The Influence of Population and Memory Sizes on the Evolutionary Algorithm's Performance for Dynamic 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 2008/02 - ISSN 0874-338X **April 2008**

Abstract. Usually Evolutionary Algorithms keep the size of the population fixed. Nevertheless, in Evolutionary Algorithms dealing with stationary problems some work has been done involving the idea of adapting the population's size along generations. In the context of dynamic environments, less attention has been devoted to the choice of this parameter. In fact, approaches based on the idea of dividing the main population into two, one that evolves as usual and the other one that plays the role of memory of past good solutions. Typically, in these approaches the size of these two populations is chosen off-line and kept constant. Usually memory size is chosen as a small percentage of population size, but this decision can be a strong weakness in algorithms dealing with dynamic environments. Recent work which makes possible changing the size of the population and memory during a run proved that the performance of evolutionary algorithm can be considerably improved. In this work we do an experimental study about the importance of this parameter for the algorithm's performance. In a first set of experiments the size of each component was kept constant but the relative proportion was changed; in a second set of runs we used an algorithm where the size of the two populations could change and compare it with the other fixed size schemes. Results show that tuning the population and memory sizes is not an easy task and the impact of that choice on the algorithm's efficacy is significant. Using an algorithm that dynamically adjusts the population and memory sizes outperforms the previous approach.

Keywords: Evolutionary Algorithms, Dynamic Environments, Population size, Memory size, Control Parameters

1 Introduction

Evolutionary Algorithms (EAs) have been used with success in a wide area of applications, either static or dynamic. Traditional EAs usually have a number of control parameters that must be specified before starting the algorithm. These parameters include, for example, the probabilities of mutation and crossover or the population size. The effects of setting the parameters of EAs has been the subject of extensive research ([1], [2], [3]) and in the last years several approaches using self-adaptive EAs, which can adjust the parameters on-the-fly, have been proposed ([4], [5], [6]). The study of EAs for ***stationary domains*** had been focused in the adjustment of parameters of genetic operators ([7]). The choice of the size of the population has received less attention by researchers. However, if we look to natural systems that inspire EAs, the number of individuals of a certain species changes over time and tends to become stable around appropriate values, according to environmental characteristics or natural resources ([8]). In evolutionary computation the population size is usually an unchanging parameter, kept constant during a run. The off-line specification of this parameter can be problematic: if is too small the EA may not be able to find good solutions; if is too large, too much computational effort is spent. Finding an appropriate population size is a difficult task and several adaptive population sizing methods have been suggested ([9], [10], [11], [12], [13]).

When EAs are used to deal with ***dynamic environments***, some modifications have to be introduced to avoid premature convergence since convergence is disadvantageous when a change happens. These enhancements include increasing diversity after a change ([14], [15]), maintaining diversity throughout the run ([16], [17], [18], [19]), using memory mechanisms ([20], [21], [22], [23], [24], [25], [26]) and multipopulation schemes ([27], [28]). Although these modifications, the global functioning of EAs designed to deal with dynamic applications had inherited most of the characteristics of EAs used for static domains: the adjustment of mutation rate when a change is detected, the use of typical values for the crossover operator of the specification of unchanging populations sizes. Even, when an explicit memory is used, its size is also set at the beginning and is usually a small percentage of the main population size. Not much attention has been devoted to study the influence that population size can have in the performance of the EA. It is also a fact that, in approaches using memory, no validation exists for the use of a specific size.

Some research has been made about this issue: Schönemann ([29], [30]) studied the impact of population size in the context of Evolutionary Strategies for dynamic environments, and concluded that the choice of the population size can be a determinant factor in certain classes of problems and, therefore, more research should be done. Simões et al. ([31]) proposed an EA to deal with changing environments which controls the size of population and memory during the run. Results from that work show that the performance of the EA can be considerably improved. Recently, Richter et al. ([32]) proposed a memory-based abstraction method using a grid to memorize useful information and the results obtained suggest that an optimal grid size depends on the type of dynamics and authors claim that the use of an adaptive grid size would increase the performance of the abstraction memory, indicating this issue as future research.

In this work we describe an extensive empirical study that we made and whose focus was the performance of distinct memory-based EAs that faced different environmental characteristics. The study is based on two benchmark problems and is divided in two parts: in the first part, we kept the populations sizes fixed and ran the EAs for different values for these parameters in order to analyze the impact of the chosen values on the EA's performance. In the second part, an EA using population and memory of changing sizes was ran and compared with the results of the previous experimentation. We show that the values set to population and memory sizes can have a great influence in the EA's performance. The use of a more flexible algorithm that can dynamically adjust the population and memory sizes during the run outperforms, in general, all the scenarios analyzed using unchanging sizes. These results are statistically supported.

The rest of the report is organized as follows: next section briefly reviews memory schemes for EAs in dynamic environments. Section 3, describes the three implemented memory-based EAs and Section 4 details the dynamic test environments used for this study. The experimental results and the analysis are presented in section 5. Section 6 concludes the report and some considerations are made about future work.

2 Memory-based Evolutionary Algorithms for Dynamic Environments

EAs using memory schemes work by storing useful information about the different environments that may occur and selectively retrieving that information later when another change in the environment is detected. When the environment is cyclic and different states repeatedly appear over time, memory-based Evolutionary Algorithms are good approaches to use, for the use of information from the past can help the EA to quickly readapt each time a previous situation reappears. Information can be stored in memory implicitly or explicitly ([33]). Next sections explain these two memory mechanisms.

2.1 Implicit Memory

In the case of *implicit memory* approaches the stored information is kept in redundant representations of the genotype. There are some variants, e.g., using diploid (or multiploid) representations ([34], [35], [36]) or dualism mechanisms ([37], [38], [39]). The idea of using diploid representations was suggested by Goldberg and Smith ([34]) as an extension of the standard GA. Diploidy was particularly studied in the context of dynamic environments ([34], [35], [36]). 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 dominance mechanism.

Dualism mechanisms are inspired by the complementarity and dominance schemes found in nature. Yang ([38]) proposed a new genetic algorithm called primal-dual genetic algorithm (PDGA) which operates on a pair of chromosomes that are primal-dual to each other in the sense of maximum distance of the corresponding genotypes, e.g., the Hamming distance for binary representations. With PDGA during the phase of survivors' selection a set of low fit individuals is chosen and their corresponding duals are evaluated. If they are better they have a chance to pass to the next generation.

2.2 Explicit Memory

In *explicit memory* schemes an extra space, called memory population, is used. Its main goal is to store useful information about the current environment and possibly reuse it each time a change occurs. It may also permit the population to move to a different area in the landscape in one step, which in principle is harder when we rely only on standard genetic operators. ([33]). The stored information can be the current best individual of the population, information about the environment or both. Another approach proposed by ([32]) does not store explicitly good solutions but an abstraction, i.e, the approximate location of the individual in the search space.

Explicit memory approaches were recently divided into direct memory and associative memory schemes ([26]). In direct memory schemes the information stored in memory consists of past good solutions that are directly used when a change is detected. Associative memory schemes store useful information about the environment as well as good solutions and both are retrieved when a change happens.

Examples of direct memory approaches can be found in ([20], [31], [17], [40]) and for associative memory schemes see ([21], [26], [40]).

2.3 Considerations about Memory

When using memory-based EA some questions can be asked concerning memory: 1) when and which individuals to store in memory, 2) which size to choose, 3) which individuals should be replaced when memory is full, 4) which individuals should be selected from memory and introduced in the population when a change happens ([33]).

As stated before, typically, in explicit memory schemes, the information stored in memory corresponds to the current best individual of the population. Other approaches, together with this information, also keep information about the environment ([24], [25], [26]). Ritcher et al. ([32]) propose a memory scheme based on the idea of abstraction. The abstract storage process consists of two steps: a selection process and a memorizing process. The selecting process picks good individuals from the population while the EA runs. During the memorizing process, the selected individuals are sorted according to the partition of the search space they represent. So, what is memorized is no longer the individual itself but an abstraction, i.e., its approximate location in the search space.

Concerning the size of the memory, no intensive investigation has been made yet. In this work we will be focused in this topic to make some conclusions about the best choice for the memory (and population) size. The size of the memory is an important issue. Since the size of the memory is limited, it is necessary to decide which individuals should be replaced when new ones need to be stored. This process is usually called *replacing strategy*. Branke ([20]) compares different replacement strategies for inserting new individuals into the memory. The most popular is called **similar** and consists in selecting the individual in memory most similar to the new one replacement taking place if the latter has better fitness. Simões and Costa ([31]) proposed two replacing strategy based on the **aging** of memory individuals. In the first one, all individuals of the memory start with an age equal to zero, and at every generation their age is increased by one. Besides, if they were selected to the population when a change occurs, their age is increased by a certain value until a limit age is reached in which case the age is reset to zero. When it's necessary to update memory, the youngest one is selected to be replaced. In a different age-based replacing scheme the age of each individual is calculated as a linear combination of its actual age and its fitness. Besides, in this scheme, memory individuals never die, i.e., their age is not reset to zero. This way, individuals that last long in memory and contributed to the evolutionary process are not penalized. Another replacing strategy, called **generational**, was proposed in ([23]). This scheme selects for replacement the worst individual present in the memory since the last environmental change. For instance, if last change occurred at generation t_1 and currently the algorithm is in generation t_2, when it is time to insert an individual into the memory the worst individual that was stored between generation t_1+1 and t_2-1 will be replaced. If no individual has been stored since last change, the similar strategy is used and the closest individual in terms of Hamming distance, if it is worse than the current best, is replaced. The generational replacing strategy was tested in several memory-based EA and significantly improved its performance ([41]). The retrieval of memory information when a change occurs depends on what was stored. In some cases, memory is re-evaluated and the best individuals of memory replace the worst of population. Other approaches use the stored environmental information together with the best memory individual to create new information to introduce in the population when a change happens.

2.4 Population and Memory Sizes

Usually, memory-based EAs for changing environments use a memory of small size, comparing with the dimension of the main population. In most cases, the dimension of memory is chosen between 5% and 20% of the population size, with 10% the most chosen one. A brief listing of the values used in several studies is presented:

The enhanced memory EA proposed by Branke ([20]) uses a population of 90 individuals and a memory with size 10. Karaman et al. ([21]) propose a memory indexing EA and compare it with Branke's algorithm. In both situations memory is set to 10 individuals, while the population size is 50.

Mori et al. ([22]) present a memory-based termodynamical GA which uses a memory of size 8 and a population with 46 individuals.

Simões et al. ([16]) propose a GA inspired in the natural immune system. The algorithm uses a search population of size 100 and a memory of 20 B-cells. Other empirical studies of these authors ([31], [41]) compare several memory-based EAs using memory of size 10 and population with 110 chromosomes.

The approach suggested by Trojanosky et al. ([24]) uses a population with 100 individuals and memory size equal to 20. EAs using associative memory schemes studied in ([39], [25]) uses memory of size 10 and population with 110 chromosomes. Other studies using diverse memory-based EAs for dynamic environments can be found in ([17], [18], [40]) and use population and memory sizes of 100 and 10 individuals, respectively.

