Evolutionary Computing Assignment 2: Generalist Agent

Group 60: Madhur Pawar [2761591], Michal Sanocki [2793387], Muska Neek [2552431], Sarah Hassouna [2601420], Shreya Ghose [2761536]
Vrije Universiteit Amsterdam

1 INTRODUCTION

This report continues on a previous experiment in which the objective is to gain more experience with Evolutionary Computation. Two evolutionary algorithms are developed, applied and compared for the task of video game playing. The python framework *EvoMan* is used for this purpose. Evoman is a platform game, where the player controls a character/avatar, usually with humanoid form, in an environment characterized by differences in altitude between surfaces ("platforms") interspersed by holes/gaps [1]. These experiments are focused on the implementation of two algorithms in the context of playing the EvoMan game. In this framework a player agent and a set of eight enemy agents are present. Each enemy has their own different behavior and set of abilities. This entails that the player agent needs to be able to exhibit different strategies to defeat each enemy. The eight enemies are Flashman, Airman, Woodman, Heatman, Metalman, Crashman, Bubbleman and Quickman and will be referred to by their chronological number in this sequence. In the previous experiment two specialist evolutionary algorithms (EAs) or methods were compared on their ability to train a specialist agent in winning against different enemies. The effect of the algorithms on the individual gain of the solutions was assessed. The same two EA methods using the simulation mode "individual evolution" of the framework will be used in this experiment. However this time they will be used with the purpose to train a generalist agent. So in this case a general solution is looked for. According to the no-free-lunch theorem it could be the case that algorithms performing well on one type of problem, perform worse on another [2]. Therefore it is interesting to test if these EA methods used in the previous experiment will perform similarly in this experiment as it involves different types of problems.

A neural network is used in the algorithm which contains one hidden layer with ten hidden neurons. The aim is to find the best solution containing the weights for the neural network with ten hidden neurons. The best solution coming out of this experiment will be submitted to use in a class competition. This competition has two ranks based on the number of defeated enemies and the Gain measure. These ranks entail that the best solution of this experiment should strive to defeat the highest number of enemies possible and have the largest possible energy points kept by the player. Also a low sum of get-time is desirable and the sum of player-life is taken into account. This report will explain how the best solution can be found. Research question: Which enemies are more easily defeated when using feedforward Neat algorithm for the Evoman framework?

2 METHODOLOGY

It is important that the agent player evolves efficiently. A feedforward neural network inspired by the NEAT algorithm with ten hidden neurons is used to do so. A genetic algorithm generates a population of 100 from which fifteen generations are to form. A random mutation and one-point crossover is applied. Where ten runs are iterated for every generation and the parents of the next generation will consist of the twenty percent of the best performing individuals.

2.1 Motivation of Algorithms and Fitness Functions

The two EAs used will be referred to as EA1 for algorithm 1 and EA2 for algorithm 2. The algorithms are the same except for the fitness function that is used. The fitness function of EA1 is given in equation 1.

$$fitness = \gamma \times (100 - e_e) + \alpha \times e_p - logt \tag{1}$$

It is the default fitness function of the EvoMan framework in which γ is a constant representing the players energy at the end of the game with value 0.9 and α is constant representing the same for the enemy with value 0.1. The purpose at the end of the level is to maximize the energy of the player and minimize the energy of the enemy. The t represents the number of time steps in the game. This last factor is disregarded in the second method EA2. This algorithm uses the fitness function given in equation 2.

$$fitness = \gamma \times (100 - e_e) + \alpha \times e_p \tag{2}$$

The logarithmic function of time is not taken into account in the fitness function of EA2 and thus does not take the number of time steps of the game into account. In the previous experiment a statistically significant difference was found in the performance between the two methods. For this experiment the same EAs are used but for the training of a generalist agent. There are two groups of enemies (games) used to experiment with the algorithms. For each group an independent experiment evolving a generalist agent is made. The group sizes are set to three. The enemies used in the groups are

as followed: group 1 consists of enemies 2 (Airman), 5 (Metalman) and 8 (Quickman). Group 2 consists of enemies 3 (Woodman), 4 (Heatman) and 7 (Bubbleman).

2.2 Parameter Settings Experimental Setup

In the Evolutionary Computing field a big challenge is presented when looking for the right parameter values for evolutionary algorithms [2]. Understanding how controlling or tuning the values of parameters of an EA influences the process is important as it has the potential of adjusting the algorithm and while doing so it can also solve the problem at the same time [3]. This section explains how the parameter values were set during this experiment and how this influenced the process.

