Evolutionary Computing - Task 1: Specialist Agent
Group 65

01/10/2021

Bruna Aguiar Guedes
Student Number: 2698211

Enrico Calleris
Student Number: 2692023

Caterina Fregonese
Student Number: 2737116

Romy Vos
Student Number: 2595704

1 INTRODUCTION

The world as we know it is the result of countless transformations that, over time, gave rise to the current observable reality - starting with the transformation of dead matter at the beginning of the Universe, which eventually lead to the vibrant spurring of life on planet Earth. These processes, which still continue to perpetuate over time, allow life to spread and evolve, thus creating more and more complex organisms. For this reason, it can be claimed that such mechanisms have proved their efficiency. Among these processes, evolution is the driving force that pulls life together and allows its unfolding, despite the constant threats posed by an unstable environment. The problem-solving nature of evolution has inspired most recent technological advances in AI, especially in one of its subfields: Evolutionary Computing (EC), whose aim is that of designing and test Evolutionary Algorithms (EAs) for global optimization. The two main criteria that these algorithms should satisfy before being labeled as "successful" are efficiency and correctness: while their efficiency is mostly inspired by biological processes as explained by Darwinian principles, their correctness results from a series of fine-tuned parameters which, after being carefully combined by the researcher in a computational setting, should yield good enough results to a given problem.

In this paper we will focus on Genetic Algorithms (GAs), a branch of EAs, that mimics biological evolution by solving optimization problems based on the process of natural selection. GAs include some parameters that should be adjusted, so as to get reliable results [1]. By choosing a representation of the problem addressed, an initial population, a method of selection, variation operators (i.e., mutation and recombination), the probabilities of crossover and mutation, and the insertion method you can create variants of a certain GA [1]. In this paper, we will take a closer look at these algorithms and at their working mechanisms by comparing two instances of the same GA (EA1 and EA2) with different mutation operators, justifying their relevance as genetic operators in the context of a specific task. In particular, the task at hand was that of developing, applying ad comparing EAs for the task of video game playing using EvoMan, a python framework inspired on the action-platformer game Mega Man II. We decided to implement and test two instances of the same GA (their only difference being a different number of step sizes used for the mutation operator) with the aim of investigating the effect of using different self-adaptation mutation heuristics on the GA's performance against different enemies. Using the simulation mode "individual evolution" of the video game framework we designed and trained a specialist agent capable of winning against an opponent. The two different self-adaptation mutation heuristics we used to test our player and its efficiency in beating different enemies were uncorrelated one step size mutation for EA1 and uncorrelated $\boldsymbol{n}$ step sizes mutation for EA2. Given that the step size affects convergence and that adopting a good mutation operator is good strategy for overcoming the problem of premature convergence [6], we have decided to adopt this strategy to investigate the EAs' behaviour in relation to the number of step sizes. In the following section we will introduce the parameters that we used to run the experiment; we will then focus on the mutation operators, explaining and exploring their working mechanisms more in detail, in an attempt to answer our research question.

2 METHODS

We explain here in detail the methods used in the framework for testing optimization algorithms with artificial agent controllers in EvoMan.

General Settings

In general, the parameters we used to test our player against a certain enemy are part of a problem-solving strategy called "individual evolution", where you have an AI player, controlled by a user-defined algorithm, and a static enemy, which tries to fight the player by performing static attacks based on fixed rules. By applying this idea to the framework we were working with, we implemented two EA methods that use the simulation mode "individual evolution" to train a specialist agent against one single objective (i.e., specialist agent against one enemy). The enemies used to test these algorithms were enemy 4 (Heatman), enemy 6 (Crashman) and enemy 7 (Bubbleman). By fixing the model's parameters, we then evolved a specialist agent and made independent experiments to test his performance against each enemy. The level of difficulty of the game for each run was set to two.

Evolutionary Algorithm

The two instances of the GA (EA1 and EA2) we used to test a specialist agent against an enemy consist of a population of neural networks with a single hidden layer of 10 nodes, with sigmoid activation function for the hidden and output layer. The input vector consists of 20 sensors which, according to how the values are modified, yield different output results.

Genotype. The genome of each individual in a population consists of 265 (= (20 + 1) • 10 + (10 + 1) • 5) weights: these are continuous values that can be thought of as alleles in a chromosome.

