Task I: Specialist agent

Team 62

10/2021
Diana Jumaili 2703794
Josip Grguric 2704719
Niharika Chaubey 2704213
Loïc Macken 2611927

ACM Reference Format:
10/2021, Diana Jumaili 2703794, Josip Grguric 2704719, Niharika Chaubey 2704213, and Loïc Macken 2611927. . Task I: Specialist agent: Team 62. In . ACM, New York, NY, USA, 4 pages. https://doi.org/

, ,
.

1 INTRODUCTION

In biology, mutation is a crucial source of genetic variation. Mutation allows the creation of new DNA sequence for genes, making new alleles [1]. Analogous to this, in the world of Evolutionary Algorithms (EA), the mutation operator has the extraordinary role of maintaining diversity. Typically, low diversity leads to premature convergence, where the best solutions are at a local optimum [2]. Therefore, mutation becomes a desired operator to find the global optimum. As it can help further explore of new areas in the search space.

1.1 Research goal

Our main research question is how the presence of mutation operator in EA can help find the optimal solution. To answer this question, we used the NeuroEvolution of Augmenting Topologies (NEAT) algorithm and manipulated the base mutations (nodes, links and weights). Additionally, we try to investigate the effect of mutation in our search by tracking the diversity between genomes.

2 METHOD

The investigation was performed in the EvoMan framework- a game environment. In short, EvoMan allows for agents playing the game (player agent or 8 enemies) to evolve overtime. Specifically, for this project we focused on single evolution of the player agent where the player must learn how to maximise their points by performing one of five actions: move left, move right, jump, release jump and shoot.

We wrote the NeuroEvolution of Augmenting Topologies (NEAT) genetic algorithm from scratch. Using NEAT allows the topology to adapt to our task from a minimal structure. The sensors outputs were given to the evolving ANN as input, whereas the network outputs 5 neurons for each action to be taken by the agent. The activation function used was sigmoid and any output neuron with a value higher than 0.5 means that the agent will perform that action. For the mutation condition (EA1) we made the probability of mutation for links at 30%, for nodes at 3%, and for weights at 80%. However, for weight we had a 90% that it is uniform, otherwise random. Whereas, for the control condition (EA2) the pm for nodes, links and weights were set at 0%. We also allowed for cross species crossover with a chance of 0.01%.

Lastly, for both EAs, the diversity analysis was only done on one enemy. We chose enemy 6 as it required more generations and a higher population. As a measure of diversity, we decided to keep track of the delta difference between the best genome of each generation. In summary, delta is the natural measure of the compatability distance between two genomes, represented by their disjoint genes, excess genes, and average weight difference of matching genes [3].

2.1 NEAT

The NEAT algorithm starts with the simplest ANN and becomes increasingly complex over generations. Concerning encoding, the algorithm has two lists of genes and a series of nodes and connections. When mutation is taking place, it is either the existing connections that are mutated or a new connection or node are added to the network. A new connection is randomly assigned a weight if it is added between a start and an end node. A new node will be placed between two nodes that are already connected, whereas the replaced (old) connection will still be present in the network but disabled. The weight of the start node will be the weight of the replaced connection and the new node is linked to the end node with a weight of 1.

For the selection operator we use tournament selection. Crossover in the NEAT algorithm uses the notion of homogeneity, in which historical markings will keep track of each parent's genome and match the same genes in both parents before crossing to avoid dysfunctional offspring. Consequently, the NEAT algorithm can perform speciation in which variation is preserved in the population. Based on the similarity of topology and connections inside the population, the algorithm will split it up into several species and set a limit as to how different the species can be - creating a pressure toward diversity. We set the delta threshold value at 4.0, we have this value to allow more room for each species. C1, C2, C3 are values used in the calculation in the delta. C1 corresponds to the weight of the excess genes, C2 is to disjoint and C3 is to the weight difference between two genes. The default values were 1, 1 and 0,8 respectively, where we have lowered C1, C2 to 0.8.

Termination will execute when the population's average fitness is reached. We chose the average to make sure that it is not one "lucky" individual that caused the termination. Then we can use that species and take the three (again to make sure that more than one in the population managed to reach the fitness average) best performing genomes out of them, test 10 times with a specific enemy. This will be done using both of our algorithms.

