Maturation of Individuals in Evolutionary Learning

T. Calenda[1], A. Vitale[2], A. Di Stefano[2], V. Cutello[1], E-G. Talbi[3] and M. Pavone[1]

[1] Department of Mathematics and Computer Science
University of Catania
V.le A. Doria 6, I-95125 Catania, Italy
cutello@dmi.unict.it; mpavone@dmi.unict.it

[2] Department of Electric, Electronics and Computer Science
University of Catania
v.le A. Doria 6, I-95125, Catania

[3] Laboratoire LIFL, Université Lille 1
UMR CNRS 8022, Cite scientifique
Bat. M3, 59655 Villeneuve dAscq cedex, France
el-ghazali.talbi@univ-lille1.fr

Abstract. Although it is well-known that a proper balancing between exploration and exploitation plays a central role on the performances of any evolutionary algorithm, what instead becomes crucial for both is the life time with which any offspring maturate and learn. Setting an appropriate lifespan helps the algorithm in a more efficient search as well as in fruitful exploitation of the learning discovered. Thus, in this research work we present an experimental study conducted on eleven different age assignment types, and performed on a classical genetic algorithm, with the aim to (i) understand which one provides the best performances in term of overall efficiency, and robustness; (ii) produce an efficiency ranking; and, (iii) as the most important goal, verify and prove if the tops, or most, or the whole ranking previously produced on an immune algorithm coincide with that produced for genetic algorithm. From the analysis of the achievements obtained it is possible to assert how the two efficiency rankings are roughly the same, primarily for the top 4 ranks. This also implies that the worst option obtained for the immue algorithm continues to be a bad choice even for the genetic algorithm. The most important outcomes that emerge from this research work are respectively (1) the age assignment to be avoided, from which are obtained bad performances; and (2) a reliable age to be assigned to any offspring for having, with high probability, robust and efficient performances.

1 Introduction

As it is well known in the natural computing field, one of the major successful factors in evolutionary algorithms is the design and development of the exploration and exploitation mechanisms. A good balancing between these two phases is crucial since it strictly affects the efficiency and robustness of evolutionary algorithms performances. While the aim of the exploration mechanism is to search for new solutions in new regions by using the mutation operator, the second mechanism has the purpose to exploit in the best possible way all information gathered using the selection process. Both phases, hence, help the algorithm in discovering, gaining and learning new information, and, subsequently, in exploiting all gained promising regions so to generate better populations.

However, what allows to take advantage of the acquired information is truly given by how long each individual lives and in doing so influencing the evolution and maturation of the population. Besides, this lifetime affects, also, the exploration phase, allowing having a better and deep search process. Thus, the time an individual remains in the population becomes crucial in the performances of any evolutionary algorithm, and it is strictly related to the good balancing between the exploration and exploitation processes. Indeed, letting individuals live for a long time produces a dispersive search, and, then, an unfruitful learning, with the final outcome of increasing the probability to easily get trapped in local optima due to the low diversity that is generated. On the other hand, allowing a short lifetime often does not help to have enough overall learning of the knowledge discovered, and it neither allows a careful search within the solutions space, producing instead high diversity into the population, which, in turn, negatively affects the convergence towards a global optimum.

A first research work on this aspect was conducted in [3], where the authors presented an experimental study whose main aim was to understand the right lifetime of any individual/solution in order to perform

a proper exploration within the search space, as well as a fair exploitation of the gained information. Such experimental analysis was conducted on an immune algorithm, whose core components are the cloning, hypermutation and aging operators.

In the cited research work, eleven different options about the lifetime of each individual were studied (see table 1), with the main goal to answer the three main questions: (i) *"is the lifespan related to the number of offspring generated?"*; (ii) *"is the lifespan related to the population size?"*; and in case of negative answer to the two previous ones, (iii) *"how long must the lifespan of an offspring be to carry out a proper exploration?"*. Once these questions were answered, an efficiency ranking was produced, from which clearly emerged that a too short lifetime is always the worst choice; whilst the best one is to let it evolve for all iterations allowed starting from scratch, i.e. assigning age 0 to each offspring. Thus, following the above described study, in this research work we want to check if the achievements produced on the Immune Algorithm (IA) are still valid, and primarily work, on a genetic algorithm (GA). Of course, what we do not expect to get the same efficiency ranking, but rather we would like to check if the top 4 for IA still appear as the top 4 for GA, even if in different ranking order, and, moreover, if the worst for IA continues to still be the worst for GA. In a nutshell, what we would like to assert with this research work is the existence (if any) of a lifespan, to be assigned to the offspring, that roughly provides robust and efficient performances, especially when high uncertainty on the problem to be solved exists; as well as which is to be avoided for sure. It is important to emphasize that such outcomes are correct and valid on all those problems that show similar landscape topologies, and similar complexities to the problem tackled.

