A Comparison of Creep Mutation and Random Resetting Mutation Operators using NeuroEvolution of Augmented Topologies

Evolutionary Computing Assignment 1: Specialist Agent

Faculty of Science – Vrije Universiteit Amsterdam

Group 64

Members:

Levi Kamsteeg
2584230
lkg510@student.vu.nl

Ian Berkhout
2683737
i.berkhout@student.vu.nl

Daniel Bemerguy
2699826
d.a.b.abenatherbemerguy@student.vu.nl

Vladimir Minnebois
2046210
v.s.minnebois@student.vu.nl

ABSTRACT

This paper investigates the different affect Creep Mutation and Random Resetting have on the evolution of a NEAT genetic algorithm with regards to game strategy development by applying the two mutation methods separately in two experiments. The experiments were ran using the Evoman framework, which is a 2D boss fighter based on the Megaman series. Both experiments were repeated 10 times for 3 different enemies. Evolution of the neuralnet was achieved through a combination of elitism, crossover, mutation, and stagnation. The experiments showed that the used genetic algorithm was more effective in the development of game strategies for the simpler fights and that the development of the average fitness was better for creep mutation in comparison to random resetting. However, comparing the average results of the best genomes between experiments showed no statistical difference between the experiments.

KEYWORDS

NeuroEvolution of Augmented Topologies, Mutation, NEAT, Creep Mutation, Random Resetting, Genetic Algorithm

1 INTRODUCTION

Mutation plays a major role in evolutionary algorithms. Mutation operators introduce small noise in the genomes. This added noise has the intended effect to increase perpetuate diversity. All these mutations and noise have the goal to steadily increase the fitness for each population towards an optimum.

There has been a lot of research done about mutation, also on Creep Mutation and Random Resetting [1]. However, there is only limited research on the different effects of Creep Mutation and Random Resetting when a NeuroEvolution of Augmented Topology algorithm is used. Therefore, we wanted to add our research to the existing literature. We made a comparison between Creep Mutation and Random Resetting Mutation Operators using a NeuroEvolution of Augmented Topologies (NEAT) framework.

NeuroEvolution of Augmenting Topologies (NEAT) has multiple mutation operation methods, it can mutate a node's activation function, its connections, its response, and its weights, and it can also add complete nodes and connections. In our NEAT framework we limited the mutation options to node's activation function, its connections, its weights, and we also let it mutate complete nodes. We let it mutate complete nodes and connections because this protected us against stagnation and let us at any time keep diversity. However, for our research we focused on two mutation operators, the mutation that operates the activation function and weights. We compared two different methods to do these mutations, namely: Creep Mutation and Random Resetting. With Creep Mutation we refer to perturbations that are added to the original value, these perturbations are drawn from a normal distribution with a zero mean and standard deviation of 0.5. And with Random Resetting we refer to a mutation that is a randomly drawn value that replaces the original value, this randomly drawn value is drawn from a normal distribution with a zero mean and standard deviation of 0.5.

With this framework that will be further introduced in the following chapter, we expressed our research question as follows:

Is there a significant benefit in using Creep Mutation versus Random Resetting Mutation Operators using a NeuroEvolution of Augmenting Topologies Genetic Algorithm?

In the following chapter we will introduce our methods, go over our results and end with a conclusion.

2 METHOD

2.1 Framework

This research is executed using the Evoman framework with a game difficulty of 2. Evoman is computer game inspired by the Megaman bossfights and created in Python. The controlled character or otherwise known as the player can be controlled by either a human or algorithm [4]. For this research, a genetic algorithm is used known as NEAT.

2.2 NeuroEvolution of Augmented Topologies

NEAT is a method of evolving an artificial neural network that means that not only the weights and biases change during evolution but also the structure of the network [2]. The specific reason why this network was chosen is because by using this method the evolution can start with a relatively simple neural net, evolve into something that can catch the more complex aspects of the fight and hopefully allow us to gain an understanding about how the specified ways of mutation affect the evolution. The experiments are executed and programmed in python and the network will be constructed and evolved using the NEAT-Python library. NEAT-Python is an implementation of NEAT in python, that is not dependent on any other python library [3]. The initial neural network consists of 20 inputs and 5 outputs, which are fixed by the Evoman framework [4] and 1 hidden layer with 10 hidden neurons. This layer is added so that the evolution starts with a basic structure and the number of nodes is halved each layer.