Some environmental parameters are changed from the default values. The enemy mode value was set to *static* and the multiple mode value was set to *yes*. The environmental parameters of difficulty level of the game and contact hurt are not changed from their default values. These are set to *2* and *player* respectively.

The procedure of the parameter tuning was in large part similar to that of the specialist agent. However, the development of the generalist agent required us to train our algorithms on the set of multiple enemies in order to achieve better results against all enemies. While working with algorithm 1, a few different enemy groups and parameter settings were tested. Some of the few enemy groups that were tested were: [1,2,5], [2,5,7,8], [7,8] and [6,8]. A general trend that was observed for both the above groups and the groups chosen by us that the performance of the algorithm deteriorated when the number of generations was increased. The mean and best fitness went down rather than increase when the number of generations was 20 or 30 compared to when the number of generations was set to 15.

Similarly, for the second algorithm, we tested different combinations of parameters including the same groups and the number of generations. Our results for the second algorithm were largely similar, however, we found that the population of 70 gave us better results than that of 100, both in terms of performance and gain, against all enemies. Once each run experiment was finished algorithm was tested against all enemies (experiments were repeated 5 times). The algorithms, in general, almost always wins against enemy 2 and has good player energy points for enemies 5, 7 and 8. It does not perform well against enemies 1, 3, 4 and 6. Therefore, we picked enemies 2, 5 and 8 in group 1 and 3, 4 and 7 in group 2 (figure 1.).

Using the environment parameters, the enemies are activated and the player plays against each enemy once before iterating it for the generation. Each run then consists of 15 generations. A run is then iterated 10 times. During testing, the final experiments are repeated ten times independently with both algorithms for each group of training enemies.

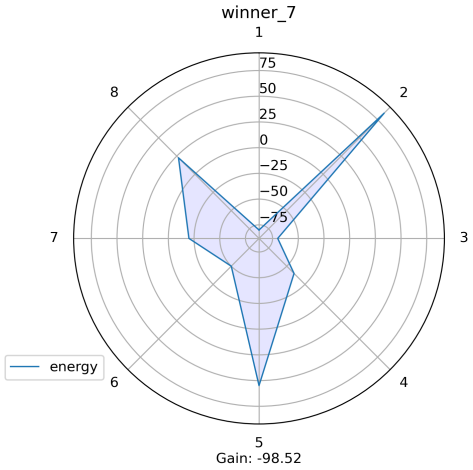

Figure 1: The gain of the best solution against each enemy

2.3 Budget

Our initial runs were made with a group of three enemies, a population of 100 and 15 generations. For this particular experiment, we worked with 3 computers. Specifications include:

Computer 1: i) 16 GB RAM ii) 11th Gen Intel i5-1135G7 @2.40GHz processor, 64 bit OS

It took nearly 14 hrs to complete an initial run.

Computer 2: i) 8 GB RAM ii) 10th Gen Intel Core i5-10210U @1.60GHz 2.11 GHz processor, 64 bit OS

In this case, it took nearly 16 hrs to complete an initial run.

Computer 3:i) 8 GB RAM ii) Intel Core i5 3,1 GHz processor, 64 bit OS

In this case, it also took 16 hrs to complete an initial run.

The group required two days to code and debug our algorithms. Considering we had 12 days to train and test our models, the number of working hours available to each computer was 192 (16 hour work-day). Hence, the total number of hours available to our group was 576. The group thus had a best-case budget of 12.5 experiments (576 divided by (12+16+16)) over the course of 12 days to arrive at our solutions.

3 RESULTS AND EVALUATION

The comparison of EA1 and EA2 by training group is done by plotting the average/standard deviation of the mean and maximum fitness across the generations with a line-plot. Here the average over the ten runs of the mean and maximum over the population in each generation is calculated and visible in the line-plots in figures 1 to 4. Figure 1 tells us that for EA2 enemy 2,5 and 8 both the maximum and the mean seem to be increasing with more generations, however tends to stabilize after a while. While Figure 2 shows for EA2 and

enemies 3,7 and 4 the maximum seems rather stable right away and the mean seems to be increasing for the first 10 generations. The opposite is the case in Figure 3 where EA1 was used for enemies 2,5 and 8 where the mean is stable right away and the maximum seems to variate while 55 and 80, however stabilizing in more generations. Figure 4 using EA1 for enemies 3,4 and 7 show a mean and maximum that are both quite rather stable. These plots are by training group.