2 The One-Max Problem

To validate and generalize the obtained results, it is crucial to develop an algorithm, which is not tailored to a specific problem, by keeping it unaware of any knowledge about the domain. As it is well-known in literature, to tackle and solve generic and complex combinatorial optimization problems, any evolutionary algorithm must incorporate local search methodologies, used as refinement and improvement of the fitness function, and this means that they have to add knowledge about the features of the problem and application domain. This, consequently, makes the algorithm unsuitable and inapplicable to any other problem, restricting then the validity of the outcomes only on such kind of problem. To overcome this limitation and make the outcomes as general as possible, in this study we tackle the classic *One–Max* (or *One–Counting*) problem [9, 2], as done in [3]. *One–Max* is a well-known toy problem, used to understand the dynamics and searching ability of a generic stochastic algorithm [8]. Although it is not of immediate scientific interest, it represents a really useful tool in order to well understand the main features of the algorithm, for example: what is the best tuning of the parameters for a given algorithm; which search operator is more effective in the corresponding search space; how is the convergence speed, or the convergence reliability of a given algorithm; or what variant of the algorithm works better [1]. It is worth emphasizing that a toy problem gives us a *failure bound*, because a failure occurs in toy problems at least as often as it does in more difficult problems. Formally, given a bit string $s = \{s_1, \cdots, s_\ell\}$ of length ℓ, the *One-Max* problem is simply defined as the task to maximize the number of 1 inside s, i.e.:

$$\text{maximize} \;\; f(s) = \sum_{i=1}^{\ell} s_i, \;\; \text{with} \;\; s_i \in \{0, 1\}. \tag{1}$$

The choice of this simple problem, but enough complex to validate the outcomes, is therefore due mainly to the faithfully reproduction of the experimental study conducted in [3], but also because of its *"blind"* features that guarantee us to can generalize all outcomes produced.

3 The Age Assignments Studied

This research work, as well as the previous one proposed in [3], arise from observing how an algorithm (specifically an IA) can obtain considerably different performances changing only the age to assign to each offspring [7], which clearly proves how the age assignment plays a crucial and central role on the performances of the algorithm in term of success and convergence. In order to reach the goals of this research work, the same eleven age assignment types proposed in [3] have been investigated, and they are reported in table 1. In the table are reported, respectively, types and symbols used for showing and describe the results, and, in the last column, a short description of them.

Table 1. Age assignments studied.

Type	Symbol	Description
0	$[0:0]$	age zero
1	$[0:\tau_B]$	randomly chosen in the range $[0:\tau_B]$
2	$[0:(2/3\ \tau_B)]$	randomly in the range $[0:(2/3\ \tau_B)]$
3	$[0:inherited]$	randomly in the range $[0:inherited]$
4	$[0:(2/3\ inherited)]$	randomly in the range $[0:(2/3\ inherited)]$
5	$inherited$ or $[0:0]$	inherited; but if constructive mutations occur then type 0
6	$inherited$ or $[0:\tau_B]$	inherited; but if constructive mutations occur then type 1
7	$inherited$ or $[0:(2/3\ \tau_B)]$	inherited; but if constructive mutations occur then type 2
8	$inherited$ or $[0:inherited]$	inherited; but if constructive mutations occur then type 3
9	$inherited$ or $[0:(2/3\ inherited)]$	inherited; but if constructive mutations occur then type 4
10	$inherited-1$	same age of parents less one

It is possible subdivide the age assignment types in three different efficacy groups: (1) the fixed ones (`type0` and `type10`); (2) the random ones (from `type1` to `type4`); and (3) the constructive change ones (from `type5` to `type9`). In the first group to each offspring is assigned the same age for everyone; in the second one, instead, to each offspring is assigned a random age, ensuring however evolve at least for a fixed number of generations, except for the `type1` that in the worst case will assign age τ_B. Besides, `type3` and `type4` differ from the previous two as each offspring will have the same age (in worst case) or less than its parent (labelled as *"inherited"*). Finally, all age types included into the last group produce higher diversity in the population than the others, and encourage those offspring that appears to be promising. Indeed, to each clone is assigned the same age of the parent at first, which generates high turnover degree in the population, but if after the genetic (crossover and/or mutation) operators, the fitness of the offspring is improved, then its age is updated in order to have more evolutionary time.

4 The Genetic Algorithm

In this research work we have developed a classical Genetic Algorithm (GA), one of the most well known evolutionary algorithms, which take inspiration from genetics and natural selection. The genetic algorithms represent an efficient and robust algorithmic class in search and optimization, thanks to their ability in fruitfully exploration of search space, and efficiently exploiting of the promising regions. This class of algorithms is based on three main evolutionary operators, such as the (i) *recombination*, where each generated offspring inherits some characteristics from the two parents; (ii) *mutation*, which introduces diversification in the offspring with respect to the parents; and (iii) *selection mechanism*, through which the individuals for mating pool are selected. The first two operators represent the exploration phase of new search points, whilst the last one helps the algorithm to exploit information learned in order to generate improved progenies (exploitation phase). Note, however, that in the GAs the mutation operator plays a secondary role than the other two.