2.2 Fitness function

In this experiment we used the default fitness function given to us by the Evoman framework. The purpose of the fitness function is to give our algorithm an indication of how close a given solution is to achieve our goal. The fitness function defines fitness to take less time in lowering the enemy health and more player health. The values in the fitness function were 0.9 and 0.1. Below is the fitness function.

$$0.9 * (100 - enemylife) + 0.1 * playerlife - ln(time)$$

We added a constant of value 6.9 to make sure that the fitness value never becomes negative.

3 RESULTS

For each of the two evolutionary algorithms, the data was collected from the plays against each enemy and the maximums and means of the 10 runs and plotted against the generations, this is shown in Figure 1. For each run, the best individuals were collected at the last generation and these genomes were run another 5 times, this data is shown in Figure 2.

In order to compare the two evolutionary algorithms and determine whether they can be proved to be differently distributed, T-tests were applied to the two algorithms for each enemy. The p-values found are 0.253, 0.957, 0.0873 for enemies 2, 4 and 6 respectively. Unfortunately these p-values are not below the required threshold in order to be statistically significant (which usually is

0.05) although it should be noticed enemy 6 is close to statistical significance. A likely cause for no statistical significance is the relatively low number of runs for each generation. Perhaps better results could be achieved by increasing the number of runs, but this would require either more running time or more powerful computers.

Furthermore, in order to look at the diversity we used delta difference as a measure. For EA1 and enemy 6, we found the average delta value between each generation to be 1.58. For EA2 and enemy 6, the average delta value was 0.67. This gives a difference of almost 1 point, implying one of the following two reasons: (1) EA2 takes longer to get to a good candidate solutions or (2) EA2 gets stuck in local optima, thus it can not find better solutions. Both of these reasons lead us to believe that EA1 provides more diversity, and thus reduces the chances of either possibilities.

4 CONCLUSIONS

Looking at the figures, we can conclude that there is an observable difference in the performance of EA 1 and EA 2, which is especially visible when playing against enemies 4 and 6. It seems that enemy 2 does not require an elaborate strategy, thus the added benefit of hidden layers and evolving network topology is minimal (Although there is still quite a discrepancy between the best performers of each algorithm). However, as the difficulty of the enemies increases, these differences become increasingly influential, up until the point where the algorithms play against enemy 6 and EA 2 becomes powerless. Thus allowing mutations and evolving topology gives an evolutionary algorithm the ability to learn more advanced strategies of playing and in that way it can manage to beat more challenging opponents.

For further research, it might be interesting to focus on enemy 6 and observe changes in fitness when altering the number of generations. Other parameters could also be changed, in order to find an optimal combination of parameters to reach a desired fitness as quickly as possible.

5 CONTRIBUTION INFORMATION

In the beginning of the project, we all attempted writing a different evolutionary algorithm to grasp the basics of it. For the continuation of the project the group voted for the best algorithm and we continued with that one. Niha:theoretical understanding, research question development. Diana: organization, methodological planning, theoretical understanding. Josip: Focused on coding and NEAT implementation. Loïc: Data processing and conclusion.

