGA-genCj -- MIGA-age2Cj		-	-	++	++	++	++	++	++
MIGA-genCj -- MIGA-simCx		+	+	++	++	++	++	++	++
MIGA-genCj -- MIGA-simCj		+	+	++	++	++	++	++	++
MIGA-genCj -- MIGA-gen Cx		+	+	-	+	-	+	-	+
MIGA-genCx -- MIGA-age2Cx	200	-	-	++	++	--	++	++	++
MIGA-genCx -- MIGA-age2Cj		-	-	++	++	--	++	++	++
MIGA-genCx -- MIGA-simCx		+	+	++	++	++	++	++	++
MIGA-genCx -- MIGA-simCj		+	+	++	++	++	++	++	++
MIGA-genCj -- MIGA-age2Cx		-	-	++	++	--	++	++	++
MIGA-genCj -- MIGA-age2Cj		-	-	++	++	--	++	++	++
MIGA-genCj -- MIGA-simCx		+	+	++	++	++	++	++	++
MIGA-genCj -- MIGA-simCj		+	+	++	++	++	++	++	++
MIGA-genCj -- MIGA-gen Cx		+	+	-	-	+	-	-	+

Table 11. Global results obtained by MIGAs in the Dynamic Knapsack Problem

Change Ratio	Change Speed	MIGA	MIGA-age2Cx	MIGA-age2Cj	MIGA-genCx	MIGA-genCj
0.1	10	1783.6	1785.8	1786.0	**1793.2**	1790.9
	50	1794.7	**1808.2**	1808.1	1803.9	1803.4
	100	1799.5	1817.6	**1817.8**	1808.4	1809.4
	200	1803.8	**1823.9**	1823.5	1814.8	1816.0
0.2	10	1783.7	1783.1	1784.3	1791.8	**1794.2**
	50	1793.8	1798.7	1799.9	**1805.4**	1804.0
	100	1797.2	1809.2	**1809.8**	1809.1	**1809.8**
	200	1801.6	1818.0	**1818.7**	1815.4	1816.9
0.5	10	1784.6	1783.3	1784.1	1803.4	**1804.2**
	50	1791.2	1792.9	1788.4	**1822.5**	1822.4
	100	1791.7	1797.9	1799.8	**1823.3**	**1823.3**
	200	1800.8	1810.4	1809.6	**1829.2**	1828.9
1	10	1788.8	1785.4	1784.9	**1810.0**	1808.6
	50	1800.1	1801.9	1802.2	**1830.5**	1829.6
	100	1793.8	1791.3	1793.7	1830.0	**1831.2**
	200	1799.0	1804.3	1804.8	**1834.1**	1833.7

Table 12. Global results obtained by MIGAs in the Dynamic Onemax Problem

Change Ratio	Change Speed	MIGA	MIGA-age2Cx	MIGA-age2Cj	MIGA-genCx	MIGA-genCj
0.1	10	225.8	222.6	222.2	240.9	**241.1**
	50	264.3	261.4	261.0	**265.2**	265.1
	100	271.4	271.3	271.5	**272.1**	271.8
	200	276.7	280.2	**280.4**	278.9	279.1
0.2	10	212.7	211.9	211.7	**237.2**	236.6
	50	252.5	249.9	249.3	**264.3**	263.9
	100	265.1	263.4	263.8	271.1	**271.9**
	200	273.3	274.4	274.7	**280.5**	279.6
0.5	10	198.8	199.7	201.7	249.0	**249.5**
	50	240.1	235.8	236.5	**280.4**	279.8
	100	252.4	250.2	250.6	**284.6**	284.1
	200	266.2	266.8	266.8	**292.5**	292.4
1	10	212.5	205.7	204.0	266.5	**268.3**
	50	257.9	255.6	257.5	**292.8**	292.4
	100	252.0	252.2	251.1	293.7	**293.9**
	200	262.4	264.9	265.1	296.9	**297.0**