2.3 Evolution Methods

The evolution is done in 3 major steps: simulation, parent selection and reproduction. Simulation is done by playing the Evoman game using the neural network for each genome in the population, this results in fitness (f) calculated through player energy (p), enemy energy (e) and time elapsed (t) ($f=0.9*(100*e)+0.1*p*\log(t)$) [4]. For this research, a population size of 50 is used, this number is large enough for a diverse set of genomes to be created but small enough for the estimated computational time to only be several hours. The entire population with corresponding fitness is passed to selection. The selection of the parents is done through two methods. First, by selecting the genomes with the highest fitness and assuring these are brought over to the next generation, which is also called elitism [3]. Secondly, by stagnation. In a population the genomes can be divided into several groups through genomic distance, called species. Genomic distance approximates the difference between genomes by looking at how far the nodes/ connections have diverged from a common ancestor [3]. When a species does not improve for several generations, then it is considered stagnated and if it's not part of the elite group of species it will be removed from the population. Species are also used to keep diversity in the population by maintaining a minimum number of genomes inside of species group. This causes the population to grow in the earlier generations, due to stagnation not yet happening. Reproduction consists of two methods: crossover and mutation. Crossover creates a new genome by combining two homologous genomes by matching connections with nodes with a common ancestor [3]. Mutation can add/ remove connections and nodes or change the weights and biases applied to the network with a certain probability. The mutation of the weights and biases in this research are achieved through two different methods. The first method applied in the first experiment is known as creep mutation. Creep mutation is mutation by an adding a small value to each bias/ weight [1] with a probability of 0.7. This value is drawn from a 0 mean normal distribution with a variance of 0.5. The second method applied in the second experiment is known as random resetting. Random resetting changes the weight/ bias to a random value [1] drawn from a 0 mean normal distribution with a variance of 0.5 with a probability of 0.25. As NEAT uses a Sigmoid activation function, a 0 mean that the mutated node gets a so called "reset" [3]. A high mutation rate is chosen to make sure the mutations have a significant impact on the evolution. The mutation

rate of random resetting is lower because a high mutation rate tends to become a random search [5]. The probability of mutation can be made adaptive but will remain static for this research, because we are interested in the difference between two mutation methods and not in optimizing the fitness. The same applies to the other parameters except for the population size. Besides mutation probability, the experiments will have the same parameters.

2.4 Generations

The research consists of 20 generations, it is estimated that this is enough for evolution to take place but will keep the total computational time within several hours. The total computational time includes that both experiments will be repeated 10. Repeating the experiment makes it possible to estimate a spread of the results.

3 RESULTS

In this chapter the results of both experiments will be analyzed. Each experiment was tested on three different enemies namely enemy one, two and five. The first plots show the average fitness across the different generations with its upper and lower bound. Also, the maximum fitness of each generation is represented in these plots.

When looking at the maximum in Fig. 1, the difference between both experiments is very small for some generation experiment 1 using Creep Mutation had a higher maximum but for others it was experiment 2 using Random Resetting. Although the difference is very small experiment 1 achieves the highest maximum which is approximately a fitness of 85. When looking at the average fitness over the generation it can clearly be seen that at the fifth generation the average fitness takes off. This is because in the first four generations the population increased because of the stagnation of species which was set to 3 generations. This means that the population was growing and only reached at generation 5 a stable state. From there on experiment 1 had a higher and smoother average fitness then experiment 2.

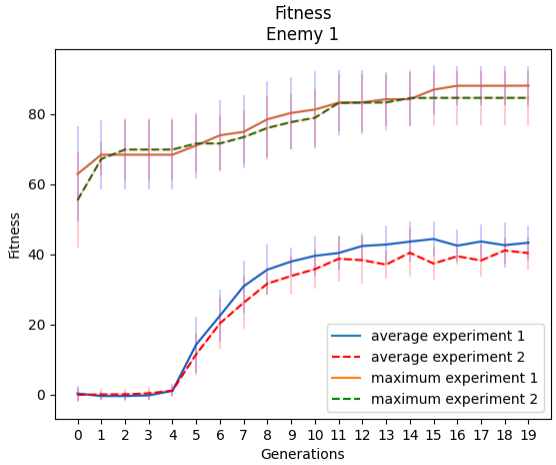

Figure 1: Maximum and average fitness of both experiments for enemy one.

For the second enemy, the fitness is much larger than the first enemy, this means that Evoman performed better against this enemy. This is because the map is different, for enemy one where there are different platforms of different heights this made it more difficult.

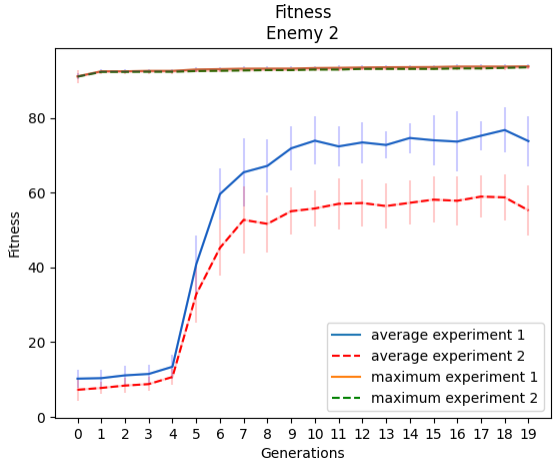

Figure 2: Maximum and average fitness of both experiments for enemy two.