Population Size and Number of Generations. To find well-performing individuals we take as our starting point a population of 100 individuals with random initialization, which populates the initial population with completely random solutions. Once the sensors' parameter are adjusted the population can evolve until it reaches a termination point which, in our case, is set at 20 generations.
The initial population is then analyzed and evaluated to find the next generation's parents. To do so, we used a ranking selection method, an exploitative selection technique which sorts the population according to fitness value and ranks them. Each chromosome is then allocated a selection probability with respect to its rank. Individuals are selected as per their selection probability based on the formula:

$$p = \frac{2-s}{n} + \frac{(2 \cdot i) \cdot (s-1)}{n \cdot (n-1)}, \tag{1}$$

where s is equal to 1.5, a value we chose as to prevent getting stuck in local optima, so that all ranks have a chance to be picked (even if that chance might be small). The rank is picked based on probability.

Fitness Function and Individual Gain. The performance of each individual was evaluated using the following fitness function:

$$f = 0.9 \cdot (100 - e) + 0.1 \cdot p - logt, \tag{2}$$

where p refers to the player's energy level after the end of the fight, e to enemy's energy level, and t to the number of timesteps needed to reach the end of the fight. The function aims at maximizing the energy of the player by the end of the level, while minimizing the energy of the enemy. The last term penalizes fights that take too long to be finished. We then evaluated the best individuals to save for each run based on the individual gain:

$$IG = p - e \tag{3}$$

where IG < 0 indicates a loss, while IG > 0 indicates a player win.

Mutation Operator. Although the parent selection method for EA1 and EA2 did not vary, we applied different mutation operators; most of our work was in fact directed at modifying and testing this parameter. It can be claimed that selecting the most appropriate type of mutation is one of the most important stages of GAs because of its impact on the exploration of the search space. For this reason, adopting a good mutation operator is good strategy for overcoming the problem of premature convergence [6]. Non-uniform mutation was applied to both EA1 and EA2, but with a different number of step sizes: uncorrelated mutation with one step size for EA1 and uncorrelated mutation with n step sizes for EA2. The idea of non-uniform mutation stems from the concept of self-adaptation, where the algorithm controls the setting of these parameters itself embedding them into an individual's genome and evolving them [7]. This concept represents a solution to the problem of how to adapt the step-sizes, a theory that has been successfully demonstrated in many domains, not only for real-valued, but also for binary and integer search spaces [2]. The essential feature is that the step sizes are also included in the chromosomes and they themselves undergo variation and selection [5]. We decided to apply uncorrelated mutation with one step size to test EA1. In this case, the same distribution is used to mutate all x_i, therefore we only have one universal strategy parameter σ embedded in the individual, which is mutated each time step by multiplying it by a term e^{Γ}, with Γ a random variable drawn each time from a normal distribution with mean 0 and standard deviation τ.
When testing EA2 we applied a mutation operator with n step sizes which treats each of $\boldsymbol{n}$ dimensions differently. Each basic chromosome is therefore extended with n step sizes, one for each dimension which provides the flexibility to use different mutation strategies in different directions [5].

Crossover Operator. As for the other operator used to combine genetic information and to pass it from the parents to the offspring, the crossover, we decided to use discrete recombination for both EA1 and EA2.
We opted for a fitness based survival selection method, where the children tend to replace the least fit individuals in the population.

Survivor Selection Method. We used (μ , λ) as selection operator (where μ indicates the size of the parent population, while λ is the size of the offspring), which combines the population of parents with their offspring and sorts them all based on fitness (from high to low). The genome with the highest fitness is then selected and put in the new population. Following this mechanisms, genomes are picked until the desired population size is reached.
This type of survivor selection is an "exploitation" method and is solely based on fitness, with no randomness involved, which could lead to getting stuck in local optima. For this reason, we added "exploration" to this mechanism, so that if the best genome (the one with the highest fitness) does not improve for 10 rounds, the 20 percent worst genomes of the population are selected, removed from the population and replaced them with random genomes.