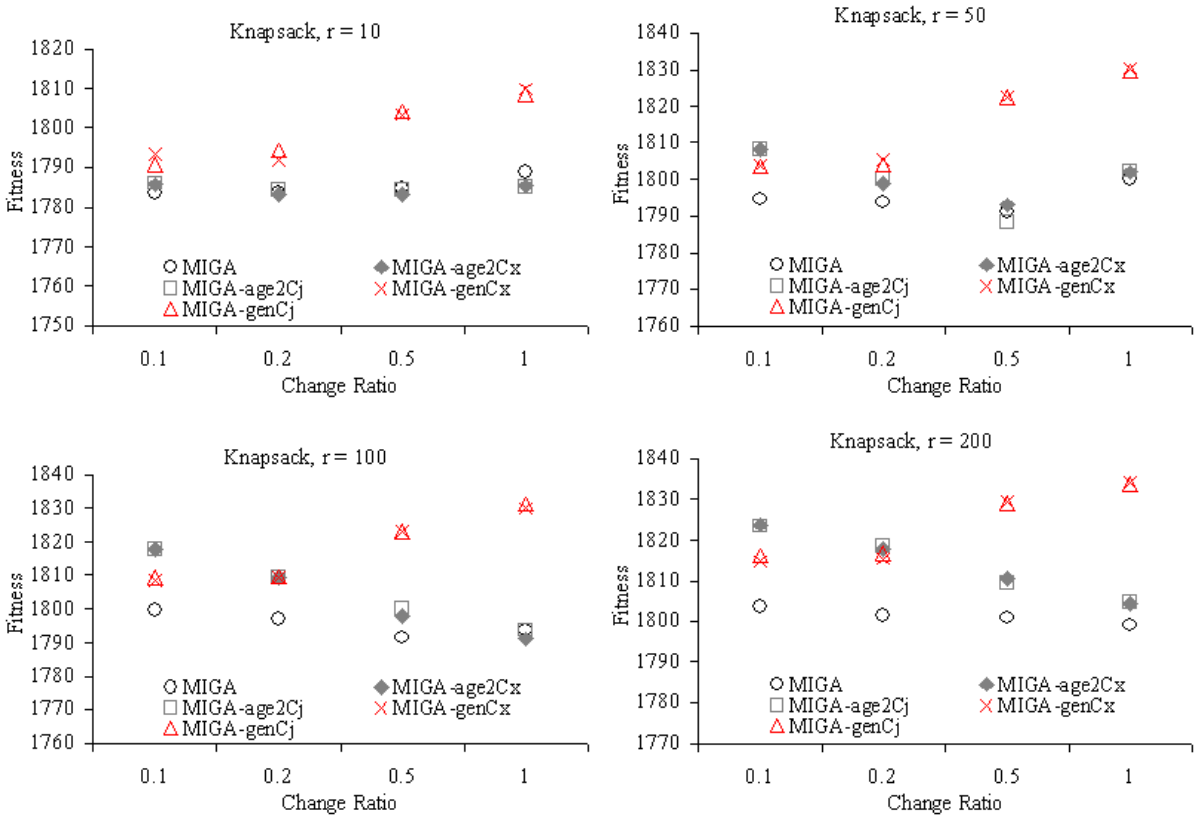


Fig. 14. Global results obtained by MIGAs in the Dynamic Knapsack Problem

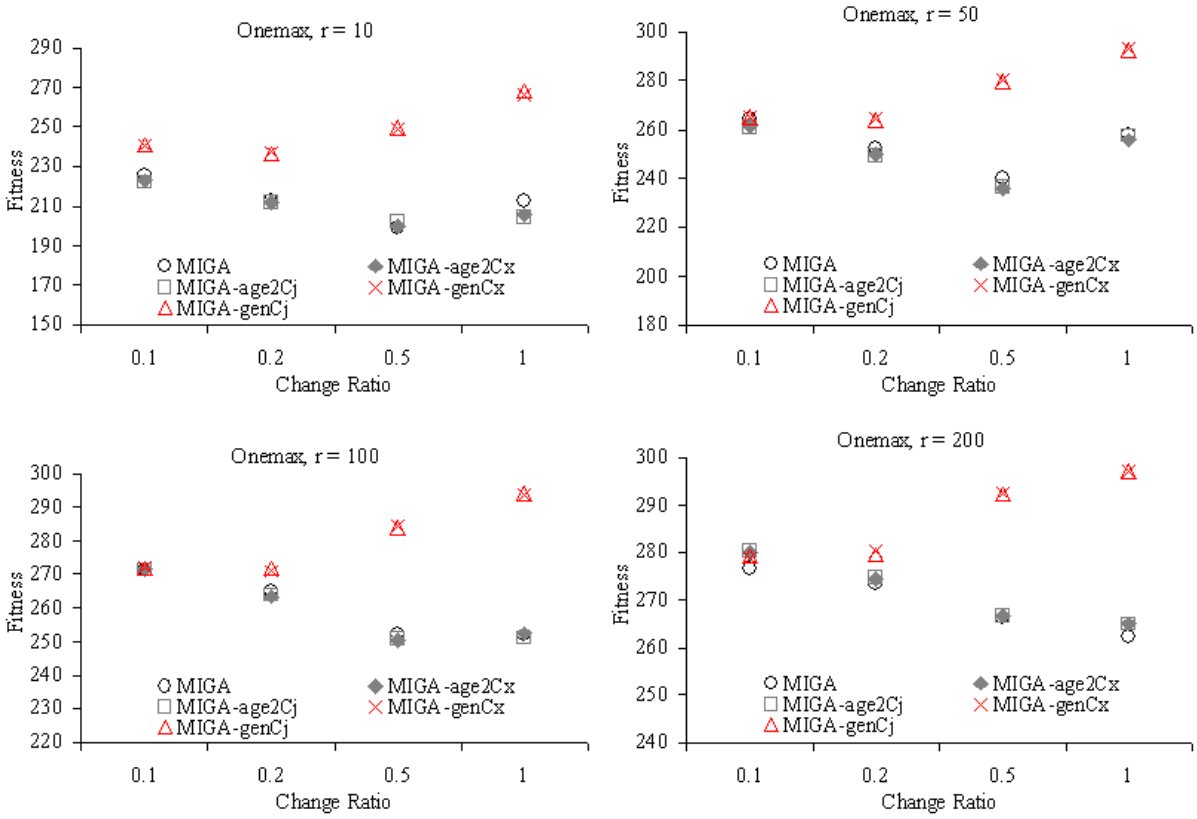


Fig. 15. Global results obtained