As we can see, choosing a constant value for population and memory sizes is widely used in memory-based EAs. Memory is always seen as playing a secondary role in the process, and is used always with a smaller dimension. Some authors start questioning that generalized assumption. Ritcher et al. ([32]) use a matrix with fixed dimension to store the abstraction of the good solutions and studied the influence of this parameter in the performance of the algorithm. This author claims that an optimal grid size depends on the type of dynamics and the size of the bounded region in search space which the memory considers and suggests the investigation of an adaptive grid size. Schönemann ([29]) also points out that it is worthwhile to study approaches where the size of the population should be dynamically adjusted. An EA that uses the global individuals of population and memory in a more flexible method was suggested by ([31]). Simões et al. proposed an EA which adapts the population and memory sizes during the run. The obtained results proved that the EA's performance was significantly improved.

3 Description of the Implemented Memory-based EAs

In this section we will describe the algorithms that were used to perform the empirical study about the issue of the importance of the sizes of the populations. We will be interested in analyzing each different algorithm and see if there is a correspondence between the choice for the population and the memory size and the performance of those algorithms.
We choose to implement a memory-immigrant based EA and a direct memory EA. Both use population and memory with unchanging sizes. An additional algorithm, using a direct memory scheme with changing population and memory sizes was implemented, ran and compared with the other two. The three algorithms will be briefly described in following sections. Additional information can be found in the original works already cited.

3.1 Memory-Immigrants Genetic Algorithm

The Memory Immigrants Genetic Algorithm (MIGA) is an EA using a direct memory scheme. It was proposed by ([17]) and works in the following way: the algorithm evolves a population of individuals in the standard evolutionary way: selection, crossover and mutation. Additionally, a memory is initialized randomly and used to store the current best individual of the population, replacing an initially created random individual (if it exists) or replacing the most similar in memory if it is better. Every generation the best individual in memory is used to create a set of new individuals, called immigrants that are introduced into the population replacing the worst ones. These new individuals are created mutating the best solution in memory using a chosen mutation rate. The number of solutions created is a percentage of the population size. When a change is detected nothing is done and it is expected that the diversity introduced in the population by adding these immigrants can help the EA to readapt to the new conditions.

3.2 Memory-Enhanced Genetic Algorithm

The Memory-Enhanced Genetic Algorithm (MEGA) also uses a direct memory scheme ([17]). This algorithm consists in an adaptation of Branke's algorithm ([20]). Later, this algorithm was used in other studies under the general name Direct Memory Genetic Algorithm (DMGA) ([26], [40]).

The algorithm can be described as follows: MEGA is a standard genetic algorithm that evolves a population of $\boldsymbol{n}$ individuals through the application of selection, crossover and mutation. Additionally, a memory of size $\boldsymbol{m}$ is used, starting with randomly created individuals. From time to time memory is updated in the following way: if any of the initial random individuals still exist, the current best solution of the population replaces one of them arbitrarily; if not, the most similar

memory updating strategy is used to choose which individual to exchange. This strategy replaces the closest individual in memory if it is better. Memory is evaluated every generation and, when a change is detected, the memory is merged with the best ***n-m*** individuals of the current population to form the new population, while memory remains unchanged.

3.3 Variable-size Memory Evolutionary Algorithm

Simões and Costa proposed a **Variable-size Memory Evolutionary Algorithm** (VMEA) to deal with dynamic environments. This algorithm uses a population that searches for the optimum and evolves as usual, through selection, crossover and mutation. A memory population is responsible for storing good individuals of the evolved population at several moments of the search process. The two populations - search and memory - have variable sizes that can change between two boundaries. The basic idea of VMEA is to use the limited resources (total number of individuals) in a flexible way. The size of the 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. The memory is evaluated every generation and a change is detected if at least one individual in the memory changes its fitness. If an environmental modification is detected, the best individual of the memory is introduced into the population. In the case of either the population size or the sum of the two populations has reached the allowed maximum, the best individual in memory replaces the worst one in the current population. The algorithm was compared with other memory-based schemes using the standard dimensions for population and memory and the results validate its effectiveness. More details about this algorithm can be found in ([42], [31]).

4 Experimental Design

4.1 Dynamic Test Environments

The dynamic environments to test our approach were created using Yang's Dynamic Optimization Problems (DOP) generator ([26]). This generator allows constructing different dynamic environments from any binary-encoded stationary function using the bitwise exclusive-or (XOR) operator. The basic idea of the generator is to perform the operation x⊕M to an individual *x*, 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 also allows the definition of problems where the changes can be cyclic, cyclic with noise or non-cyclic. In the first case, several masks are generated according to the 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.

In this work we constructed 16 cyclic DOPs, setting the parameter ***r*** to 10, 50, 100 and 200. The ratio ρ was set to different values in order to test different levels of change: 0.1 (a light shifting) 0.2, 0.5 and 1.0 (severe change). This group of 16 DOPs was tested using the three different algorithms in two benchmark problems, using different population and memory sizes. A total of **960** different situations were tested in this work.

The DOP was applied to two benchmark problems, used to test the different EAs with different parameter settings: the dynamic Knapsack problem and the Onemax problem.

4.2 Knapsack Problem

The knapsack problem is a NP-complete combinatorial optimization problem often used as benchmark. It consists in selecting a number of items (*m*) to a knapsack with limited capacity. Each item has a value (v_i) and a weight (w_i) and the objective is to choose the items that maximize the total value, without exceeding the capacity of the bag (*C*):

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

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 (m=100) using strongly correlated sets of randomly generated data constructed in the following way ([43], [39]):

w_i = uniformly random integer [1, 50] (3)

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

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

4.3 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.

4.4 Parameters Setting

The EA's parameters were set as follows: generational replacement with elitism of size one, tournament selection with tournament of size two, uniform crossover with probability p_c=0.7 and mutation with probability p_m=0.01. Binary representation was used with chromosomes of size 100 for the Knapsack and 300 for Onemax problem. The probability of flip mutation and the ratio of immigrants introduced in population used in MIGA were 0.01 and 0.1, respectively.

All simulations used a global number of individuals equal to **100**. These individuals were divided in population and memory, in the following way: Population varied between 10 and 90 individuals, in increases of 10 (10, 20, 30 ,… 90). Memory size was calculated with the remaining individuals: M(size)=100–P(size), i.e., 90, 80, 70, …, 10.

The **generational** replacing strategy proposed by [23] was used in all EAs. For each experiment of an algorithm, 30 runs were executed for 200 environmental changes. In each case the algorithm ran ***r*number_of_changes*** generations.

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:

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

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

5 Results

In this section we will show the obtained results concerning the overall performance of the studied algorithms. Results showing the adaptability of the algorithms along time will also be presented. We will also analyze the diversity in population and memory for the different algorithms, as well as the variation of population and memory sizes in VMEA.

5.1 Global Results

The experimental results regarding MIGA, MEGA and VMEA in the two studied benchmarks under different environments are shown in Table 1. Best results obtained by MIGA and MEGA are marked in bold. The blue shadowed line reports the results obtained by VMEA which can be compared with the first two.

As we see, depending on the change period and the change ratio, the choice of population and memory sizes affects the algorithm's performance. For the Knapsack problem, using MIGA, the best choice was population with 60 or 50 individuals (and memory of 40 and 50 respectively), for dynamics changing rapidly (*r = 10*). In environments with slower changes (*r = 50, r = 100, r = 200*), the best results were achieved with population size of 80 or 70 and memory with 20 or 30 individuals, respectively. Either smaller or larger populations lead to a decrease of the EAs' performance. For the Onemax problem, MIGA obtained best results using population size of 40 individuals using faster changes and lower change ratios *(ρ = 0.1* and *ρ = 0.2*) and population with 60 individuals for larger change ratios (*ρ = 0.5* and *ρ = 1.0*). Increasing the change ratio and the change period, larger populations are required to achieve the best results: 70 to 90 individuals.

Using MEGA for the Knapsack problem, in general, the best choice was population with smaller size and larger memory: populations of 40, 30 or 20 and memory with 60, 70 and 80 individuals, respectively. Larger population sizes conjugated with smaller memories were always bad options. When the population size is used with a very small size (10) and larger memory size, the performance was also very poor. For the Onemax problem, analyzing the results obtained by MEGA we conclude that smaller populations are needed to achieve the best performance when the change period is small. As the change period increases, larger populations were those that allowed best performances, mainly in the case of lower change ratios.

It is evident that exists an interval of values which optimize the different algorithms' efficacy, and outside that interval its performance decrease. In Table 1 those intervals are marked using a shadow. In all cases, VMEA outperformed all instances of MIGA and MEGA, demonstrating robustness and adaptability in the studied problems under different environmental characteristics.

Figures 1 to 4 show the previous scores in a graphical form. Each plot refers to a different value of the change period r. All graphics have four lines for different change ratios: dashed gray line with squares corresponds to ρ = 1.0; dashed black line with triangles refers to ρ = 0.5; dashed gray line with crosses is used to ρ = 0.2 and the black solid line with circles concerns to ρ = 0.1. In dark red, using the same markers, are the results obtained by VMEA. Observing these plots, it is visible the influence that population and memory sizes can have in the performance of this algorithm. It is evident there must be a tradeoff between memory and population sizes. The usual split of 10% for the memory size and 90% for the population size is not the best choice. As happens in the stationary environments ([7]) it is possible that different sizes of population and memory might be optimal at different stages of the evolutionary process.