Experimental Setup

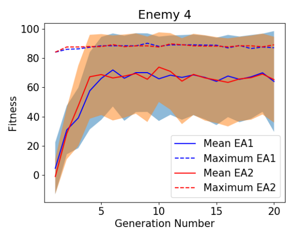
We ran EA1 and EA2 for each of the enemies (4, 6, and 7), for a total of 10 runs per EA and enemy (with population size = 100 and number of generations = 20), saving results (maximum, mean, and std for each generation) and best individuals for each run based on Individual Gain. After that, we ran 5 times the best individuals of each run for each EA and enemy, saving the obtained individual based on IGs. For both training and best individuals run we used the random initialization of the player position.

Plotting. We plotted line plots to compare maximum and mean of EA1 and EA2 for each enemy. The data for each plot were taken as the average over the 10 runs of all generations. The standard deviation was added as a shaded region. Boxplots were created to compare the performances of the best individual's runs for the two EAs for each enemy. The data to plot were obtained by averaging the 5 IGs obtained by each individual.

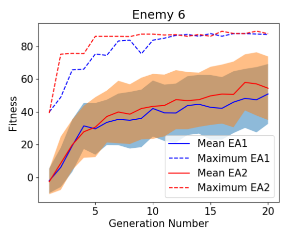
3 RESULTS AND DISCUSSION

After implementing the methods previously described, we analyzed and plotted the results, which we are now going to present. Line plots from the 10 runs of the two EAs for each enemy are presented in Figure 1. As it can be observed from the three different graphs, there appears to be very little variations between the performances of the two different EA instances. Enemy 4 (Figure 1a) and enemy 7 (Figure 1c) present a very similar pattern of maximum and mean values, with comparable standard deviations. Enemy 6 (Figure 1b) has a little more spread in the maximum for the first generations, but then the two EA instances converge in later generations. It can also be observed that the maximum fitness achieved for all the three enemies is roughly the same (around 90), showing that both algorithms are capable of producing high-performing individuals. Enemy 4 and enemy 7 present a similar mean fitness (both around 70), while enemy 6 has an overall lower mean level of fitness, barely reaching 60. The two mean lines of enemy 6 also look more distant, although not significantly. Enemy 4 shows the most interesting behaviour, since the maximum fitness achieved is already very high from the first generations and remains almost constant throughout the generations. Overall, the two algorithms' instances seem to increase their fitness relatively rapidly, reaching already a satisfactory maximum level before generation 10.

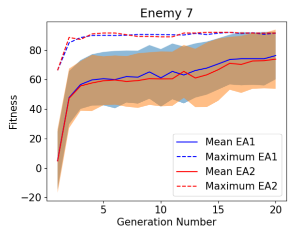
The results of the best individuals' runs are presented in Figure 2. The best individuals were selected based on the highest individual gain achieved in each run, which was also the metrics used. The first observation that can be made is that all algorithms win against all enemies (all means of the boxplots are greater than 0). Enemy 4 (Figure 2a) and enemy 7 (Figure 2c) are the two best performing enemies for the two EAs, with mean values well above the threshold of 0. Enemy 6 (Figure 2b) appears to be tougher to beat, with means still positive, but with greater portions of the boxplots under the 0 threshold.



(a)

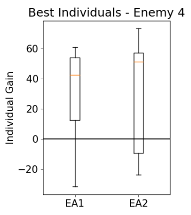


(b)

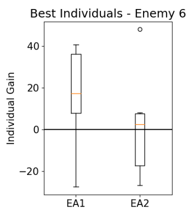


(c)

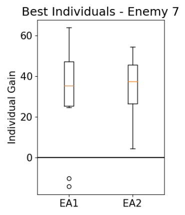
Figure 1: Mean and Maximum Fitness for the 2 EAs for (a) Enemy 4, (b) Enemy 6, (c) Enemy 7, with standard deviation for the mean as a shaded region.



(a)



(b)



(c)

Figure 2: Boxplots for best individuals run for (a) Enemy 4, (b) Enemy 6, (c) Enemy 7, with straight line representing a gain of 0 (threshold between winning and losing).

Finally, statistical tests for the differences between the two EA instances for each enemy are shown in Table 1. None of the samples was found to adhere to normality, so a Kolmogorov-Smirnov test was used to assess significant differences between the two EAs. The tests for enemy 4 and enemy 7 resulted to be non-significant ($p > .05$), while the p-value for enemy 6 sits right at the threshold ($p = \alpha = .05$). This last enemy would require additional inspection to determine the presence of a statistical difference between the performance of the two instances of the EA, using a bigger sample size and more runs.