For achieving the set out goals in this research work, we have appropriately adapted the developed genetic algorithm so that a given age is assigned to each generated individual at each time step. In this way, it is determined the lifespan of each offspring inside the population, whose evolution starts from such assigned age (see section 3), which is increased by one at each generation, until reaching the maximum age allowed (τ_B, an user-defined parameter). Further, the aging operator as designed in [3, 7], was developed and included into this proposed GA [5, 6].

The algorithm starts with the creation of the initial population ($P^{(t=0)}$) of size pop_size, by generating random solutions, i.e. bit strings of length ℓ, using uniform distribution, and thus the next step is the evaluation of the fitness of each individual, using the function *Compute_Fitness(*$P^{(t)}$*)*. Whereupon, begins the evolution of the algorithm whose considered termination criterion is the reaching of the maximum allowed number of fitness function evaluations (T_{max}). The algorithm terminates in advance the execution if the

global optimal solution is found. Note that age zero is assigned to each newly created individual, regardless of the age assignment chosen (see the age types in section 3). A summary of the developed Genetic Algorithm is presented in the pseudocode shown in Algorithm 1.

Algorithm 1 Pseudo code of GA

Genetic Algorithm ($pop\_size, p_c, p_m, \tau_B, Elitism$)
$t \leftarrow 0$
$FFE \leftarrow 0$
$P^{(t)} \leftarrow$ Create_Initial_Population(pop_size);
Compute_Fitness($P^{(t)}$)
$FFE \leftarrow FFE + pop\_size$
while ($FFE < T_{max}$) **do**
  Increase_Age($P^{(t)}$);
  $P^{(parents)} \leftarrow$ Roulette_Wheel_Selection($P^{(t)}, pop\_size$)
  $P_c^{(kids)} \leftarrow$ Crossover($P^{(parents)}, p_c$)
  $P_m^{(kids)} \leftarrow$ Mutation($P_c^{(kids)}, p_m$)
  Compute_Fitness($P^{(t)}$)
  $FFE \leftarrow FFE + pop\_size$
  $({}^a P^{(t)}, {}^a P_m^{(kids)}) \leftarrow$ Aging($P^{(t)}, P_m^{(kids)}, \tau_B, Elitism$);
  $P^{(t+1)} \leftarrow (\mu + \lambda)$–Selection($^a P^{(t)}, {}^a P_m^{(kids)}$);
  $t \leftarrow t + 1$
end while

Inside to the iterative loop the first step is to increase the age of each individual by one, becoming older than a generation (function *Increase_Age(P^t)*). Hence, the selection of the individuals for the mating is applied using the classical *Roulette-Wheel-Selection* model [4, 10], which basically select the individuals with a probability proportionally to their fitness: the higher the fitness, the higher the likely is that they will be selected. At this step a new population $P^{(parents)}$ of size pop_size is created that contains all selected individuals, from whose mating will be produced the offspring. In particular, from a pair of individuals are generated a pair of offspring.

Afterwards, the parents population is then undergo to the recombination phase, with a probability p_c, which generates the population of the new offspring ($P_c^{(kids)}$). In this work we have developed the classic *Uniform Crossover*, through which each element (gene) of the offspring is randomly selected by both parents; i.e. the parents will contribute equally to generating their own descendants. Thanks to this kind of recombination operator each offspring will have 50% genes from the first parent, and the other 50% from the second one. Once $P_c^{(kids)}$ is produced, each chromosome is mutated with a p_m probability. The mutation used in our study is the well known *bit-flip mutation*, which - if applied - randomly select a gene s_i ($\forall\ i = 1, ..., \ell$) in the chromosome s, and inverts its value (from 0 to 1, or from 1 to 0). The mutated chromosomes produce a new population, labelled $P_m^{(kids)}$ (see Algorithm 1), containing the final offspring generated. When an offspring is created, to it is assigned an age that affects its lifespan inside the population. Starting from this age each chromosome will evolve until to reach a maximum age allowed (τ_B, a user-defined parameter), after which it will be removed from the population by the aging operator. Age assignment, and the aging operator have the main purpose to keep high the diversity into the population in order to avoid premature convergences and then reduce the probability to get stuck in local optimum. Therefore, choosing the age to be given plays a crucial role in the performances of the algorithm, since the evolution and maturation of the solutions depend strictly on this. Note that in general, the crossover and mutation operators do not affect the age of any new individual, except for the $5 - 9$ options of the age assignment types (see table 1, section 3), where the assigned age is updated only if its fitness value is improved. This happens and it is computed in the function *Compute_Fitness($P^{(t)}$)*.