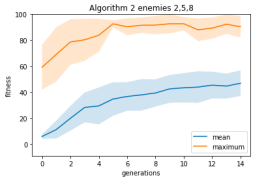

Figure 2: Evolutionary algorithm 2 with enemies [2,5,8]

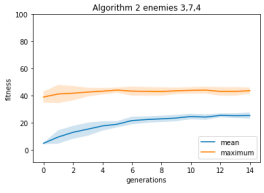

Figure 3: Evolutionary algorithm 2 with enemies [3,7,4]

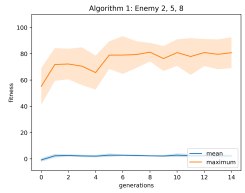

Figure 4: Evolutionary algorithm 1 with enemies [2,5,8]

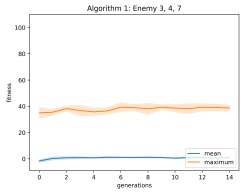

Figure 5: Evolutionary algorithm 1 with enemies [3,7,4]

The points in the box-plot are the values of the mean of the five times that are calculated for each solution of the algorithm. Figure 5 shows the box plots for the algorithms and their performance against the enemy groups. Each box contains ten data points that represent the average of five points. According to Figure 5 EA2 group 1, containing enemies 2,5 and 8, seem to be performing the best in terms of individual gain.

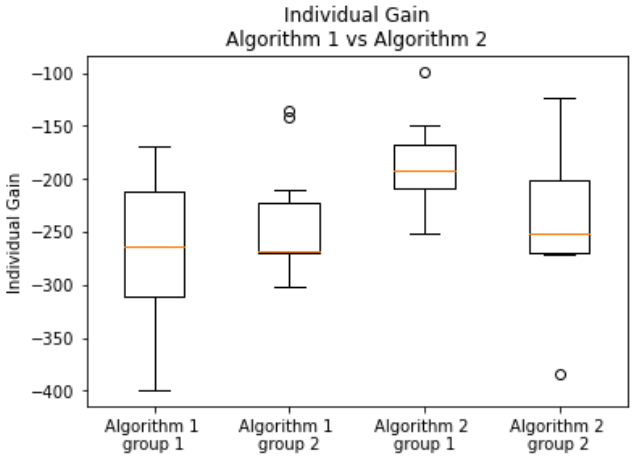

Figure 6: Individual gain EA1 vs EA2 for every group of enemies

A statistical test to verify if the differences in the average of these means are significant between the groups of best solutions when comparing the two algorithms is done.

EA 1 vs EA 2	Enemies [2,5,8]	Enemies [3,4,7]
Statistic	-2.963	0.004
p-Value	0.010	0.996

Table 1: T-Test Results of EA 1 vs EA 2

Whereas as seen in Table 1 difference between EA1 and EA2 is statistically significant for enemies 2,5 and 8. Which means that for group 1 there is a significant difference in EA while in group 2 both EA's have a similar performance.

3.1 Best Solution

The solution chosen for submission for the class competition is our Algorithm 2 playing successfully against enemies Airman, Metalman and Quickman (2, 5 and 8). The following table takes the average over 5 repetitions of energy points of player and enemy for each of all enemies for the best solution.

Enemy	Player Energy	Enemy Energy
Flashman	0	80
Airman	84	0
Woodman	0	70
Heatman	0	40
Metalman	57.5	0
Crashman	0	50
Bubbleman	0	20
Quickman	22.6	0

Table 2: Energy points of Player vs Energy points of Enemy

4 CONCLUSION

From the experimentation, it is evident that a feed-forward neural network (NEAT) with 10 hidden neurons performs considerably well against the enemies defined within the Evoman framework. Both of our evolutionary algorithms work better against the enemy group [2,5,8] compared to the enemy group [3,4,7]. Removing the logarithmic dependency on time in the defined fitness function, provides better mean and best values of fitness compared to the default fitness function.

Prospective future work: It may be interesting to see how the evolutionary algorithms perform against the enemy groups if the rate of mutation is increased or the inputs to the neural network are normalized.

REFERENCES

[1] S. Karakovskiy J. Togelius and R. Baumgarten. The 2009 Mario AI Competition. EEE Congress on Evolutionary Computation.

[2] S.K Smit and A.E Eiber. 2014. Parameter Tuning of Evolutionary Algorithms: Generalist vs. Specialist. (2014). https://doi.org/10.1007/978-3-642-12239-2_56

[3] Robert Hinterding Ágoston E. Eiben and Zbigniew Michalewicz. 1999. Parameter Control in Evolutionary Algorithms. (1999). https://doi.org/10.1109/4235.771166