	Knapsack								*Onemax*							
Performance	MIGA				MEGA				MIGA				MEGA			
r = 10, ρ ⇒	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
P90_M10	1775.0	1774.3	1779.6	1780.9	1767.7	1764.7	1771.7	1775.3	239.1	231.9	230.4	239.8	232.8	217.0	203.9	209.3
P80_M20	1778.3	1778.8	1784.9	1790.8	1771.0	1770.9	1775.2	1779.7	244.5	236.5	237.7	251.0	238.6	223.4	212.7	221.6
P70_M30	1779.4	1780.8	1787.6	1793.1	1773.3	1773.5	1777.4	1782.7	245.8	238.3	242.1	259.3	240.3	226.9	219.9	230.2
P60_M40	1779.4	**1780.9**	1787.1	1794.9	1775.5	1775.6	1778.5	1784.4	**246.3**	**239.5**	243.9	262.5	241.5	229.5	224.2	237.0
P50_M50	**1779.5**	1780.8	**1788.4**	**1795.7**	1775.6	**1776.8**	1780.1	1787.1	245.9	239.3	246.2	263.0	**242.0**	230.9	228.1	244.6
P40_M60	1778.8	1780.1	1787.1	1795.6	**1775.7**	1776.5	**1781.8**	1790.1	244.8	238.7	**246.4**	**264.0**	241.3	232.0	232.8	250.5
P30_M70	1776.7	1778.1	1784.8	1795.0	1775.3	1775.9	1781.1	**1790.6**	242.8	237.2	245.9	263.9	239.6	**232.7**	235.3	254.9
P20_M80	1773.5	1775.4	1781.1	1790.5	1773.0	1774.4	1780.1	1788.6	238.6	234.5	243.5	262.0	236.9	231.4	237.9	**257.3**
P10_M90	1766.3	1768.2	1775.2	1783.3	1768.4	1768.0	1773.0	1780.7	229.4	226.5	237.9	256.3	229.4	226.8	**238.3**	257.0
VMEA	1782.5	1784.1	1791.6	1799.3	1782.5	1784.1	1791.6	1799.3	249.1	241.5	250.3	266.5	249.1	241.5	250.3	266.5
r = 50, ρ ⇒	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
P90_M10	1798.5	1803.8	1812.3	1816.1	1786.8	1788.3	1796.6	1802.1	271.0	279.7	293.4	297.2	268.6	270.7	276.6	283.6
P80_M20	1804.4	1809.4	1813.3	1817.3	1791.6	1795.8	1798.3	1802.9	280.1	287.0	**294.2**	297.2	274.4	**276.7**	279.2	284.4
P70_M30	**1805.1**	**1809.6**	**1814.5**	**1817.7**	1793.4	1796.7	1799.8	1805.6	**281.8**	**287.6**	294.1	**297.3**	**275.6**	276.7	280.4	286.6
P60_M40	1804.9	1806.9	1813.6	1817.1	1793.4	1797.2	1802.4	1806.7	281.7	286.1	293.1	296.6	274.9	275.6	281.4	287.9
P50_M50	1801.9	1806.6	1812.0	1816.8	1795.1	1797.8	1802.5	1809.1	281.1	285.6	292.9	296.4	274.0	275.2	**281.8**	288.0
P40_M60	1801.0	1804.8	1811.1	1815.5	**1795.5**	1797.6	1803.3	**1810.5**	279.0	283.7	291.3	295.9	272.5	274.5	281.4	288.5
P30_M70	1797.9	1802.2	1808.8	1813.5	1794.5	**1798.5**	1804.1	1810.2	276.5	281.2	290.1	294.6	269.9	272.7	281.3	**288.5**
P20_M80	1795.2	1798.8	1806.7	1812.3	1792.5	1796.3	**1804.6**	1810.2	272.1	276.4	286.7	293.0	266.5	269.9	280.4	288.0
P10_M90	1786.1	1789.5	1800.6	1807.7	1787.6	1790.5	1800.4	1807.9	261.3	265.4	279.6	287.1	260.8	266.3	278.2	287.3
VMEA	1807.2	1811.3	1817.9	1819.6	1807.2	1811.3	1817.9	1819.6	282.6	289.2	296.1	298.4	282.6	289.2	296.1	298.4
r = 100, ρ ⇒	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
P90_M10	1806.1	1812.7	1815.9	1817.7	1792.6	1797.7	1805.8	1808.4	279.5	289.5	**297.5**	298.8	275.5	282.2	287.3	292.0
P80_M20	**1811.5**	**1814.9**	**1817.8**	1819.9	1797.2	1802.7	1807.3	1811.3	288.5	294.1	297.2	**298.8**	280.7	**284.8**	289.0	293.7
P70_M30	1811.3	1814.5	1817.7	**1820.1**	1799.4	1802.8	1808.1	1811.5	**290.6**	**294.2**	297.3	298.7	**280.8**	284.5	**289.7**	294.0
P60_M40	1811.1	1813.2	1817.8	1819.2	1800.3	1803.7	1809.1	1811.9	290.4	294.0	297.3	298.6	280.6	283.3	288.8	**294.2**
P50_M50	1809.3	1812.4	1816.9	1819.2	1800.7	1804.2	1809.3	1813.2	289.2	293.1	296.8	298.3	279.4	282.6	288.6	293.8
P40_M60	1807.7	1811.2	1816.4	1818.0	**1801.4**	1804.4	1810.1	1813.8	287.5	291.7	296.3	298.3	278.3	281.7	288.3	293.7
P30_M70	1805.3	1809.6	1814.7	1817.8	1800.7	**1805.3**	**1811.1**	**1815.2**	285.1	290.1	295.4	297.8	276.5	280.4	287.8	293.4
P20_M80	1801.9	1806.9	1812.3	1816.3	1800.2	1805.0	1810.7	1815.1	281.3	286.2	293.3	296.5	274.2	279.0	288.0	293.5
P10_M90	1794.4	1799.9	1806.8	1813.0	1793.0	1799.4	1807.6	1813.1	270.8	277.3	287.1	292.9	271.0	277.4	287.3	293.4
VMEA	1814.2	1816.2	1819.5	1822.6	1814.2	1816.2	1819.5	1822.6	291.9	296.6	297.8	298.9	291.9	296.6	297.8	298.9
r = 200, ρ ⇒	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
P90_M10	1811.7	1816.0	1818.2	1820.0	1797.3	1805.0	1808.8	1813.5	286.5	297.0	296.9	298.4	279.8	289.1	291.9	296.7
P80_M20	**1816.2**	**1818.2**	**1820.6**	1821.8	1803.0	1806.7	1810.0	1814.6	295.5	**297.7**	**298.8**	299.3	**285.5**	**289.6**	293.7	296.7
P70_M30	1816.0	1817.9	1820.4	**1821.9**	1804.8	1808.1	1811.9	1815.4	**295.7**	297.5	298.8	**299.4**	285.2	**289.6**	293.9	297.3
P60_M40	1815.7	1817.4	1820.2	1821.2	1806.1	1809.2	1813.3	1816.2	295.6	297.2	298.8	299.3	285.1	288.7	**294.0**	**297.3**
P50_M50	1814.6	1816.3	1819.4	1821.3	1806.4	1809.4	1813.9	1816.6	294.6	296.8	298.5	299.2	284.2	288.2	293.8	296.5
P40_M60	1813.5	1815.7	1818.9	1820.7	1806.6	1809.5	1813.9	1816.8	293.6	296.2	298.4	299.1	283.4	287.5	293.3	297.0
P30_M70	1811.5	1813.9	1817.5	1820.3	**1807.6**	1809.7	1814.8	1818.3	292.2	295.2	297.8	298.9	282.3	286.9	293.0	296.4
P20_M80	1808.6	1811.9	1816.4	1818.8	1806.5	**1809.8**	**1815.1**	**1818.9**	288.7	293.1	297.0	298.5	281.0	285.8	293.4	296.4
P10_M90	1801.3	1806.9	1812.7	1817.3	1801.3	1806.8	1813.2	1816.8	279.8	286.1	293.4	296.5	280.0	285.4	293.3	296.8
VMEA	1817.4	1820.4	1822.5	1823.6	1817.4	1820.4	1822.5	1823.6	295.7	298.1	298.9	299.5	295.7	298.1	298.9	299.5

Table 1. Experimental results concerning the overall performance of the algorithms

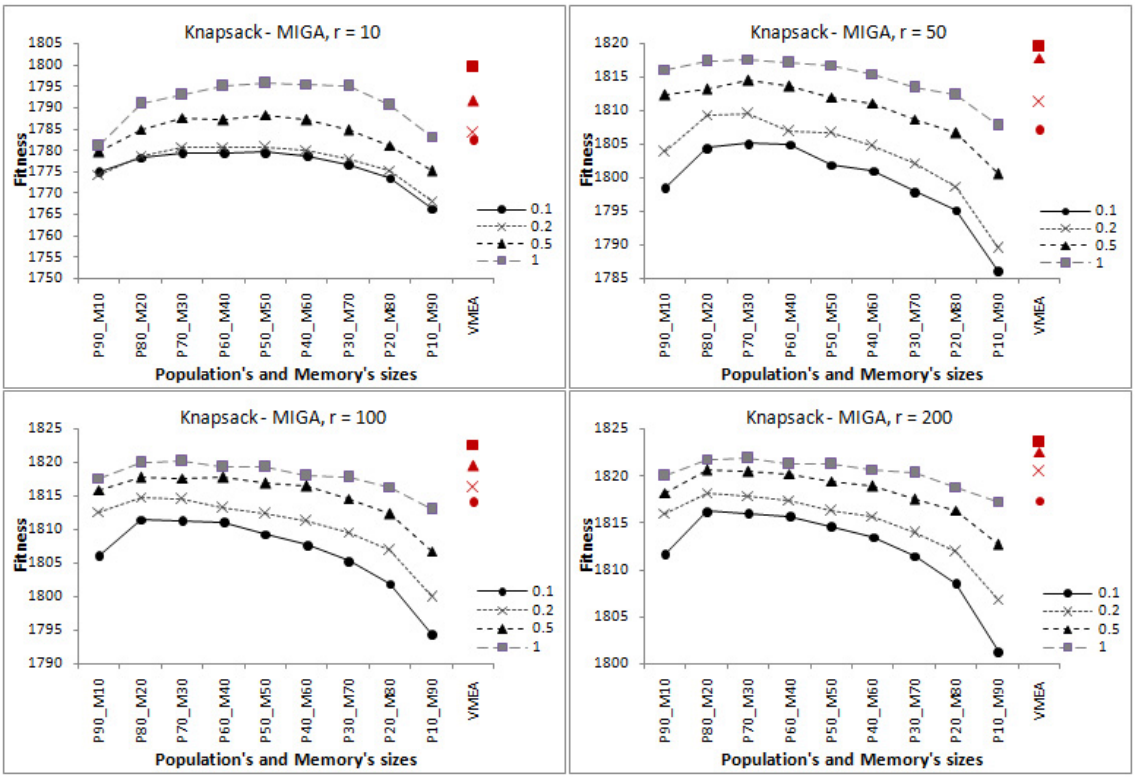

Figure 1. Global results obtained in the dynamic Knapsack problem using MIGA with different population and memory sizes