As described above, in our GA was included the aging operator in order to achieve the determined objectives, and whose main task is to help the algorithm in jumping out from the local optima, producing high diversity into the population and avoiding, consequently, premature convergences. It simply eliminates the old chromosomes from the two populations $P^{(t)}$ and $P_m^{(kids)}$. Every individual is allowed to mature for a fixed number of generations: as soon as it reaches age $(\tau_B + 1)$, it is removed from the population of belonging regardless of the fitness value, included the best solution found so far. The parameter τ_B indicates

the maximum number of generations allowed to any chromosome to stay into the population. An exception, however, is allowed only for the best solution found so far, that is the global best solution found is always kept into the population, even if it is older than $\tau_B + 1$. Such exception is called *Elitism Aging operator*. This variant helps the algorithm keep track of the most promising region - which would otherwise be lost - and whose exploitation instead might be useful in solving some specific kinds of problems. The boolean variable $Elitism$ (Algorithm 1) controls the activation of the variant elitism aging operator.

Unlike to classic GA, where usually the offspring replace the parents for the next generation, in this work the new population $P^{(t+1)}$ is created by using the $(\mu+\lambda)$-*Selection operator*, which selects the best pop_size survivors to the aging step from the two populations ${}^aP^{(t)}$ and ${}^aP_m^{(kids)}$. This operator, simply, selects the best pop_size chromosomes from the offspring set – created by the crossover and mutation operators – and the old parent chromosomes guaranteeing monotonicity in the evolution dynamics. Nevertheless, due to the aging operator, it could happen that the number of survivor chromosomes (pop_size^1) is less than the input population size (pop_size). If so, the selection operator randomly generates ($pop\_size - pop\_size^1$) new individuals. This step is called *Birth phase*.

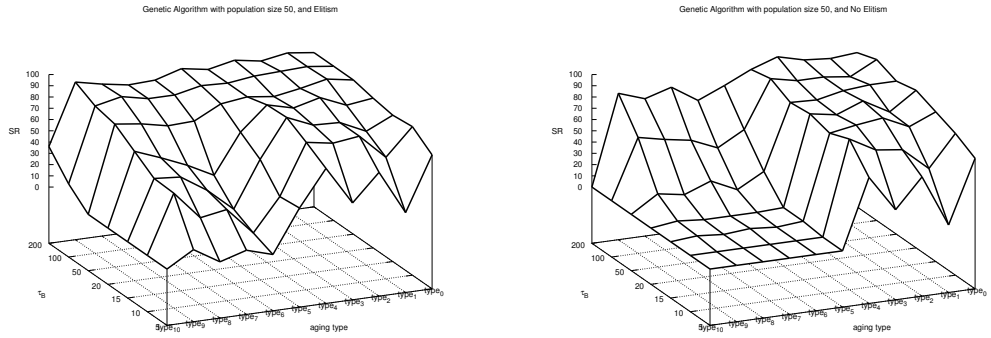
5 Experimental Results

In this section we present all outcomes obtained in our experimental study in order to (i) understand which between the age assignment types studied is more robust and efficient in the overall; (ii) produce an efficiency ranking between them; and, most important, (iii) to check if the best 4 of the efficiency ranking obtained for IA [3] still appear in the top 4 for GA, although in different rank order. Besides, it becomes also interesting to observe if the worst assignment - or the two worst - for IA continue to be so also for GA. With the outcomes of this research work, we want to give a rough indication on the needed lifetime to a solution to have a proper balancing between exploration and exploitation in order to maximize the evolutionary learning, and which one instead to not consider. Further, having developed an algorithm not tailored to a specific problem, and performed our study in an unaware way about any domain knowledge, the outcomes obtained will be even more effective when high uncertainty exists to be managed. However, the existence of one or more age assignments in common between IA and GA allow us also to provide a reliable lifespan, which, with high probability, leads to efficient and robust performances on similar features problem, and on evolutionary algorithms, in general.

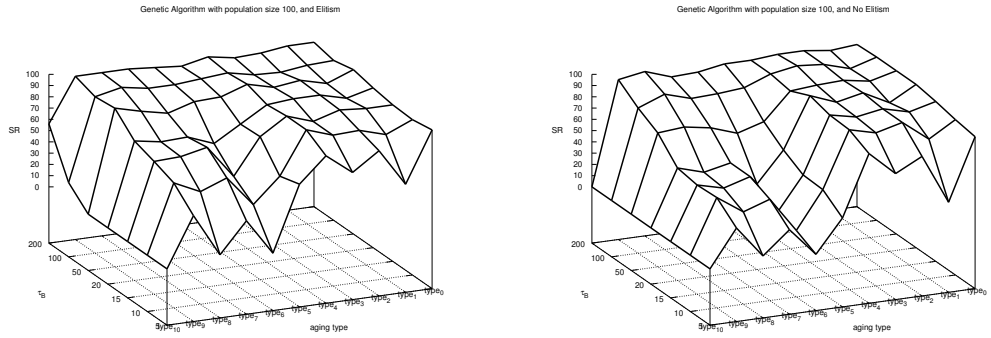
In order to reach our goals, we have then used the same experimental protocol proposed in [3], except for the string length, which was instead fixed to $\ell = 2000$. This setting is due to the difficulty of the algorithm in solving the problem for higher values. Therefore, all age assignment options in table 1, have been studied varying $pop\_size = \{50, 100\}$, $\tau_B = \{5, 10, 15, 20, 50, 100, 200\}$, and setting $Tmax = 10^5$ as termination criteria (i.e. the maximum number of fitness function evaluations). Each experiment was computed on 100 independent runs. Both variants of GA have been performed: *elitism* and *no_elitism*. Regarding GA parameters, after several preliminary experiments (not included in this paper), the probability to apply the crossover (p_c) and mutation (p_m) operators have been set respectively to $p_c = 1$ and $p_m = 0.4$, for all experiments presented.