By comparing our results for best individual gain with those presented in a pioneer paper [3] we noticed that the values we obtained are slightly better or similar to those obtained by [3], with mean value well above 0, meaning that, in our case, enemies 4, 6 and 7 are beatable. The value that differs the most is that of enemy 6, which in [3] is not beatable (mean value = 0), while it can be classified as beatable according to our results.

Table 1: Statistical Comparison for Best Individuals of EA1 vs. EA2 for different enemies. * = *p > .05. KS = Kolmogorov-Smirnov Test.*

Enemy Number	Statistical Test	D-Statistic	p
4	KS	0.3	.79*
6	KS	0.6	.05
7	KS	0.2	.99*

4 CONCLUSION

The goal of this paper was to investigate whether changing the number of step sizes of the mutation operator in two instances of the same GA (in our case EA1, one step size mutation, and EA2, n step sizes mutation) in the context of EvoMan (a video game framework), would impact the GA's overall behaviour, thus resulting in better or worse outputs values for different mutation operators.
Overall, the mutation operators with one and n step sizes do not have much of an impact on the overall behaviour of our GAs, as the tests for enemies 4 and 7 resulted to be non-significant ($p > .05$). Interestingly, enemy 6 sits right at the threshold ($p = \alpha = .05$). It could be interesting to further investigate its behaviour by using a bigger population sample and by running the experiment for more than 20 generations. By doing so, we hope to observe a more significant statistical difference between the two instances of the EA, which would confirm the hypothesis that the step size affects convergence and that adopting a good mutation operator is good strategy for overcoming the problem of premature convergence in GAs [6].
After all, according to Charles Darwin, the father of evolution, the individuals most suited to their environment survive and, given enough time, the species will gradually evolve [4]. Perhaps the same concept could hold for EAs, which might also need enough time to evolve and to show improvements from an individual to species level. Hopefully future research could help shed light on such a fascinating scenario.

CONTRIBUTIONS

Bruna Aguiar Guedes. I have written the code for having the 10 runs for each enemy automatized, as well as writing the two heuristics on the mutation operator. I have implemented statistical tests for the boxplots, in addition to running EA 1 and EA2 for enemies 3 (not used in the report) and 4.

Enrico Calleris. I implemented the saving of best individuals based on individual gain, as well as the code for running them against each enemy ans saving the results of each run. I wrote the code for plotting and saving lineplots and boxplots, in addition to running EA1 and EA2 for enemy 7.

Caterina Fregonese. I wrote the report.

Romy Vos. I implemented the code of the parent selection and survivor selection, as well as running and testing several enemies.

REFERENCES

[1] Otman Abdoun, Jaafar Abouchabaka, and Chakir Tajani. 2012. Analyzing the performance of mutation operators to solve the travelling salesman problem. *arXiv preprint arXiv:1203.3099* (2012).

[2] Thomas Bäck. 2001. Introduction to the special issue: Self-adaptation.

[3] Karine da Silva Miras de Araujo and Fabricio Olivetti de Franca. 2016. Evolving a generalized strategy for an action-platformer video game framework. In *2016 IEEE Congress on Evolutionary Computation (CEC)* (Vancouver, BC, Canada). IEEE, 1303–1310. https://doi.org/10.1109/CEC.2016.7743938

[4] Charles Darwin. 1909. *The origin of species*. PF Collier & son New York.

[5] A.E. Eiben and J.E. Smith. 2015. *Introduction to Evolutionary Computing*. Springer Berlin Heidelberg, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-44874-8

[6] Ahmad Basheer Hassanat, Esra'a Alkafaween, Nedal A. Al-Nawaiseh, Mohammad Ali Abbadi, Mouhammd, Alkasassbeh, and Mahmoud Bashir Alhasanat. 2016. 1 Enhancing Genetic Algorithms using Multi Mutations : Experimental Results on the Travelling Salesman Problem.

[7] Silja Meyer-Nieberg and Hans-Georg Beyer. 2007. *Self-Adaptation in Evolutionary Algorithms*. Vol. 54. 47–75. https://doi.org/10.1007/978-3-540-69432-8_3