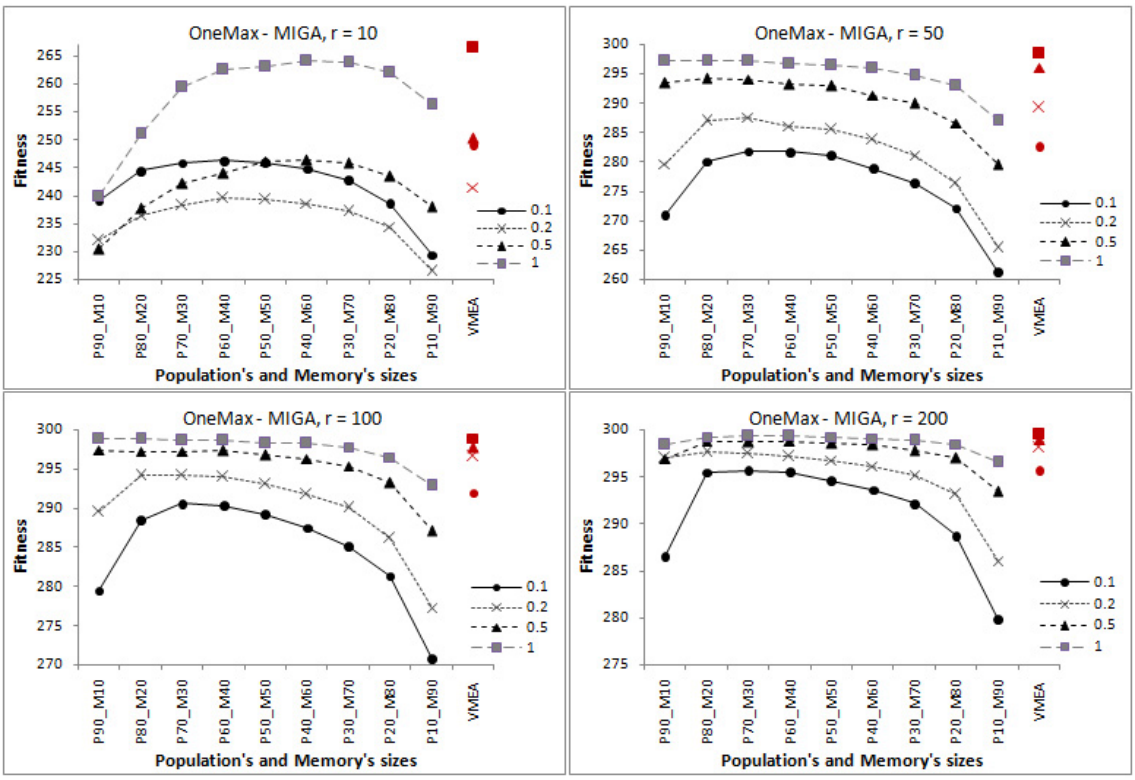

Figure 2. Global results obtained in the dynamic Onemax problem using MIGA with different population and memory sizes

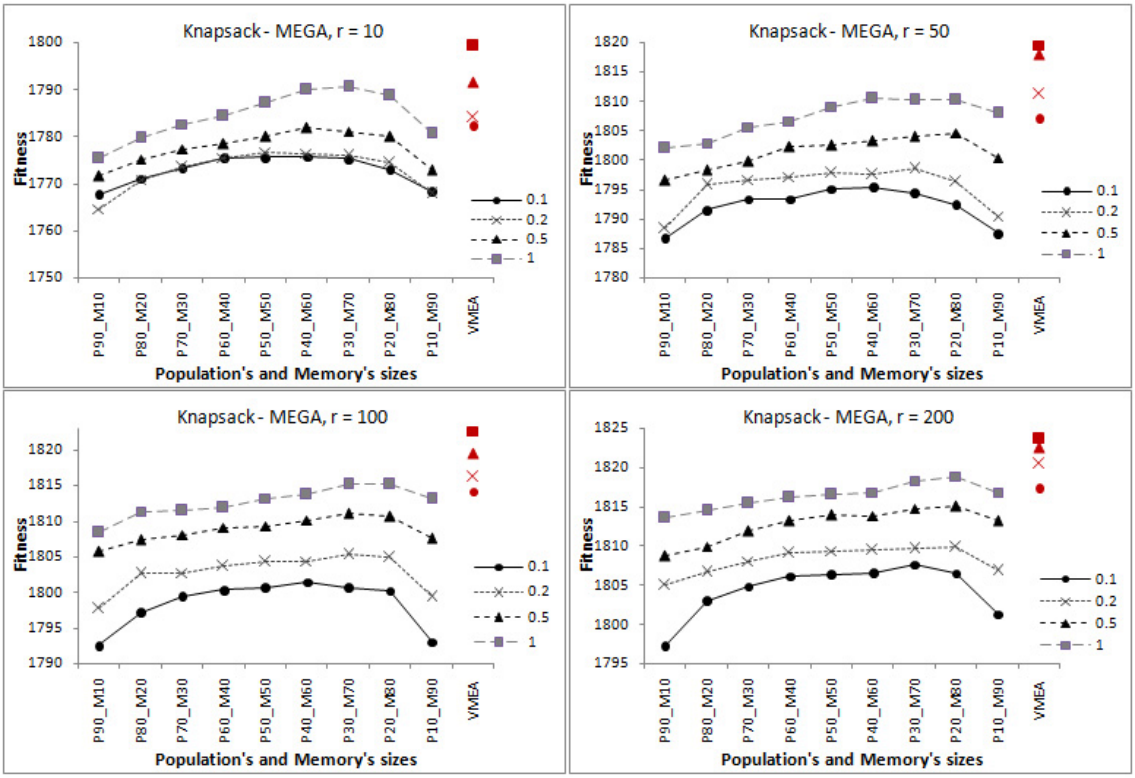

Figure 3. Global results obtained in the dynamic Knapsack problem using MEGA with different population and memory sizes

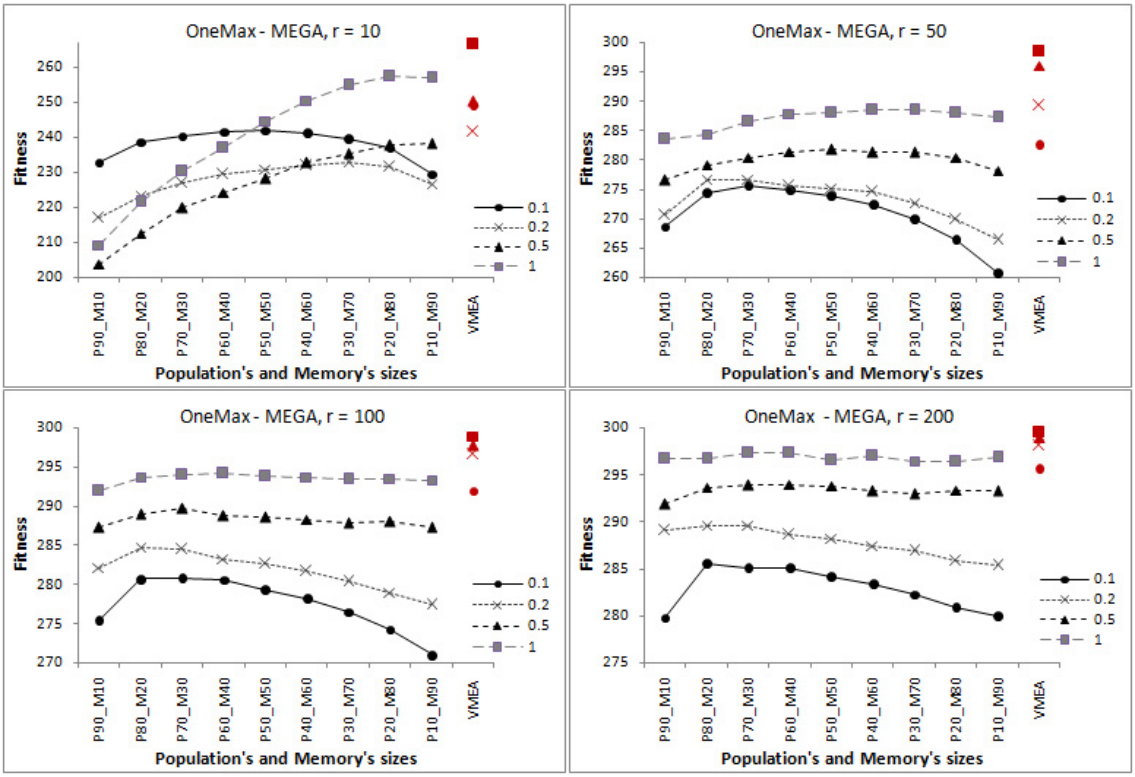

Figure 4. Global results obtained in the dynamic Onemax problem using MEGA with different population and memory sizes

The major statistical results of comparing the different EAs are in Table 2. We used paired one-tailed t-test at a 0.01 level of significance. The notation used in Table 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.

We saw before that choosing different sizes for the population and memory affects the algorithm's performance. Results on Table 2 support that this difference is statistically significant. In general, VMEA performed significant better than MIGA's and MEGA's best results on most dynamic environments. These results validate our expectation of the impact that a bad choice of population and memory sizes can have in EA's performance and also that dynamically sizing approaches should be further investigated.

t-test	*Knapsack*				*Onemax*			
r = 10, ρ ⇒	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
MIGA's best -- MIGA's worst	++	++	++	++	++	++	++	++
VMEA's -- MIGA's best	++	+	++	++	++	+	++	++
MEGA's best -- MEGA's worst	++	++	++	++	++	++	++	++
VMEA's -- MEGA's best	++	++	++	++	++	++	++	++
r = 50, ρ ⇒	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
MIGA's best -- MIGA's worst	++	++	++	++	++	++	++	++
VMEA's -- MIGA's best	+	++	++	++	+	++	++	+
MEGA's best -- MEGA's worst	++	++	++	++	++	++	+	+
VMEA's -- MEGA's best	++	++	++	++	++	++	++	++
r = 100, ρ ⇒	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
MIGA's best -- MIGA's worst	++	++	++	++	++	++	++	++
VMEA's -- MIGA's best	++	++	++	++	++	++	+	+
MEGA's best -- MEGA's worst	++	++	++	++	++	++	++	+
VMEA's -- MEGA's best	++	++	++	++	++	++	++	++
r = 200, ρ ⇒	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
MIGA's best -- MIGA's worst	++	++	++	++	++	++	++	++
VMEA's -- MIGA's best	++	++	++	++	+	++	+	+
MEGA's best -- MEGA's worst	++	++	++	++	++	++	+	+
VMEA's -- MEGA's best	++	++	++	++	++	++	++	++

Table 2. Statistical significance of the obtained results

5.2 Dynamic Behavior of different EAs

In this section we show some examples of how the different algorithms through the entire run. For MIGA and MEGA we show the best and the worst results. VMEA's performance is also included.

For all cases, we will show the results obtained using $\rho = 0.5$. For the other cases, the results were analogous. Figures 5 and 6 refer to the results obtained by MIGA and VMEA. In Figures 7 and 8 is the performance of MEGA and VMEA.

In all cases we can see that the difference in the algorithm's performance using a "good choice" and a "bad choice" for the population and memory sizes is considerable, especially in rapidly changing environments. In MIGA's and MEGA's worst results, it is evident that the evolution is slower and the best performance is only achieved when $r = 200$ and at the end of the process, since more time between changes is given to the algorithms. VMEA's performance is always superior when compared with best performances of the other algorithms.