In figure 1 are showed the efficiency surfaces produced by varying τ_B and age assignment types, and evaluate with respect their success rate (SR), that is how many time the optimal solution is found in 100 runs. Both GA variants are showed for $pop\_size = 50$ (first row), and $pop\_size = 100$ (last row): the *elitism* version in the left plots, and the *no_elitism* in the right ones. Inspecting these surfaces, appear clearly in all plots how the last age assignment option (`type10`) shows the worst performances at any values of τ_B and pop_size, unable almost always in finding the optimal solution (except for the elitism variant with high τ_B values). Those who instead show better performances in the overall are `type0` and `type4`, which exhibit more robust and efficient performances, regardless of the parameters used. It is important to highlight how, for low τ_B values, the age assignment options from `type5` to `type9` (constructive change group; see table 1) produce the lower regions of SR, and this is due to their characteristic to produce high diversity into the population, which means high turnover degree and, therefore, a greater use of the *Birth* operator. This appears more prominent in the *no_elitism* variant, and especially for low pop_size values, where diversity becomes more pronounced. In plot, it is also possible to see how any age assignment becomes irrelevant on the performances for high τ_B value ($\tau_B = \{100, 200\}$), whose values are very close to have an infinite life.

The same results are also, and better, presented in tables 2 and 3. In each row are showed the success rate (SR) obtained, and AES (line below), i.e. the average number of fitness function evaluations to reach the optimal solution. Of course, AES is not null when $SR \neq 0$, i.e. if the optimal solution was found at



(a)



(b)

Fig. 1. Efficiency surfaces produced for *elitism* (left plots) and *no_elitism* (right plots) variants for $pop\_size = 50$ in plots (a), and $pop\_size = 100$ in plots (b).

least once. In particular, the results presented in table 2 have been obtained with $pop\_size = 50$, whilst the ones in table 3 with $pop\_size = 100$. Analysing both tables, we have clearly the confirmation that `type10` is unable to reach the optimal solution, except for the *elitism* version with setting high values of τ_B ($\tau_B = \{100, 200\}$). Further, it is interesting to note that increasing the population size (pop_size) helps GA to increase the SR (as we expect) but it doesn't affect nor alter in any way, the overall influence of the age assignments on the performances of GA, producing approximately the same efficiency rankings. This statement answers, and confirms, as also claimed for IA, that the right lifespan for which each individual must evolve is not related to the population size considered.

Inspecting table 2, considering the *elitism* variant, it is possible to see how the algorithm finds the optimal solution, at least once, with all age assignments considered, except for `type10` ($\forall\ \tau_B < 100$); `type6` and `type8` when the τ_B value is low; unlike of the *no_elitism* version where GA struggles to reach the optimal solution, and this is what we expect. However, one of our goal is to understand which age assignment type provides more robust and efficient performances. Therefore, from this point of view, it is possible to assert that `type0` and `type4` seems to be more robust than the others, guaranteeing then more reliability, whilst `type10` and `type6` continue to be the worst ones. The same statement can also be made for the *no_elitism* version. From table 3 is possible to see how the SR increases on all experiments, allowing the achievement of the optimal solution, where GA failed in the previous table. Also on these experiments `type0` and `type4` appear to be more robust and efficient than the others. Comparing these two age assignments types, it is possible to assert that, in the overall, `type0` shows more reliability than `type4`, since its use allows GA to obtain, approximately, the same, and high, success rate both in the *elitism*, and *no_elitism* variants (`type0`: 82.14 vs. 83 for $pop\_size = 50$, and 92 vs. 92 for $pop\_size = 100$ – `type4`: 83.86 vs. 79.86 for $pop\_size = 50$, and 91.57 vs. 91.71 for $pop\_size = 100$).

Table 2. GA on *One–Max* problem with $\ell = 2000$. The results have been obtained by setting: $pop\_size = 50$, $p_c = 1$, $p_m = 0.4$, with and without Elitism.

