Evolutionary Computing - Assignment 1

Group 52

Alex Mourier *1234567*
Max Zwager *2709802*
Mickey van Immerseel *2693643*

Artificial Intelligence
Vrije Universiteit Amsterdam
Netherlands
01/10/2021

1 INTRODUCTION

1.1 Evolutionary computing

Evolutionary computing is a type of algorithm which utilizes the principles of Darwinian evolution in an effort to create the most efficient method of solving a particular problem. Through the use of iterative updating an initial solution through mutations across multiple generations and elimination of the weakest/selection of the strongest, an optimal solution to an optimization problem can be found. This technique has been utilized to solve all kinds of problems, from Graph Coloring [4] to being utilized by NASA design antenna for applications involving stringent, conflicting, or unusual design requirements which would outperform their human designed counterparts [10]. Many fields such as e.g. bioinformatics, networks and robotics found that genetical algorithms can perform as well, if not better, than their human counterparts [11].

Evolutionary algorithms have previously been utilized to solve games such as football simulations [5] and strategy games [13],[1]. They have even been utilized in the actual infrastructure of games such as in the Galactic Arms Race [12] to great success. However, surprisingly enough, so far there is little research on this topic, which motivates the current research.

1.1.1 Research question. Within this report we will be comparing two genetic algorithms (EA) and their performance on a game called EvoMan, which is inspired by the "Mega Man II" game from 1987, but has been slightly altered to only include the final boss fights[3]. The main goal of this report is to find out which EA can achieve the highest mean fitness of the population. For our EA's, we will be utilizing the DEAP framework which offers a pre-existing structure for these types of algorithms [6, 15]. Using the framework, two EA's are created to allow for a later comparison with fitness being the measure of success. The performance is compared by plotting the results for the fitness statistics over generations, and average fitness of the best solutions provided by the EA's. Finally, a statistical analysis is done to prove significance.

The following chapter will elaborate more on the EvoMan framework, DEAP framework and the algorithms that were developed. Note that the words *algorithm* and *EA* are used interchangeably.

2 METHODS

2.1 EvoMan

The EvoMan framework is a game platform for the development and testing of optimization algorithms. The game consists of a main character (played by our algorithms) that must defeat one or more of a total of eight unique bosses, often equipped with multiple attacks. The EvoMan framework provides us with a number of sensors (n=20) which can be used as an input for some model to generate the players' actions (walk left, walk right, jump, shoot, and release of the jump). The framework also provides a simulation function which returns the fitness of an individual using a predefined function.

In this paper we will make use of a player controller which is a single hidden layer neural network with 10 hidden neurons, which is provided in *demo_controller.py*. The neural network is chosen as it provides us with an evolutionary optimization problem of which the weights of the neural network (n=265) form the genotype. In this manner the genotype can be seen as a list of floating point numbers with a length of 265, which itself maps to a certain neural network instance which defines the players' behaviour.

As is mentioned before, the framework also provides a fitness function which is used in simulation to return the fitness of an individual (1). The fitness F of an individual player is a weighed average of one minus the enemy players' health h_e and the players' health h_p, and added a log of the total time t that has passed. It is chosen to use the original fitness function for both EA's because it combines all important values for evaluation in a smart manner. The weighed average not only forces the player to win (minimizing the enemy players' health) but also to prevent damage to itself. Adding the time factor forces the player to be more efficient and saves computation time.

$$F = 0.9 * (100 - h_e) + 0.1 * h_p - log(t) \tag{1}$$

2.2 DEAP

As mentioned in the introduction, we made use of the DEAP framework. The framework lends itself to the task perfectly, as users get the option to build custom EA's in an intuitive manner by choosing their own individual and population instantiating functions and evolutionary operators[7]. Using the framework, two EA's were developed which share the same body that is based on the *simpleEA* algorithm, which can be found on the DEAP documentation website [15]. However, the selection, mutation and crossover operators for the EA's are of different kinds and have different parameters. Note that the algorithms do use the same base fitness function (1).

The main body of both EA's consists of first instantiating a population of 100 individuals with random genotypes, followed by a loop repeated once for every generation containing the following operator classes: selection, crossover, mutation, normalisation. Whereas EA's exploit the evolutionary principle, they are not restricted by the same boundaries as natural evolution. This is also the case with both algorithms, which will soon become clear.

2.2.1 Algorithm 1. The first algorithm contains not only the same body of the *simpleEA* algorithm from the DEAP documentation website, but also its operarators. Tournament selection is used in which 8 individuals are randomly chosen from the population (with replacing). The individual with the highest fitness is returned as the victor. This process is repeated 100 times (population size) and generates a selection of individuals to which crossover can be applied. Note that since there is a predefined possibility of crossover, the selection of individuals may contain duplicates which, if not participating in crossover, end up in the final new generation. This is an example of how the developed EA may differ from natural evolution.