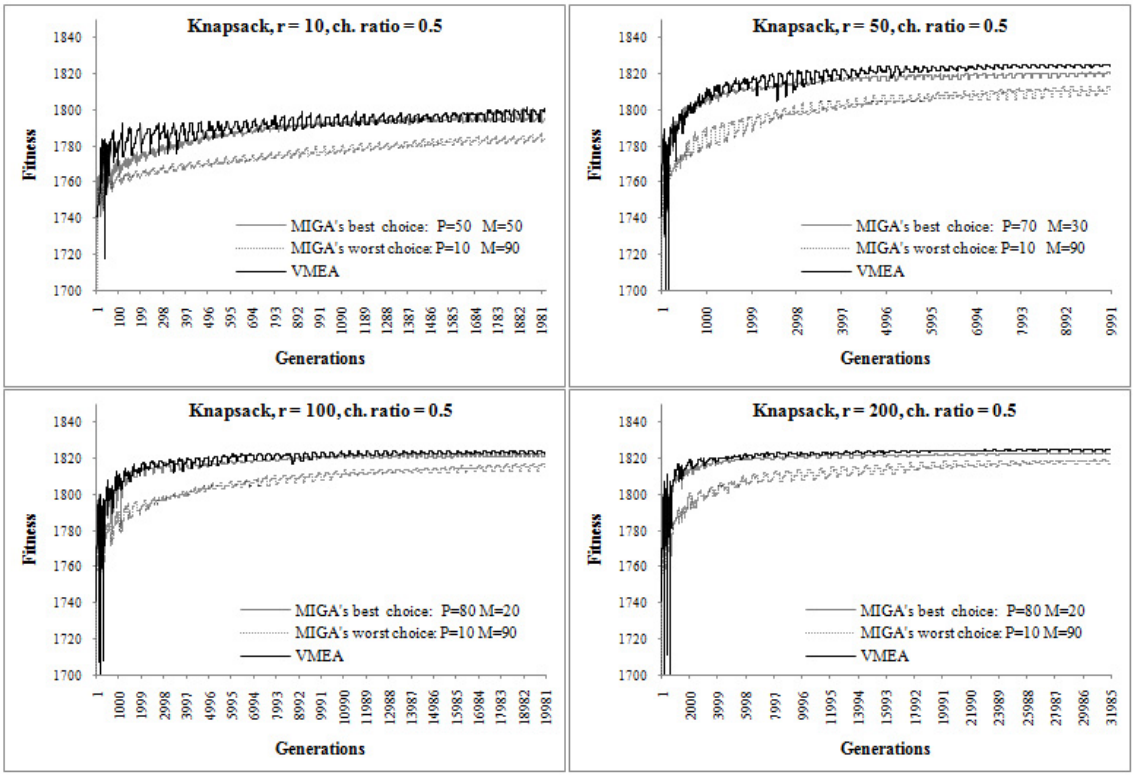

Figure 5. Behavior of MIGA and VMEA in the Knapsack problem, r = 10, r = 50, r = 100 and r = 200, $\rho = 0.5$

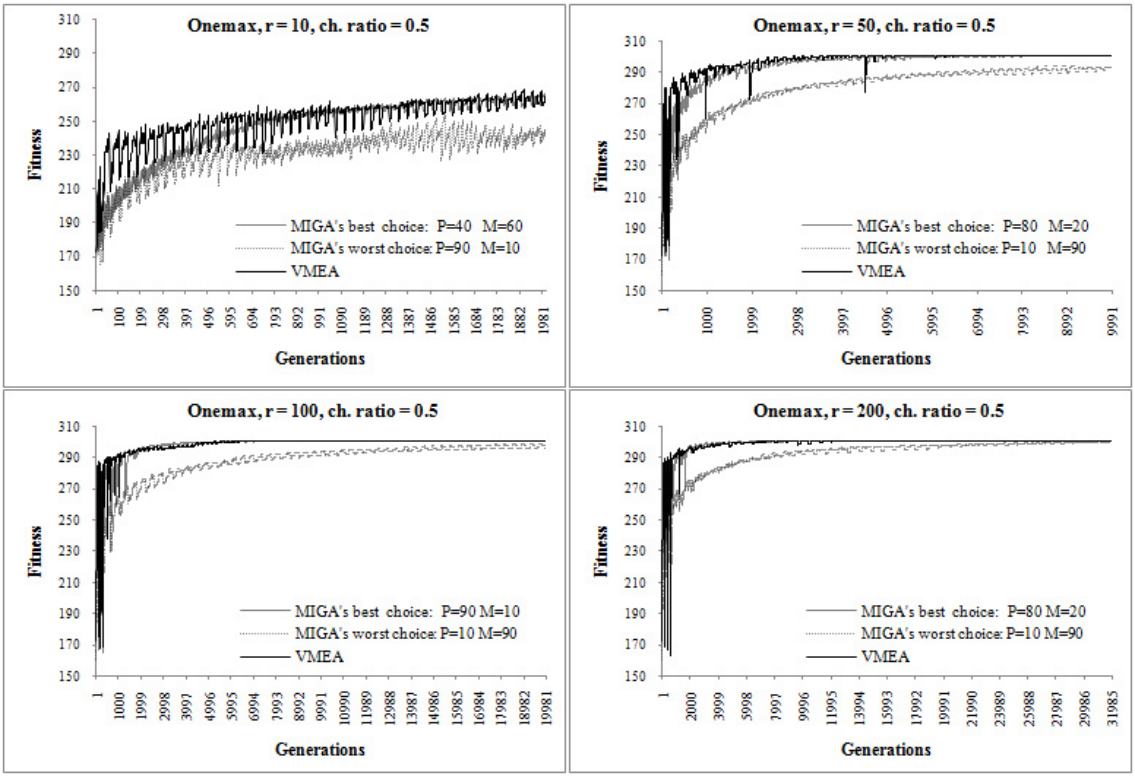

Figure 6. Behavior of MIGA and VMEA in the Onemax problem, r = 10, r = 50, r = 100 and r = 200, $\rho = 0.5$

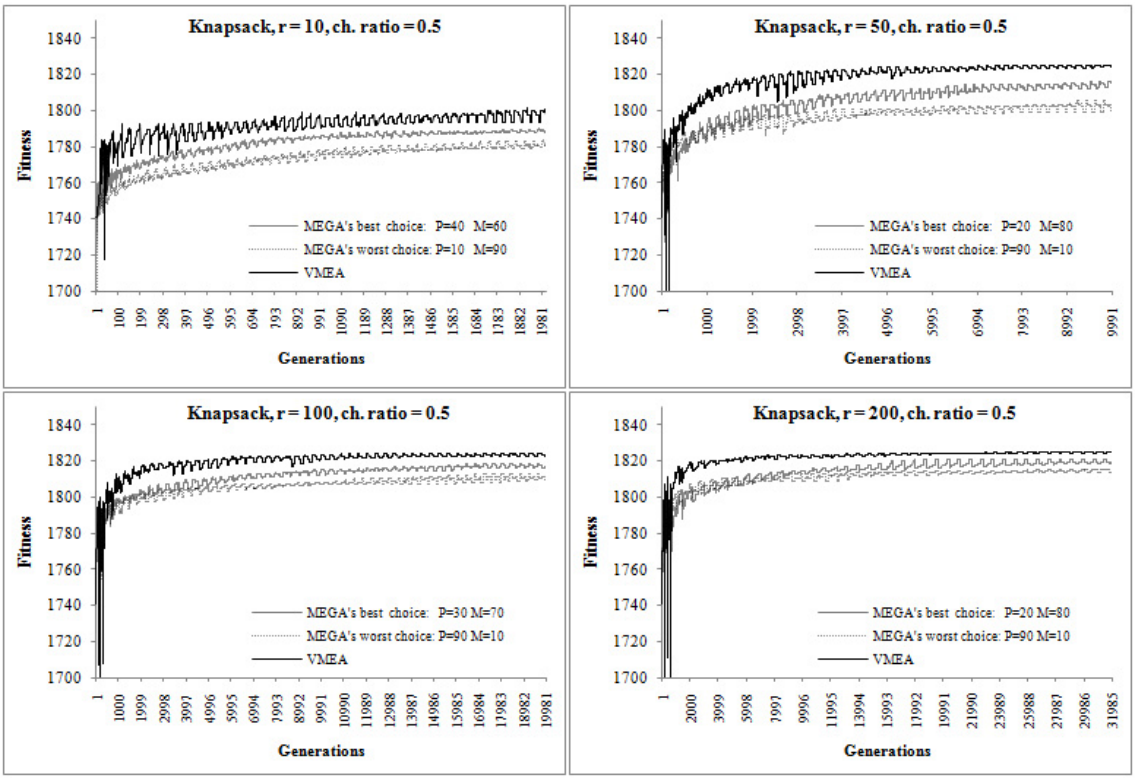

Figure 7. Behavior of MEGA and VMEA in the Knapsack problem, r = 10, r = 50, r = 100 and r = 200, $\rho = 0.5$

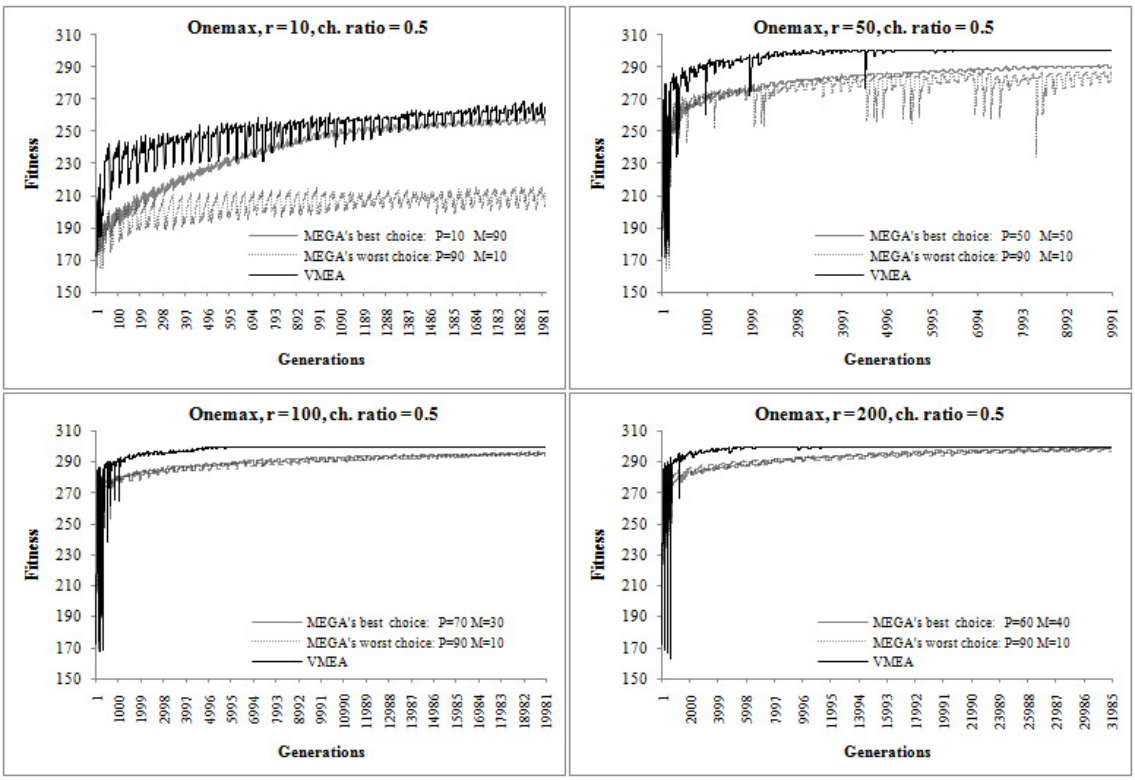

Figure 8. Behavior of MEGA and VMEA in the Onemax problem, r = 10, r = 50, r = 100 and r = 200, $\rho = 0.5$

5.3 Diversity of Population and Memory

To understand the previously shown results, we analyzed the diversity of population and memory using different sizes. We used a standard measure of diversity defined by Equation 8.

$$Div(Pop) = \frac{1}{LP(P-1)} \sum_{i=1}^{P} \sum_{j=1}^{P} HD(p_i, p_j) \qquad (8)$$

where **L** is the length of the chromosome, **P**, the population (or memory) size, $\mathbf{p_i}$, the i^{th} individual in the population and **HD** the Hamming distance.