type	$\tau_B = 5$	$\tau_B = 10$	$\tau_B = 15$	$\tau_B = 20$	$\tau_B = 50$	$\tau_B = 100$	$\tau_B = 200$
				Elitism			
0	69% 90741.44	82% 86409.38	80% 87103.44	83% 86652.88)	87% 84676.01	87% 82797.44	87% 83204.83
1	21% 98724.99	58% 94472.81	69% 92170.46	80% 88594.79	91% 85552.17	90% 82780.77	90% 83890.19
2	60% 94039.49	80% 87698.82	80% 86159.5	90% 84839.10	86% 84058.78	89% 84007.95	86% 84330.07
3	36% 97341.97	77% 89434.23	81% 86796.83	84% 86317.53	82% 85648.48	88% 84421.64	90% 83416.20
4	70% 91853.51	81% 87440.62	91% 85747.82	90% 83773.17	85% 84213.71	85% 83853.50	85% 84004.85
5	38% 96661.74	36% 96847.64	57% 93269.04	69% 90052.98	82% 87376.17	86% 84609.75	89% 84330.95
6	0%	7% 99763.66	16% 98820.08	34% 97435.21	70% 89703.56	81% 86601.79	82% 85642.79
7	7% 99723.64	31% 97398.44	36% 96937.86	46% 94309.49	69% 89642.67	81% 86399.53	81% 85410.66
8	0%	27% 98454.84	51% 94153.06	56% 92745.16	74% 87680.47	86% 87118.12	85% 83791.32
9	14% 99273.65	52% 93683.66	53% 94559.11	65% 91631.61	77% 87691.02	81% 85953.13	90% 83402.34
10	0%	0%	0%	0%	0%	15% 98903.89	36% 96232.28
				No Elitism			
0	66% 90771.76	76% 87042.24	83% 86277.61	90% 85208.42	86% 85111.56	93% 82111.92	87% 83868.15
1	10% 99486.34	48% 95815.82	73% 92056.89	74% 90457.19	86% 85435.83	81% 86840.81	85% 83652.09
2	56% 94334.12	68% 88144.89	84% 87116.65	87% 84443.68	84% 86296.26	84% 85577.26	87% 84776.40
3	30% 97689.88	71% 89510.53	81% 86619.09	78% 86543.17	79% 85487.73	84% 85086.12	93% 83041.88
4	62% 92538.22	84% 86222.68	77% 87402.21	82% 86827.35	80% 85629.22	89% 84230.84	85% 83664.17
5	0%	0%	2% 99666.75	2% 99557.23	22% 95892.60	47% 90973.89	74% 84776.14
6	0%	0%	0%	1% 99921.99	9% 98130.29	34% 92421.01	64% 87918.47
7	0%	0%	2% 99601.45	1% 99673.91	20% 95497.91	44% 89819.80	79% 83756.32
8	0%	0%	0%	1% 99911.66	11% 97763.70	48% 91077.56	72% 86381.38
9	0%	0%	2% 99607.87	3% 99240.23	15% 96522.20	53% 88524.80	80% 85093.18
10	0%	0%	0%	0%	0%	0%	0%

In figure 2 is showed the histograms produced by the average of the success rates for each age assignment type. The left plot shows the average SR produced using the *elitism* variant, whilst the right plot those produced by the *no_elitism* version. Such results have been produced averaging on all τ_B values. Thanks to these plots becomes easy to produce the efficiency ranking for each variant, which seem to be the same in the overall, except for the first position: for the *elitism* variant appears `type4` on the first rank; whilst for the *no_elitism* version the top one becomes `type0`. From the third position onwards, the two efficiency ranking seems to be the same, respectively: 3) `type2`, 4) `type3`, 5) `type1`, 6) `type5`, 7) `type9`, 8) `type7`, 9) `type8`, 10) `type6`, and 11) `type10`.

Since it is important to produce an efficiency ranking regardless of variants used, in figure 3 we show the average success rate ($\widehat{SR}$) produced by both the variants on all experiments performed. From this plot, emerge `type0` as the best ($\widehat{SR} = 87.27\%$), followed by `type4` ($\widehat{SR} = 86.75\%$), `type2` ($\widehat{SR} = 84.89\%$), `type3` ($\widehat{SR} = 80.61$), and `type1` ($\widehat{SR} = 73.93\%$). From these overall results appear clear as the last type, the `type10`, is always the worst with an overall average success rate considerably low with respect the others ($\widehat{SR} = 4.39\%$), and never reaching the optimal solutions in *no_elitist* variant (Fig. 2, plot b). Just to note that the penultimate in the efficiency ranking between the age assignment types (*type6*) produces an overall average success rate of $\widehat{SR} = 38.75\%$.

Once generated the efficiency ranking for GA, it is possible to compare the outcomes obtained with the ones produced by IA in [3], in order to reach the third and most important goal of this work. Analysing the comparisons, then, it is possible to clearly assert that the top 4 types produced by IA are still in the top 4 of

Table 3. GA on *One–Max* problem with $\ell = 2000$. The results have been obtained by setting: $pop\_size = 100, p_c = 1, p_m = 0.4$, with and without Elitism.