by MIGAs in the Dynamic Onemax Problem

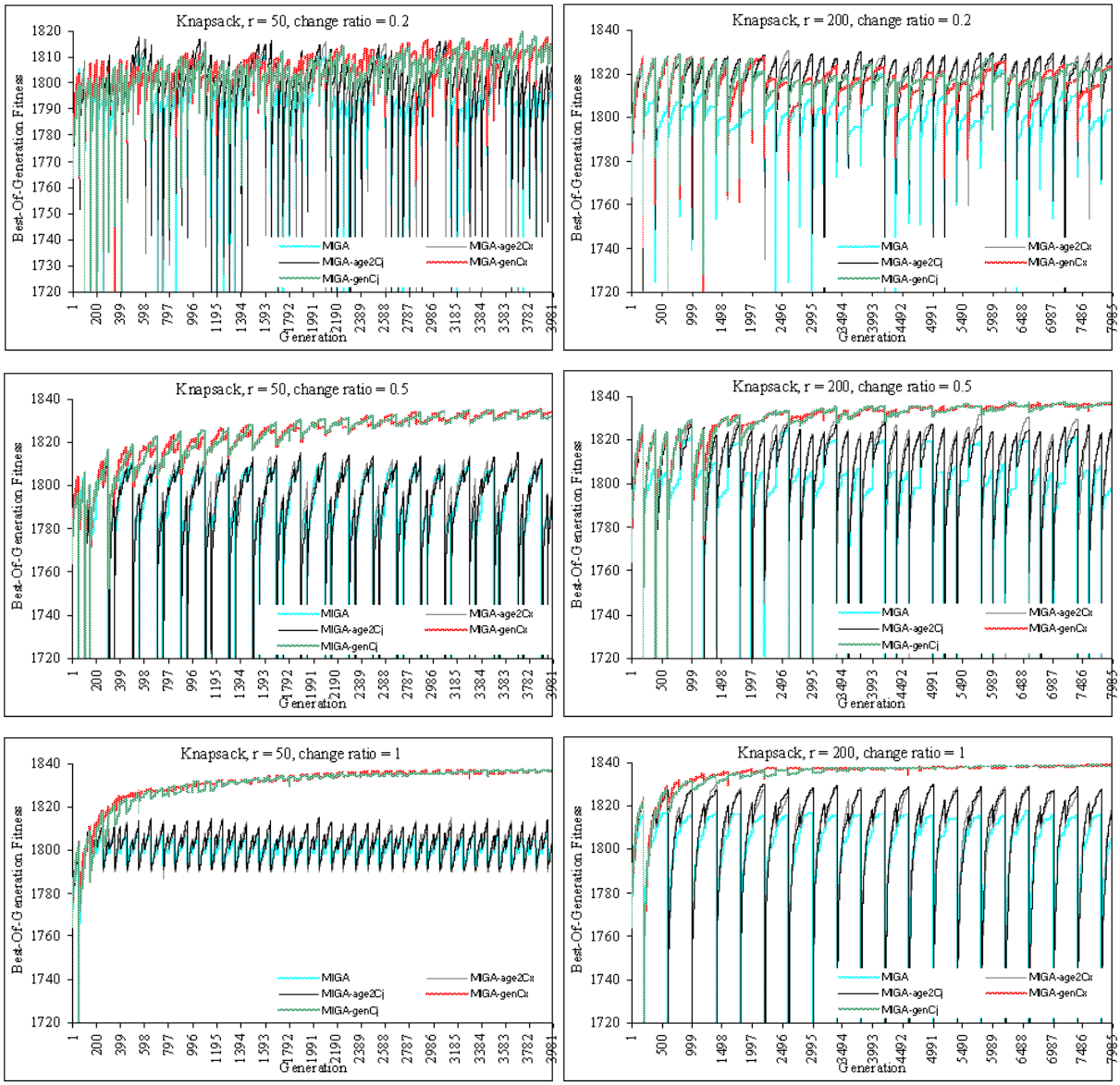


Fig. 16. Comparing MIGAs on dynamic Knapsack, with r = 50 and r = 200, when ρ = 0.2, ρ = 0.5, ρ = 1.0