Figures 9 and 10 show the population's diversity using MIGA (best and worst results) and VMEA for the Knapsack and Onemax problems. As we can see the worst performance of MIGA corresponds to the lower population's diversity and higher memory's diversity. VMEA was the algorithm that maintains higher diversity in the population. In MIGA this observation was generalized to all situations. Figures 11 and 12 show the diversity of population using MEGA and Figures 13 and 14 refer to memory's diversity using MEGA. In this case the results are not clear and there are some cases where the best results were achieved by the population and memory sizes that maintained lower diversity. Thus, it's not straightforward that diversity is the reason for best or worst performances.

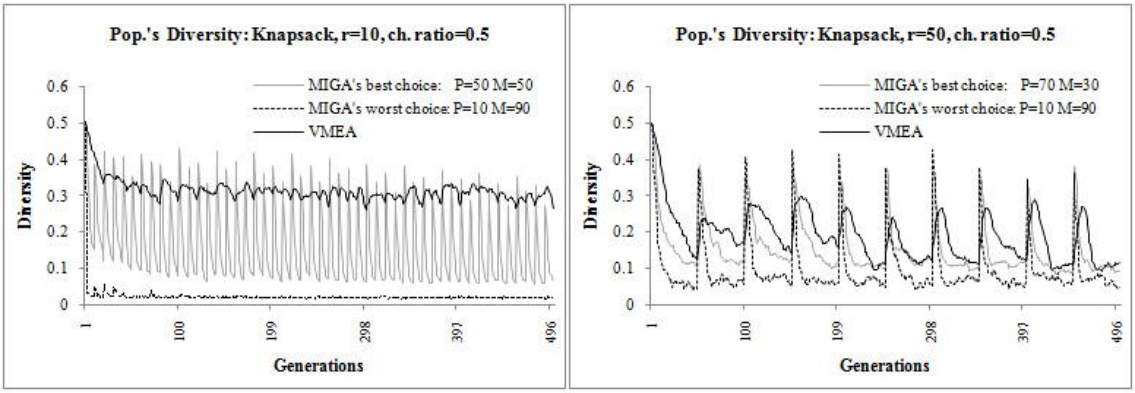

Figure 9. Population's Diversity using MIGA and VMEA in the Knapsack problem, r = 10 and r = 50, ρ = 0.5

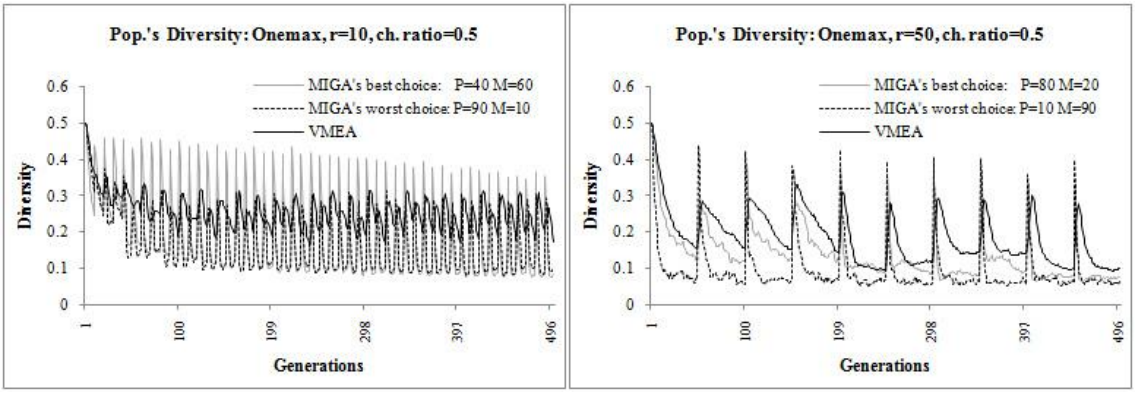

Figure 10. Population's Diversity using MIGA and VMEA in the Onemax problem, r = 10 and r = 50, ρ = 0.5

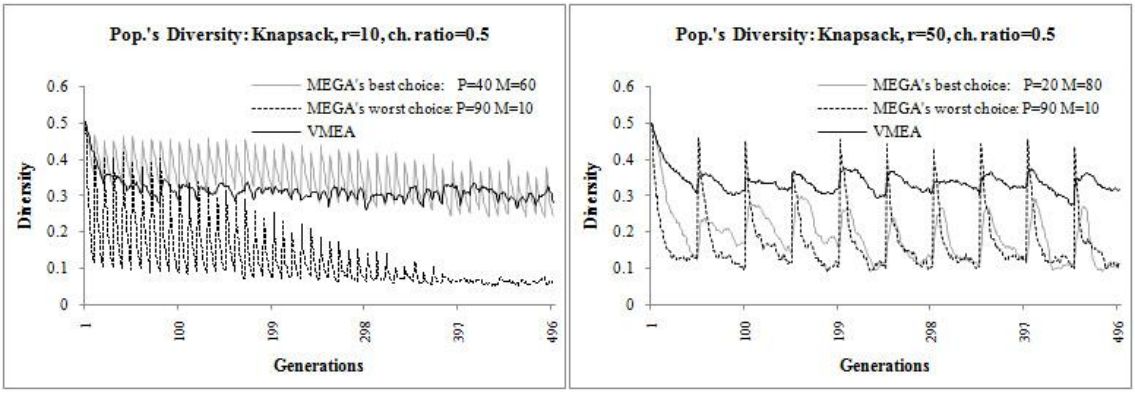

Figure 11. Population's Diversity using MEGA and VMEA in the Knapsack problem, r = 10 and r = 50, $\rho = 0.5$

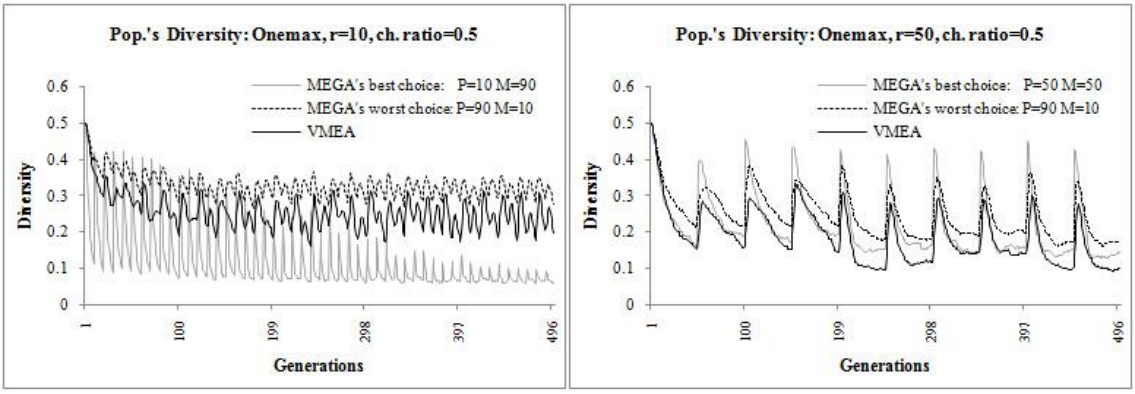

Figure 12. Population's Diversity using MEGA and VMEA in the Onemax problem, r = 10 and r = 50, $\rho = 0.5$

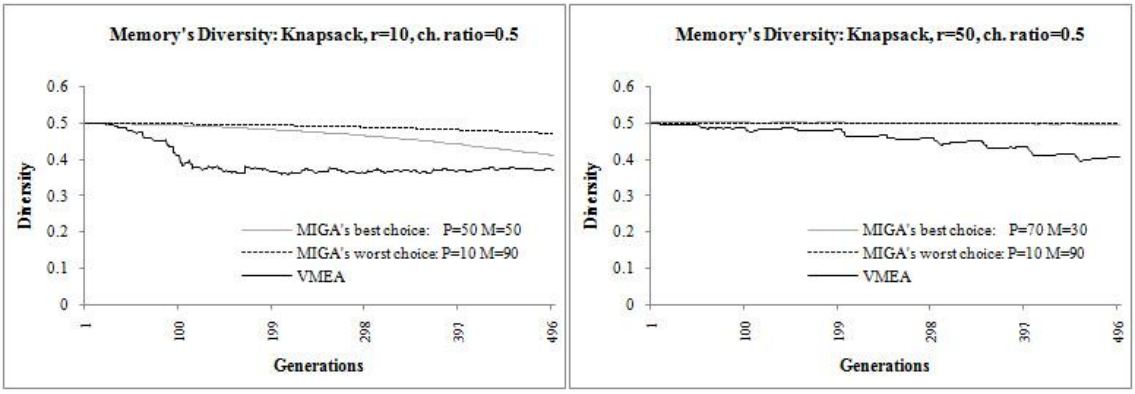

Figure 13. Memory's Diversity using MIGA and VMEA in the Knapsack problem, r = 10 and r = 50, $\rho = 0.5$

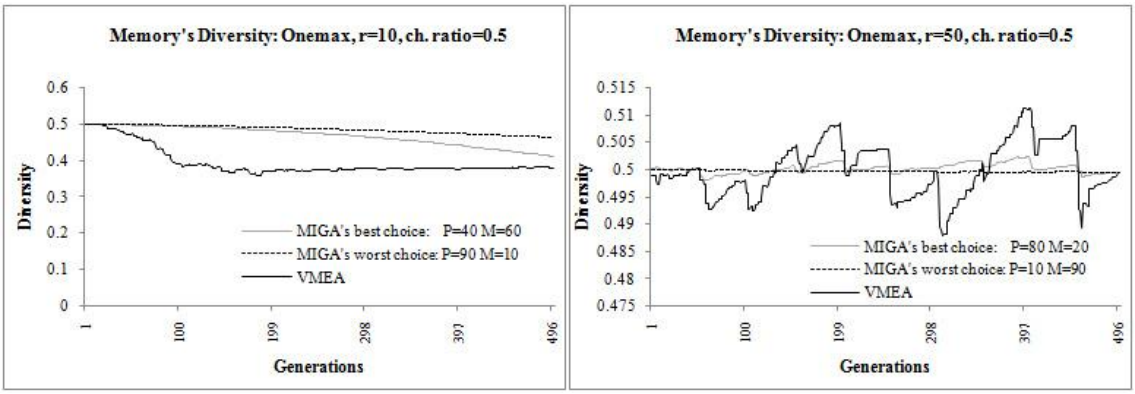

Figure 14. Memory's Diversity using MIGA and VMEA in the Onemax problem, r = 10 and r = 50, $\rho = 0.5$