type	$\tau_B = 5$	$\tau_B = 10$	$\tau_B = 15$	$\tau_B = 20$	$\tau_B = 50$	$\tau_B = 100$	$\tau_B = 200$
Elitism							
0	91% 82312.79	88% 79513.60	89% 80284.09	92% 78748.99	96% 76946.22	92% 77181.40	96% 76713.31
1	46% 96226.19	78% 89019.29	89% 84632.08	86% 84135.64	87% 81507.88	91% 78715.07	96% 78471.94
2	79% 86728.32	85% 80317.57	91% 80431.61	92% 78505.86	93% 78550.34	88% 78540.41	93% 77438.99
3	63% 91211.17	84% 84436.42	84% 82546.80	92% 80466.22	90% 78691.13	89 78144.62	92% 78627.42
4	81% 86788.92	92% 81076.87	92% 79704.66	90% 80032.76	96% 77980.19	94% 77136.83	96% 75580.17
5	59% 93334.05	54% 92368.72	77% 88048.81	76% 86315.99	88% 81170.38	92% 76130.42	91% 77768.30
6	1% 99989.08	32% 97195.04	45% 95906.89	62% 91813.69	87% 83105.46	90% 80479.18	93% 78291.70
7	33% 97217.75	59% 90119.24	74% 89718.72	71% 87589.66	80% 83268.17	90% 78213.40	95% 78284.81
8	6% 99720.49	50% 94409.41	69% 88542.49	70% 89541.02	85% 82873.96	94% 78983.65	95% 78189.25
9	44% 95440.97	61% 91484.43	68% 87646.74	74% 87221.28	91% 78151.71	89% 79294.21	95% 77613.35
10	0%	0%	0%	0%	0%	16% 98535.69	56% 93547.39
No Elitism							
0	85% 83264.41	88% 79521.29	92% 78683.06	95% 78733.66	95% 77829.72	95% 77960.86	94 79148.00
1	30% 98132.36	74% 90100.46	86% 84718.40	82% 84605.64	89% 80768.79	88% 80283.13	92% 79894.21
2	74% 88613.47	92% 80472.08	92% 79932.29	94% 78718.43	93% 77271.95	95% 78857.49	95% 76172.57
3	62% 94039.55	86% 84023.41	88% 81425.63	89% 79838.93	92% 77831.94	97% 78117.67	95% 78376.93
4	83% 84655.22	92% 80158.74	93% 79879.89	98% 77569.49	90% 78153.02	93% 77655.90	93% 77784.34
5	29% 95268.10	42% 92692.33	43% 91418.28	53% 88893.36	66% 84100.89	84% 79270.36	93% 78672.59
6	0%	9% 99385.86	12% 98633.38	26% 96286.96	59% 87265.87	82% 80200.76	89% 79989.41
7	25% 96783.49	40% 92715.84	48% 91968.87	48% 90444.85	67% 83522.93	81% 79741.93	88% 79447.61
8	5% 99780.98	32% 95006.62	41% 92756.55	43% 91618.77	71% 82310.38	88% 78390.08	96% 77734.65
9	33% 95163.12	48% 91905.64	47% 91211.35	50% 88393.57	69% 84288.04	79% 80440.52	92% 77513.32
10	0%	0%	0%	0%	0%	0%	0%

the efficiency ranking produced by GA, respecting exactly the order of the first two positions. Further, the worst age types on IA continue to be the worst even on GA, and in particular in the last two positions appear `type 6` and `type 10` respectively. Their bad performances are due to the high diversity they produce, not allowing a relevant lifetime to perform a good exploration. Finally, this research work, having found a common efficiency ranking between IA and GA, provides a reliable individuals maturation time in order to optimize the evolutionary learning and yield robust and efficient performances on those problems that show similar landscape topologies, and similar complexities to the problem considered.

6 Conclusion

In this paper we show how the age assignment, i.e. how many generations an offspring must remain into the population, plays a crucial role on the performances of any evolutionary algorithm, since it is strictly related to a correct balancing between the exploration and exploitation mechanisms. In this research work we present an experimental study focused on understanding the right maturation time of each solution in order to optimize the evolutionary learning, which means to perform a careful search process, and take advantage of the information discovered as best as possible. Since this experimental study is based on a previous one, conducted on an immune inspired algorithm, the main goal of this paper is to check and prove that the achievements previously obtained continue to be valid, and work, on a genetic algorithm. In this way, we can assert the existence of a reliable lifespan able to provide, with high probability, robust and efficient performances for any population-based algorithm, especially in uncertainty environments.

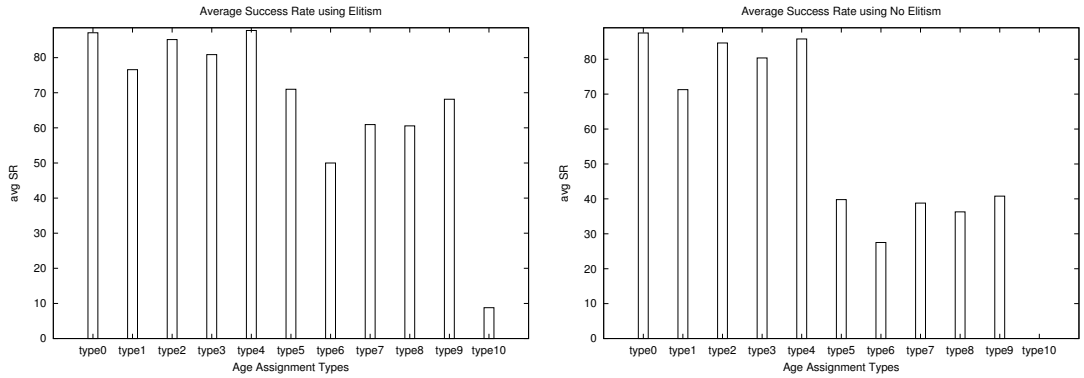


Fig. 2. Average Success Rate over all performed trials for the *elitism* (left plot) and *no_elitism* (right plot) variants.