For crossover we used two-point. Two points are selected on the parent chromosomes and exchanged, which results in two offspring. The crossover operator is used to avoid that an exact duplicate of the parents from the old population shows up in the new offspring,

enabling both exploration and exploitation by exchanging genes of individuals with high fitness (selected in the tournament) and exchanging genes with other different individuals [2].

Finally we made use of Gaussian mutation with individual mutation probability. Given a certain probability, if an individual is to be mutated, every gene of that individual has a independent probability of 0.2 to be mutated. The mutation itself is an application on the current value of the gene of a random value taken from a normal distribution with μ 0 and σ 1. After mutation the weights of an individual are normalised. The parameters for tournament size and mutation are the result of rigorous testing, though lack of time may have prevented us to find the optimal values.

To balance exploitation and exploration, we will make use of dynamic Decreasing of High Mutation ratio/dynamic Increasing of Low Crossover ratio (DHM/ILC). Previous research has suggested this dynamical balance can increase performance when dealing with small populations [8]. Starting the experiment, the individual chance of crossover is set to 0 and the chance of mutation to 1. This results in maximum exploration. During the run, the chance of crossover is increased to almost 1 and the chance of mutation is decreased to almost 0, both with step-size 1/n with n being the number of generations.

2.2.2 Algorithm 2. The operators for the second algorithm are the most effective for defeating the EvoMan enemies according to literary research.

The combination of two selection operators, namely roulette-tournament is used. Roulette and tournament are the most commonly used selection operators. As it turns out, according to Rahman et al., the combination of the two selection operators performed better than either separately [16]. The combination is implemented by first performing the roulette selection, after which the tournament selection is executed over the previously selected data.

For the crossover operator of the second algorithm, two-point is used as well. Some research was found comparing the effectiveness of crossover operators. The OX (Ordered Crossover) came out as best operator to achieve the best solutions [14]. However, we could not use this crossover operator as we worked with floats and not integers. In the DEAP framework only a couple operators work with floats and we came to the conclusion that the two-point crossover is the most suitable for this task.

The mutation operator shuffle indexes is used. According to Hassanat et al. the exchange type mutation operators perform well when the algorithm makes use of a single mutation[9]. This still leaves some room for personal preference which mutation operator to choose for this research. Out of the mutation operators provided by the DEAP framework, the shuffle indexes seems to be the closest to what is described in that paper.

Whereas the first algorithm made use of dynamic probabilities for the mutation and crossover, the second algorithm uses static values. These values are 0.9 as the probability for crossover and 0.03 as probability for mutation. These numbers were picked out of the same article that described the dynamic probabilities for the first EA [8]. Table 1 shows a simplified comparison between the two algorithms.

2.2.3 Hypothesis. The DHM/ILC came out as the better option as opposed to the static probabilities for cross-over and mutation in the article by Hassanat et al.[8]. However, we expect to see that the effectiveness of the researched operators work comparatively or even better than the dynamic probabilities in terms of creating an effective EA.

Table 1: EA comparison

EA steps	**EA 1**	**EA 2**
Population	Size=100 Generations=50	Size=100 Generations=50
Hyper-parameters	DHM/ILC	$prob_{cross}$=.9 $prob_{mut}$=.03
Selection	Tournament (size=8)	Tournament-Roulette (size=8)
Crossover	Two-point	Two-point
Mutation	Gaussian (μ=0, σ=1)	Shuffle indexes $prob_{gene}$=.03
	Normalisation	**Normalisation**

2.3 Experimental set-up

To compare the effectiveness of the algorithms, they are implemented in an optimization problem for three different enemies. The chosen enemies were enemy number three, seven and eight. These enemies were chosen because they are not the easiest to solve, have differences in behaviour and therefore would be a good selection to test the effectiveness of the EA's. For every optimization experiment the evolutionary process is independently run 10 times. Statistics such as the averages of the maximum, means and standard deviation of the fitness are gathered for these runs. These statistics are plotted over every generation, showing the population fitness properties over time. This is done for every enemy containing the data of both EA's resulting in three line plots. Moreover, for every run of both EA's the best solution is run five times of which the average score is taken. For every enemy the average fitness of the best solution of the 10 runs per EA is plotted in a boxplot, resulting in three boxplots containing the individual gain for each EA.