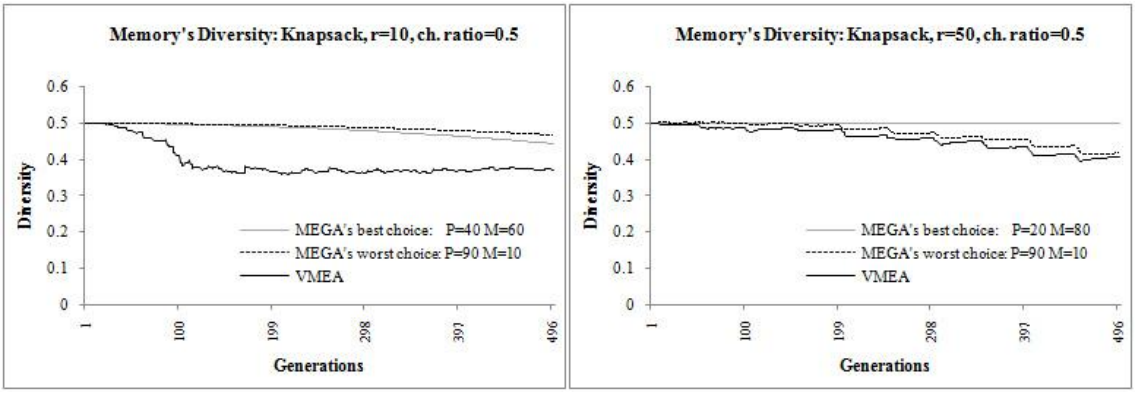

Figure 15. Memory's Diversity using MEGA and VMEA in the Knapsack problem, r = 10 and r = 50, $\rho = 0.5$

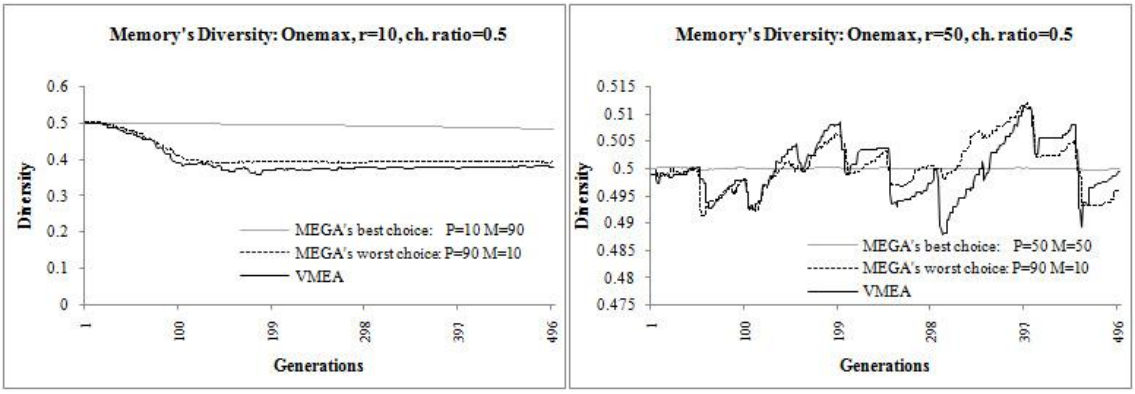

Figure 16. Memory's Diversity using MEGA and VMEA in the Onemax problem, r = 10 and r = 50, $\rho = 0.5$

5.4 Population and Memory sizes in VMEA

In this section we show how population and memory sizes changed during time using VMEA in the studied problems. Figure 17 shows the evolution of these sizes for different change periods in the Knapsack problem, and Figure 18 has the same results for the Onemax problem.

The population and memory sizes are changed according to the established limits to population and memory maximum and minimum values and also to the sum of all individuals. In this case the total number of individuals cannot surpass 100, the minimum and maximum for population and memory were 90 and 10 individuals. When the established limits are reached, the algorithm performs a cleaning process which removes from memory individuals with identical genotype, to store new and different individuals. Based on this implementation, the evolution of population and memory sizes has a typical

behavior. In general memory tends to increase and population to decrease. As the number of generations increase this increase/decrease is slower. At the end of the process, population and memory sizes establishes around similar values: Population of 60-70 individuals and memory of 40-30 individuals, corresponding to the set of values that often lead to best scores using MIGA and MEGA.

The adjustment of population and memory sizes performed by VMEA can be considered "*blind*" since it regards only to the fixed boundaries and the global number of individuals. Nevertheless, superior results were attained. Other sizing mechanisms, incorporating information about the average fitness or the diversity of population, will be tested in near future.

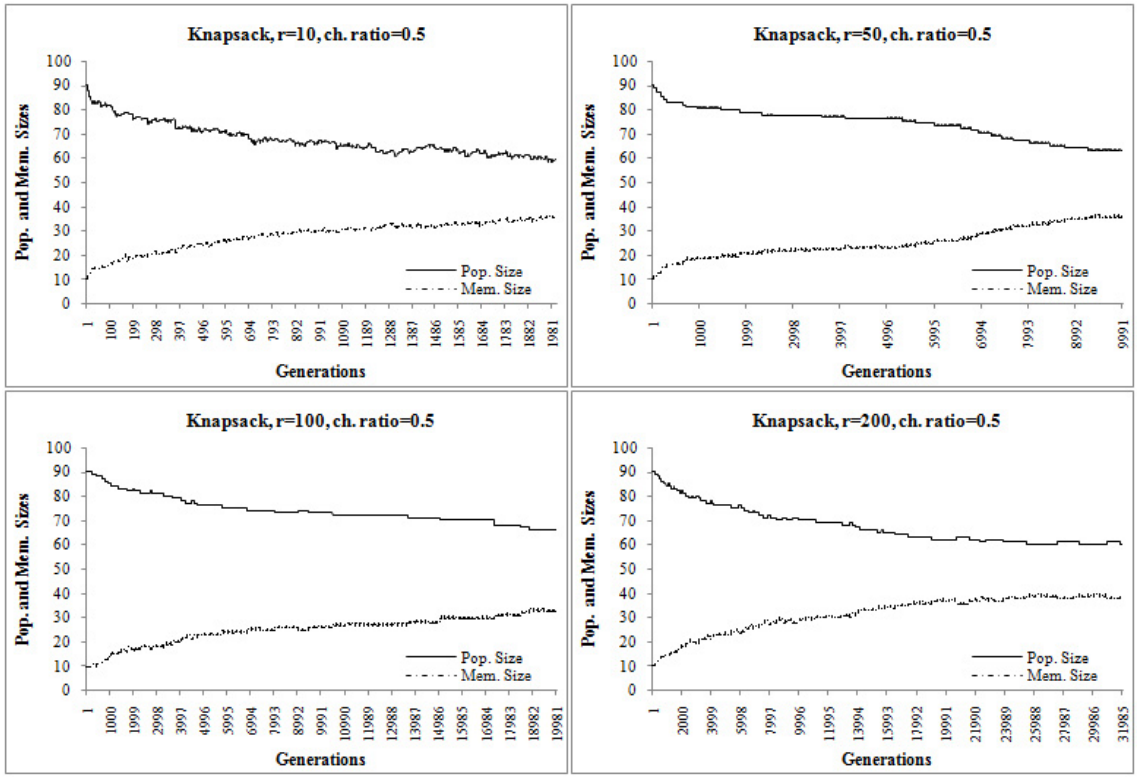

Figure 17. Changes on Population and Memory Sizes using VMEA in Knapsack, r = 10, r = 50, r = 100 and r = 200, $\rho = 0.5$

6 Conclusions and Future Work

Using population and memory of fixed sizes is commonly used in memory-based EAs to deal with dynamic environments. Typically, memory size is set as a small percentage of population size. No valid or extensive justification was found to this generalized use. In this work we tried to understand if different population and memory sizes can have considerable influence in the performance of memory-based EAs dealing with different dynamic environments. Two direct memory EAs were ran with different values for population and memory sizes in two benchmark problems. A third algorithm, using dynamically adjusting population and memory sizes was also tested and compared with previous.

The obtained results show that the traditionally used values for population and memory sizes do not allow the best performance of the implemented EAs. The best performance was achieved using different combinations of population and memory sizes, depending on the type of the environment: severity or speed of change, and also on the used algorithm.

Population's and memory's diversity were analyzed but no consistent results were obtained in order to allow some conclusions about the relation of diversity and performance. It is clear that the tuning of the population and memory sizes has significant influence in the efficacy and convergence of the EAs. The best choice of values depends on the environmental characteristics, the problem to solve or the used algorithm. So, this choice is not linear or easy and trying to tune the population size before running the algorithm is practically impossible, since the combinations are huge and time consuming. A preferred solution is the use of an EA capable of controlling the populations' sizes during the run. In this

work we used a simple EA which dynamically adjusts the population and memory dimensions and the results demonstrate its effectiveness and robustness to all environments and problems tested.