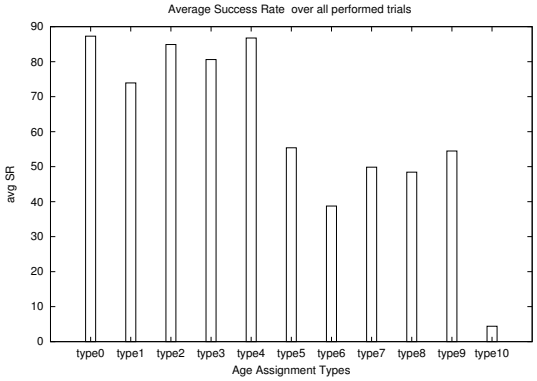


Fig. 3. Average Success Rate over all performed trials.

A classical genetic algorithm has been developed, based on the genetic operators: *roulette-wheel-selection*; *uniform crossover* and *flip mutation*. The GA was properly adapted in order to reach the set out achievement, adding an age assignment for each chromosome and introducing the aging operator, the use of which helps the algorithm to escape from local optima. Eleven different age assignment types have been considered in our study, each of which affects on the algorithm performances in different way: *static ones*; *random ones*; and *constructive change ones*. Analysing the achievements obtained in this work, and comparing them with the ones previously obtained by IA, it is possible to assert as the previous top 4 continue to be in the best 4 positions of the new efficiency ranking, keeping exactly the same order in the first two positions. In the overall, it is possible also to say that the two efficiency rankings are approximately the same, except 2-3 positions that alternate each other. This then imply that the worst, or the two worst between the age assignment types in IA continue to be the worst also in GA.

Having found a common efficiency ranking between IA and GA, points out the existence of a reliable lifetime to be assigned to each individual - for any population-based algorithm - that with high probability guarantees efficient and robust performance, especially when many information on the problem are not known a priori. Of course these statements are valid on all those problems that show similar complexities and similar landscape topologies to the problem tackled. In light of this, as future work, we want to test and perform the same experimental study on a mathematical model, that is the NK-Model, which is able to produce *"tunable rugged"* fitness landscapes. In this way we can test the achievements produced on different roughness level of the landscape.

References

1. V. Cutello, A. G. De Michele, M. Pavone: "*Escaping Local Optima via Parallelization and Migration*", VI International Workshop on Nature Inspired Cooperative Strategies for Optimization (NICSO), Studies in Computational Intelligence, vol. 512, pp. 141–152, 2013.

2. V. Cutello, G. Narzisi, G. Nicosia, M. Pavone: "*Clonal Selection Algorithms: A Comparative Case Study using Effective Mutation Potentials*", 4th International Conference on Artificial Immune Systems (ICARIS), LNCS 3627, pp. 13–28, 2005.
3. A. Di Stefano, A. Vitale, V. Cutello, M. Pavone: "*Document How long should offspring lifespan be in order to obtain a proper exploration?*", 2016 IEEE Symposium Series on Computational Intelligence (SSCI), INSPEC number 16670548 2016, pp. 1–8, 2016.
4. D. Goldberg: "*Genetic Algorithms in Search, Optimization, and Machine Learning*", Addison-Wesley publishing company, 1989.
5. N. Kubota, T. Fukuda: "*Genetic Algorithms with Age Structure*", Soft Computing, vol. 1, pp. 155–161, 1997.
6. A. Ghosh, S. Tsutsui, H. Tanaka: "*Individual Aging in Genetic Algorithms*", Australian and New Zealand Conference on Intelligent Information Systems (ANZIIS), INSPEC number 5534628, pp. 276–279, 1996.
7. M. Pavone, G. Narzisi, G. Nicosia: "*Clonal Selection - An Immunological Algorithm for Global Optimization over Continuous Spaces*", Journal of Global Optimization, vol. 53, no. 4, pp. 769–808, 2012.
8. A. Prugel-Bennett, A. Rogers: "*Modelling Genetic Algorithm Dynamics*", Theoretical Aspects of Evolutionary Computing, pp. 59-85, 2001.
9. J. D. Schaffer, L. J. Eshelman: "*On crossover as an evolutionary viable strategy*", 4th International Conference on Genetic Algorithms, pp. 61–68, 1991.
10. E. G. Talbi: "*Metaheuristics: from Design to Implementation*", John Wiley & Sons, 2009.