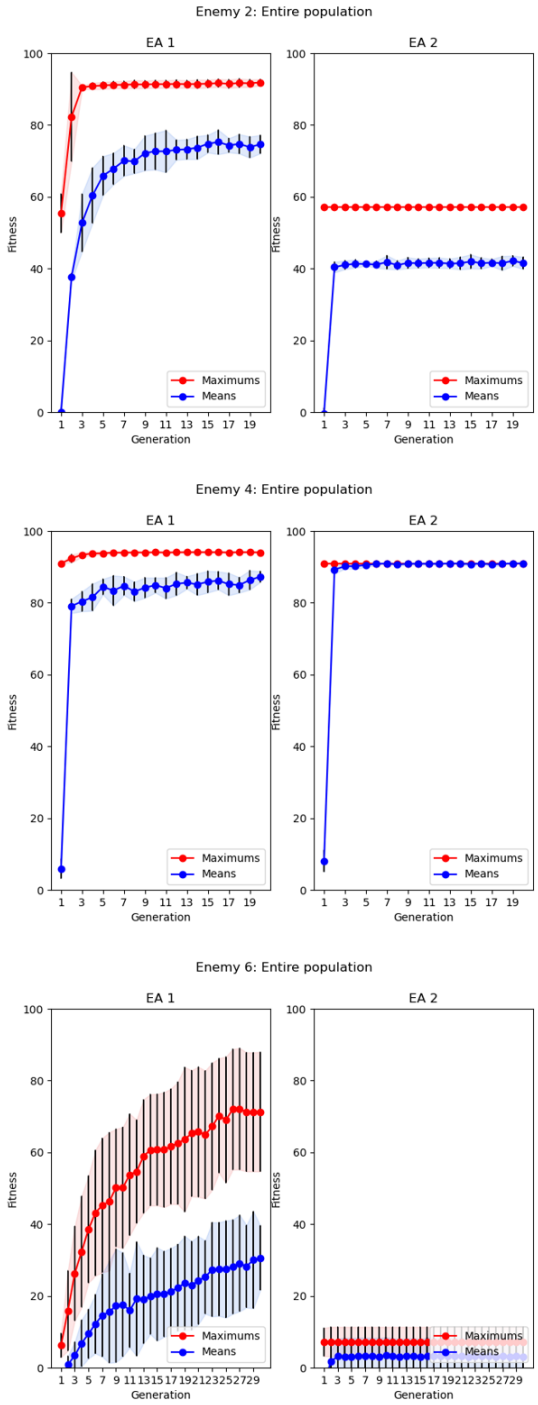

Figure 1: The maximums and means of the fitness of each evolutionary algorithm against each enemy, plotted against the generations.

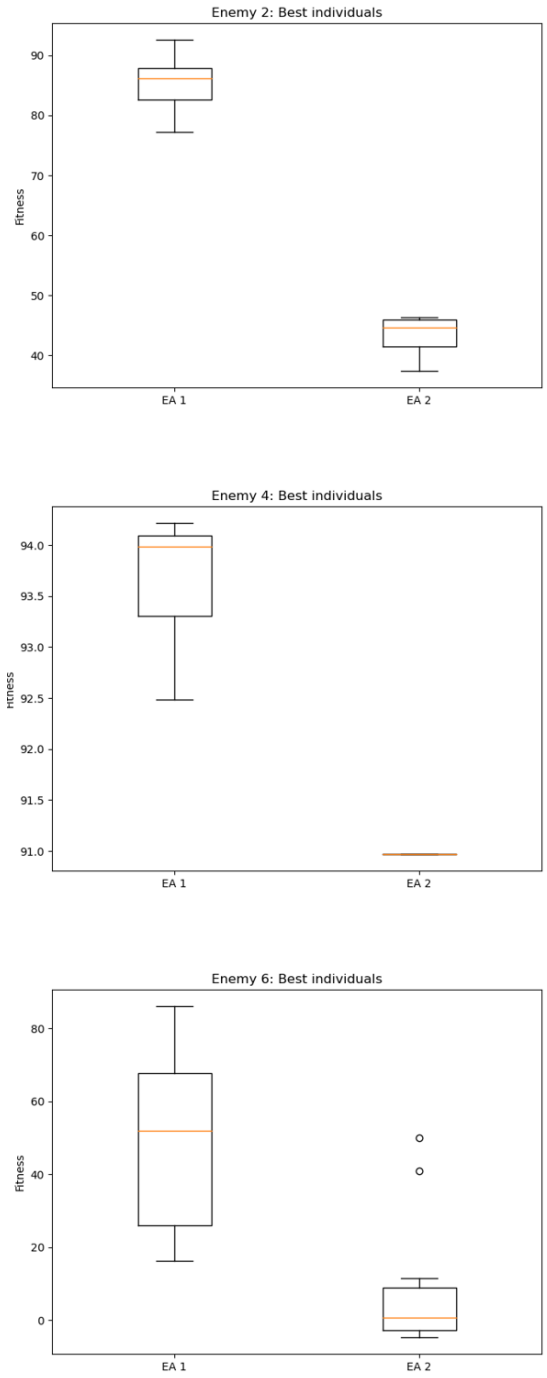

Figure 2: The fitnesses of the best individuals after the last generation of each evolutionary algorithm against each enemy.

REFERENCES

[1] Bruce A. McDonald. Population genetics of plant pathogens mutation, 2004. URL https://www.apsnet.org/edcenter/disimpactmngmnt/topc/PopGenetics/Pages/Mutation.aspx#:~:text=The20ultimate20source20of20all,specific20gene20through20intragenic20recombination.

[2] A.E. Eiben J.E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series, 2015.

[3] Kenneth O. Stanley and Risto Miikkulainen. Efficient evolution of neural network topologies. 2005. URL https://web.unbc.ca/~lucas0/papers/Old/20Papers/Efficient-Evolution-of-NN-Topologies.pdf.