Finally, a statistical analysis is performed on the gathered data to make a statement about a possible significant difference in performance between the two EA's. To compare the results of the different algorithms a Wilcoxon test is used on the mean fitness of the generations of each run, per enemy. This test is the nonparametric version of the t-test for dependent samples as the data is not normally distributed and there are only 10 runs to be compared per enemy. The Wilcoxon test gives answer to whether the two algorithms significantly differ in their mean fitness per enemy.

3 RESULTS AND DISCUSSION

3.1 Results

Both EA's were run on different devices overnight, taking approximately 3-4 hours to run each enemy. After receiving the raw data, the evaluation script was run for each EA which took a few minutes. Finally, plots were made using the output from the evaluation using the plotting script and the statistical test was run using the stat_test script.

3.1.1 Plots. Figure 1 shows the population statistics over all generations, containing the best, mean and standard deviation of the population fitness. Note that in the plots the first and second EA are referenced as *spec_1* and *spec_2* respectively. Figure 2 contains the boxplots that show the average fitness of 5 runs of the best solution of each run for both EA's, resulting in the fitness of the best solution per EA. These values are also finally used in the statistical analysis.

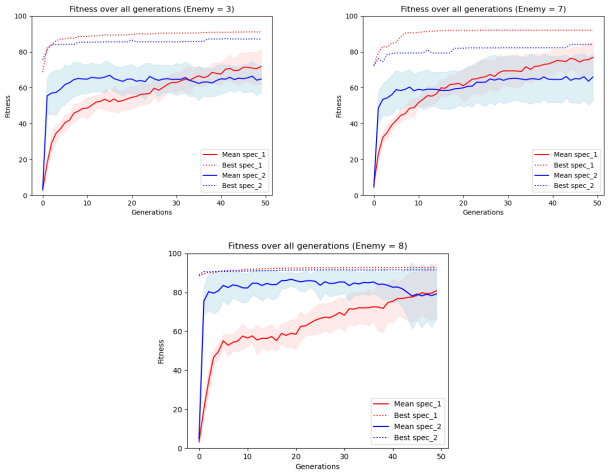

Figure 1: Population statistics over generations

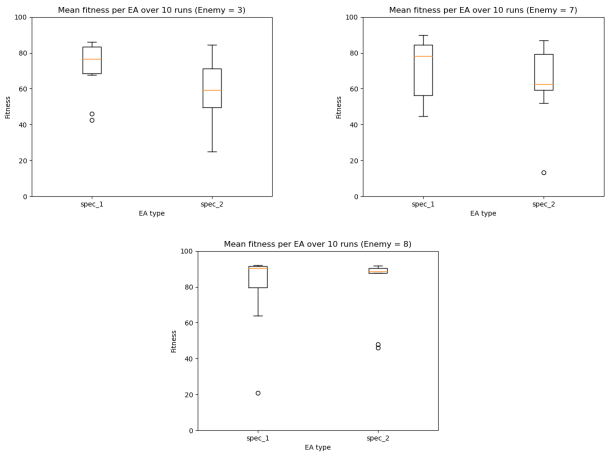

Figure 2: Best solution fitness per enemy

3.1.2 Statistical analysis. Table 2 shows that only for enemy three the *p-value* for the Wilcoxon test is significant (*p-value<0.05*) whereas this was not the case for enemy seven and eight.

3.2 Discussion

According to the Wilcoxon test only for enemy three the algorithms differ significantly in performance. This means that only for this enemy the null hypothesis is rejected. An overall performance difference between the algorithms is hard to point out, which corresponds to our hypothesis. Looking at the line plots both EA's show different behaviour. Whereas the mean population fitness moves up slowly for the first EA, the fitness of the second EA peaks much earlier. This indicates that the performance is influenced by the number of generations. The first EA also does tend to find the best solution earlier. This goes against intuition, as mutation is highest at the start which is only an exploitation strategy. The deviation from the mean also seems to differ between the two EA's. Whereas the first EA seems to have a slowly increasing standard deviation, the standard deviation of the second EA seems to be relatively high from the start and stays constant. This is clearly caused by the DHM/ILC approach.

Table 2: Wilcoxon test

	Wilcoxon	**p-value**
Enemy 3	266.0	*p=0.0003*
Enemy 7	498.0	*p=0.1781*
Enemy 8	18.0	*p=2.2286*

3.2.1 Comparing the results to the base article. The EvoMan framework is provided with a base article describing the effectiveness of their demo-algorithms [3]. Since we have used the same fitness function as them we can try to compare the results. The article unfortunately uses individual gain instead of fitness to compare their algorithms, there are no clear fitnesses to use. They do test different combinations of training sets with the NEAT algorithm that shows the final fitness as the average fitness of the agent with respect to the training enemies [3]. We can somewhat compare these numbers to the mean of the average fitness of our EA's. These numbers are visible in Table 3, as one can see the fitness of the base article is higher than ours. This could be due to multiple reasons, including the way they derived these fitnesses by training with multiple enemies instead of one.