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

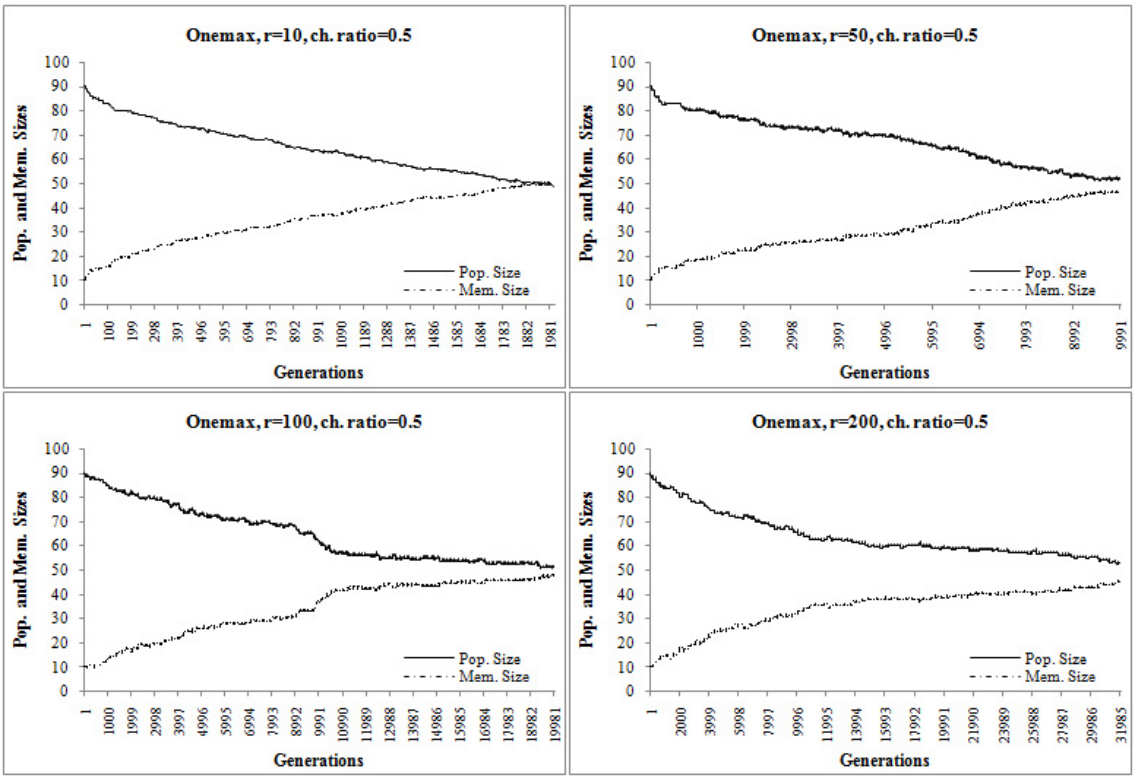

Figure 18. Changes on Population and Memory Sizes using VMEA in Onemax, r = 10, r = 50, r = 100 and r = 200, $\rho = 0.5$

7 Acknowledgments

The work of the first author, 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.

8 References

1. Bäck, T.: Optimal mutation rates in genetic search. In Forrest, S., ed.: Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA 1993), Morgan Kaufmann (1993), pp. 2–8.
2. DeJong, K.A.: An Analisys of the Behaviour of a Class of Genetic Adaptive Systems. University of Michigan. Phd thesis edn. (1975).
3. Grefenstette, J.J.: Optimisation of control parameters for genetic algorithms. IEEE Transactions on Systems, Man and Cybernetics 16(1) (1986), pp. 122–128.
4. Bäck, T.: Self adaptation in genetic algorithms. In Varela, F., Bourgine, P., eds.: Toward a Practice of Autonomous Systems: Proceedings of the First Conference on Artificial Life, MIT Press (1992), pp. 263–271.
5. Davis, L.: Adapting operator probabilities in genetic algorithms. In Schaffer, J., ed.: Proceedings of the Third International Conference on Genetic Algorithms (ICGA 1989), Morgan Kaufmann (1989), pp. 61–69.
6. Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Transactions on Evolutionary Computation 2(3) (1999), pp. 124–141.
7. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer (2003).

8. Gould, J.L., Keeton, W.T.: Biological Science. W. W. Norton & Company (1996).
9. Arabas, J., Michalewicz, Z., Mulawka, J.: Gavaps - a genetic algorithm with varying population size. In: Proceedings of the First IEEE Conference on Evolutionary Computation (CEC 1994), IEEE (1994), pp. 73–78.
10. Bäck, T., Eiben, A.E., van der Vaart, N.A.L.: An empirical study on gas 'without parameters'. In Schoenauer, M., Deb, K., Rudolf, G., Yao, X., Lutton, E., Merelo, J.J., Schwefel, H.P., eds.: Proceedings of Parallel Problem Solving from Nature (PPSN V). Volume 1917 of Lecture Notes on Computer Science, Springer (2000), pp. 315–324.
11. Eiben, A.E., Narchiori, E., Valkó, V.A.: Adapting operator probabilities in genetic algorithms. In Yao, X., et al., eds.: Proceedings of Parallel Problem Solving from Nature (PPSN VIII). Volume 3242 of Lecture Notes in Computer Science, Springer (2004), pp. 41-50.
12. Goldberg, D.E., Deb, K., Clark, J.H.: Genetic algorithms, noise and the sizing of populations. Complex Systems (6) (1992), pp. 333–362.
13. Lobo, F., Lima, C.F.: Revisiting evolutionary algorithms with on-the-°y population adjustment. In Keijzer, M., et al., eds.: Proceedings of the Eighth International Genetic and Evolutionary Computation. Conference (GECCO 2006), ACM Press (2006), pp. 1241–1248.
14. Cobb, H.G.: An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Technical Report TR AIC-90-001, Naval Research Laboratory (1990).
15. Grefenstette, J.J.: Genetic algorithms for changing environments. In Männer, R., Manderick, B., eds.: Proceedings of Parallel Problem Solving from Nature (PPSN 2). (1992), pp. 137–144.
16. Simões, A., Costa, E.: An immune system-based genetic algorithm to deal with dynamic environments: Diversity and memory. In Pearson, D., et al., eds.: Proceedings of the Sixth International Conference on Artificial Neural Networks (ICANNGA 2003), Springer-Verlag (2003), pp. 168–174.
17. Yang, S.: Memory-based immigrants for genetic algorithms in dynamic environments. In Beyer, H.G., ed.: Proceedings of the Seventh International Genetic and Evolutionary Computation Conference (GECCO 2005). Volume 2., ACM Press (2005), pp. 1115–1122.
18. Yang, S.: A comparative study of immune system based genetic algorithms in dynamic environments. In Keijzer, M., et al., eds.: Proceedings of the Eighth International Genetic and Evolutionary Computation. Conference (GECCO 2006), ACM Press (2006), pp. 1377–1384.
19. 19. Yang, S.: Genetic algorithms with elitism-based immigrants for changing optimization problems. In Giacobini, M., et al., eds.: Applications of Evolutionary Computing. Volume 4448 of Lecture Notes in Computer Science., Springer-Verlag (2007), pp. 627–636.
20. Branke, J.: Memory enhanced evolutionary algorithms for changing optimization problems. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999), IEEE Press (1999), pp. 1875–1882.
21. Karaman, A., Uyar, S., Eryigit, G.: The memory indexing evolutionary algorithm for dynamic environments. In: Applications of Evolutionary Computing. Volume 3449 of Lecture Notes in Computer Science., Springer-Verlag (2005), pp. 563–573.
22. Mori, N., Imanishi, S., Kita, H., Nishikawa, Y.: Adaptation to changing environments by means of the memory-based thermodynamical genetic algorithm. In Bäck, T., ed.: Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA 1997), Morgan Kaufmann (1997), pp. 299–306.
23. Simões, A., Costa, E.: Improving memory's usage in evolutionary algorithms for changing environments. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2007), IEEE Press (2007), pp. 276–283.
24. Trojanowski, K., Michalewicz, Z.: Searching for optima in nonstationary environments. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999), IEEE Press (1999), pp. 1843–1850.
25. Yang, S.: Population-based incremental learning with memory scheme for changing environments. In Beyer, H.G., ed.: Proceedings of the Seventh International Genetic and Evolutionary Computation Conference (GECCO 2005). Volume 1. ACM Press (2005), pp. 711–718.
26. Yang, S.: Associative memory scheme for genetic algorithms in dynamic environments. In Rothlauf, F., et al., eds.: Applications of Evolutionary Computing. Volume 3907 of Lecture Notes in Computer Science., Springer-Verlag (2006), pp. 788–799.
27. Branke, J., Kau¼ler, T., Schmidt, C.: A multi-population approach to dynamic optimization problems. In Parmee, I., ed.: Proceedings of Adaptive Computing in Design and Manufacture (ACDM 2000), Spriger-Verlag (2000), pp. 299–308.
28. Wineberg, M., Oppacher, F.: Enhancing the GA's ability to cope with dynamic environments. In Whitley, D., ed.: Proceedings of the Second International Genetic and Evolutionary Computation. Conference (GECCO 2000), Morgan Kaufmann (2000), pp. 3–10.
29. Schönemann, L.: On the influence of population sizes in evolution strategies in dynamic environments. In Cantú-Paz, E., et al., eds.: Proceedings of the Fifth International Genetic and Evolutionary Computation Conference (GECCO 2003), EvoDOPWorskshop. Volume 2723 of Lecture Notes in Computer Science, Springer (2003), pp. 123–127.

30. Schönemann, L.: The impact of population sizes and diversity on the adaptability of evolution strategies in dynamic environments. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2004). Volume 2., IEEE Press (2004), pp. 1270–1277.
31. Simões, A., Costa, E.: Variable-size memory evolutionary algorithm to deal with dynamic environments. In Giacobini, M., et al., eds.: Applications of Evolutionary Computing. Volume 4448 of Lecture Notes in Computer Science., Springer-Verlag (2007), pp. 617–626.
32. Richter, H., Yang, S.: Memory-based on abstraction for dynamic fitness functions. In Giacobini, M., et al., eds.: Applications of Evolutionary Computing. Volume 4974 of Lecture Notes in Computer Science., Springer-Verlag (2008), pp. 597–606.
33. Branke, J.: Evolutionary Optimization in Dynamic Environments. Kluwer Academic Publishers (2002).
34. Goldberg, D.E., Smith, R.E.: Nonstationary function optimization using genetic algorithms with dominance and diploidy. Genetic Algorithms (1987), pp. 59–68.
35. Ng, K.P., Wong, K.C.: A new diploid scheme and dominance change mechanism for nonstationary function optimization. In Eshelman, L.J., ed.: Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA 1995), Morgan Kaufmann (1995), pp. 159–166.
36. Uyar, A.S., Harmanci, A.E.: A new population based adaptive dominance change mechanism for diploid genetic algorithms in dynamic environments. Soft Computing 9(11) (2005), pp. 803–814.
37. Wang, H., Wang, D.: An improved primal-dual genetic algorithm for optimization in dynamic environments. In: Neural Information Processing. Volume 4234 of Lecture Notes in Computer Science., Springer-Verlag (2006), pp. 836–844.
38. Yang, S.: Nonstationary problem optimization using the primal-dual genetic algorithm. In Sarker, R., et al., eds.: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2003). Volume 3., IEEE Press (2003), pp. 2246–2253.
39. Yang, S., Yao, X.: Experimental study on population-based incremental learning algorithms for dynamic optimization problems. Soft Computing 9(11) (2005), pp. 815 – 834.
40. Yang, S.: Explicit memory schemes for evolutionary algorithms in dynamic environments. In Yang, S., Ong, Y.S., Jin, Y., eds.: Evolutionary Computation in Dynamic and Uncertain Environments. Springer-Verlag (2007), pp. 3–28.
41. Simões, A., Costa, E.: Improving memory-based evolutionary algorithms in changing environments. Technical Report TR 2007/04, CISUC (2007).
42. Simões, A., Costa, E.: Variable-size memory evolutionary algorithm to deal with dynamic environments: an empirical study. Technical Report TR 2006/04, CISUC (2006).
43. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag (1999).