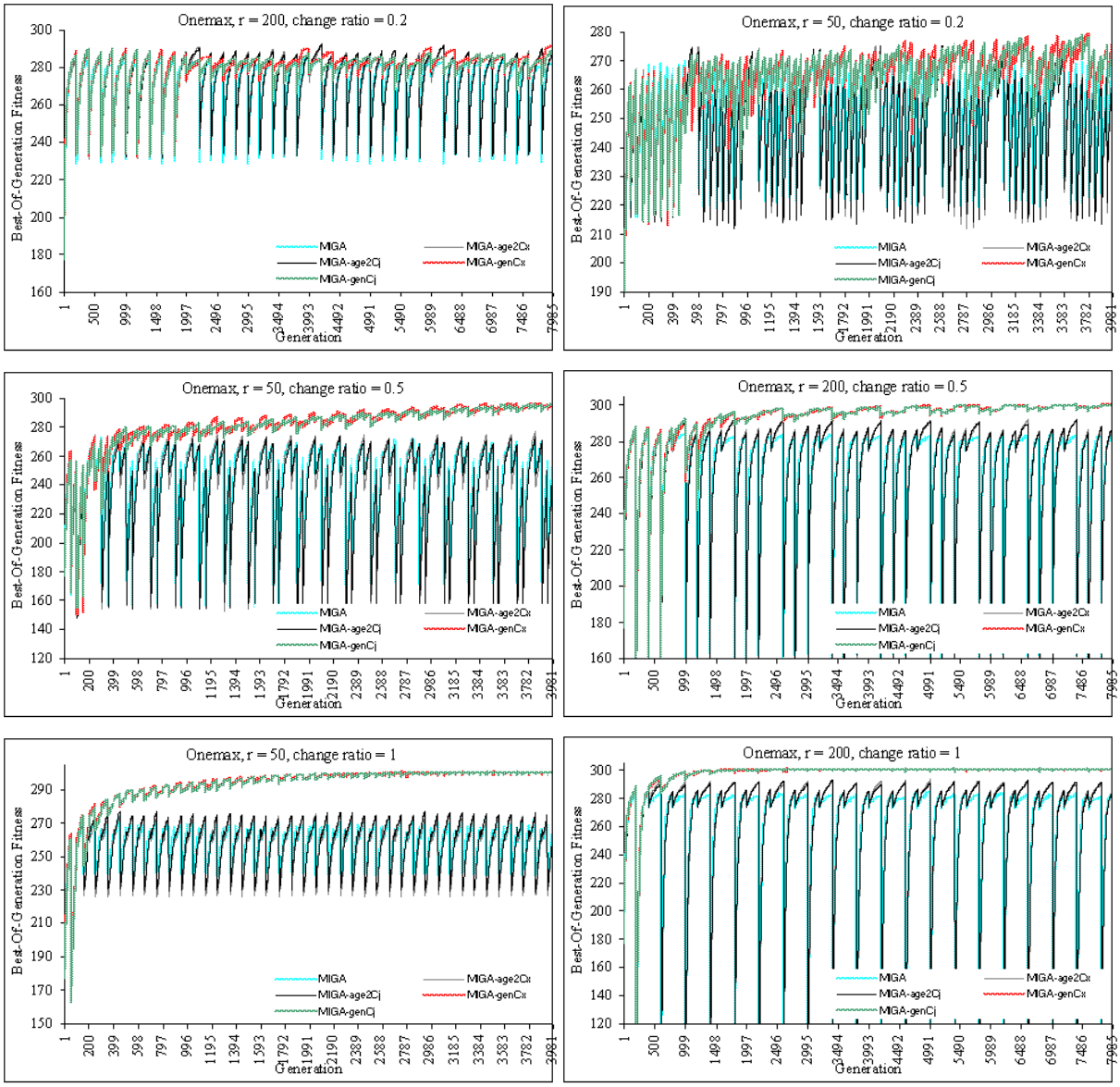


Fig. 17. Comparing MIGAs on dynamic Onemax, with r = 50 and r = 200, when ρ = 0.2, ρ = 0.5, ρ = 1.0