Table 3: Average fitness

	Base article	**EA1**	**EA2**
Enemy 3	85 (trained with enemy 7)	56.20	62.77
Enemy 7	88 (trained with enemy 8)	61.27	60.63
Enemy 8	90 (trained with enemy 3)	63.05	81.59

4 CONCLUSIONS

The overall performance between the two EA's are comparable. The performance of the EA's seems to be influenced by the amount of generations. This indicates that one of either EA may be the better choice given a certain amount of generations. Moreover, EA number one seems to find the best solution fastest. Based on the average fitness found in the base article [3], our algorithms have a lower mean average fitness. This could be due to the difference between the training methods. For future research we propose a combination of the EA with dynamic probabilities and the EA with the best operators according to literature.

REFERENCES

[1] Kai Arulkumaran, Antoine Cully, and Julian Togelius. 2019. Alphastar: An evolutionary computation perspective. In Proceedings of the genetic and evolutionary computation conference companion. *Journal of Computer Science and Technology* 27, 5 (2019), 314–315.

[2] Pratibha Bajpai and M. Kumar. 2010. Genetic Algorithm - an Approach to Solve Global Optimization Problems. *Indian Journal of Computer Science and Engineering* 1 (10 2010), 199–206.

[3] Karine da Silva Miras de Araujo and Fabrício Olivetti de Franca. 2016. Evolving a generalized strategy for an action-platformer video game framework. (2016).

[4] J.K. van der Hauw E. Eiben and J.I. van Hemert. 1998. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics* 4, 1 (1998), 25–46.

[5] Antonio J. Fernandez, Carlos Cotta, and Rafael Campaña Ceballos. 2008. Generating Emergent Team Strategies in Football Simulation Videogames via Genetic Algorithms. *GAMEON* 11, 3-4 (2008), 120–128.

[6] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary algorithms made easy. *The Journal of Machine Learning Research* 13, 1 (2012), 2171–2175.

[7] Felix-Antoine Fortin, Marc Parizeau MARC Francois-Michel De Rainville, Marc-Andre Gardner, and Christian Gagn. 2012. GDEAP: Evolutionary algorithms made easy. *The Journal of Machine Learning Research* 13, 1 (2012), 2171–2175.

[8] Ahmad Hassanat, Khalid Almohammadi, Esra' Alkafaween, Eman Abunawas, Awni Hammouri, and VB Prasath. 2019. Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach. *Information* 10, 12 (2019), 390.

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

[10] Gregory Hornby, Al Globus, Derek Linden, and Jason Lohn. 2006. Automated antenna design with evolutionary algorithm. *In Space* 27, 5 (2006), 7242.

[11] John Koza. 2010. Human-competitive results produced by genetic programming. Genetic programming and evolvable machines. *Genet Program Evolvable Mach* 11, 3-4 (2010), 251–284.

[12] S.M Lucas and G. Kendall. 2006. Evolutionary computation and games. *IEEE* 1, 1 (2006), 10–18.

[13] Antonio M. Mora, Antonio Fernández-Ares, Juan J. Merelo, Pablo García-Sánchez, and Carlos M. Fernandes. 2012. Effect of noisy fitness in real-time strategy games player behaviour optimisation using evolutionary algorithms. *Journal of Computer Science and Technology* 27, 5 (2012), 1007–1023.

[14] Abdoun Otman and Abouchabaka Jaafar. 2011. A Comparative Study of Adaptive Crossover Operators for Genetic Algorithms to Resolve the Traveling Salesman Problem. *International Journal of Computer Applications* 31 (10 2011). https://doi.org/10.1504/IJCAT.2017.10005868

[15] DEAP project. [n. d.]. DEAP Documentation. ([n. d.]). https://deap.readthedocs.io/en/master/index.html

[16] Rosshairy Abd Rahman, Razamin Ramliand Zainoddin Jamari, and Ku Ruhana Ku-Mahamud. 2016. Evolutionary Algorithm with Roulette-Tournament Selection for Solving Aquaculture Diet Formulation. *Mathematical Problems in Engineering* 2016 (2016), 10. https://doi.org/10.1155/2016/3672758

5 AUTHOR CONTRIBUTIONS

Max Zwager has written the code for the first EA and the plotting. Mickey van Immerseel has written the code for the second EA and the statistical test. All three of us worked on writing the paper.