In Fig. 2 the maximum fitness for both experiments are almost similar and are around 90. Since it is constant the upper and lower bound are very small.

When looking at the average fitness it again lifts off at the fifth generation, as it did with enemy one. For the first generations it was relatively stable around a fitness of 10. It then took off to a maximum for experiment one of 75 and for experiment two of 55. Here again experiment one performed better overall than experiment two.

Fig. 3 shows the same pattern as for enemy two, when looking at the maximum fitness across the generations. It is stable at around 90 with a very small upper and lower bound. Also, as for enemy two there is no difference between bot experiments.

For enemy three the average fitness does not lift off as rapidly as for the other two enemies. At generation seven the average fitness begins to increase; it reaches a maximum of around 75 for experiment one and 55 for experiment two. Here again experiment one outperforms experiment two.

When looking at these three plots each time experiment one outperformed experiment two. Therefore, it can be assumed that Creep Mutation performs better than Random Resetting Mutation.

When comparing these results to the results that were found in the paper of Karine da Silva Miras de Araujo. It shows that the results found in this paper had a lower fitness. This can be explained by the fact that the population used in this paper was smaller and therefore could not get to an optimum.

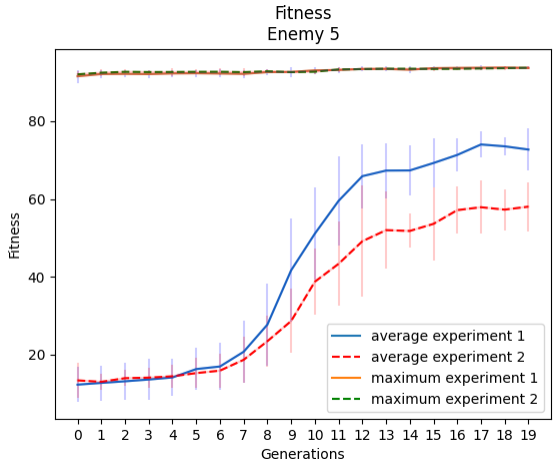

Figure 3: Maximum and average fitness of both experiments for enemy three.

When looking at Fig. 4, that contains the boxplots with individual gain, there is a clear difference between enemy one and the other enemies. The spread of the individual gain for enemy one is a lot bigger than the spread for the other enemies. Which means that enemy two and five almost lost every time, but this was not the case for enemy one. Also the mean of the individual gain is approximately the same and around 90 for enemy two and five but for enemy one it is around 25 for experiment one and negative for experiment two which means that on average the enemy wins the game.

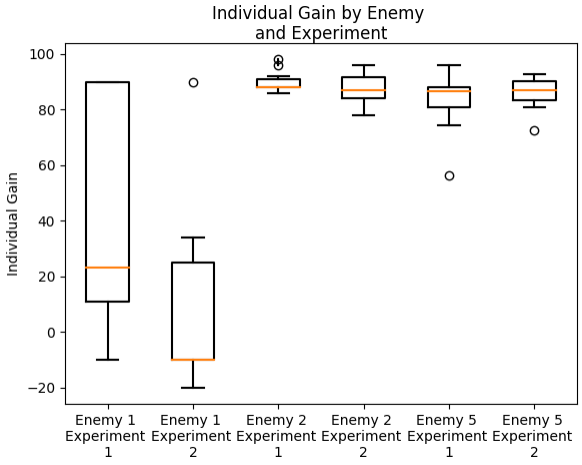

Figure 4: The individual gain per enemy per experiment.

Next a statistical test was used to test if the mean per enemy between the two experiments were equal. When performing the t-test, with a null hypothesis with the assumption that the average means both experiments are equal. Because our p-values were bigger than 0.05, we could not reject the null hypothesis and it could be assumed that the mean per enemy are equal.

Table 1: P-values of T-test

	Enemy 1	Enemy 2	Enemy 5
P-value	0.29867	0.91770	0.94669

In table 2 the average fitness of the experiments is compared to a baseline paper, which is the paper about evolving a generalized strategy for a video game framework written by Karine da Silva and Fabríicio Olivetti [4]. It can be said that the experiments performed similarly to the baseline except for the first enemy. The lower score is mostly caused by the fact that the terrain of the boss fight is not flat, this causes an extra level of complexity which the algorithms used in the experiment aren't manage as efficiently as the algorithm used in the baseline paper.

Table 2: Comparison between Experiments and Baseline Paper

Enemy	Experiment 1	Experiment 2	Multi Evolution Paper [4]
1	88	79	90
2	94	94	94
5	93	93	94

4 CONCLUSIONS

In this paper we researched the difference between Creep Mutation Operators and Random Resetting Operators, we did this using a NeuroEvolution of Augmented Topologies algorithm. We ran our algorithm in a game called Evoman. We ran ten game simulations with Creep Mutation and ten game simulations with Random Resetting, we did this against three different bosses. This means that in total we had 60 simulations, from these 60 we saved the best solution. We rerun each best solution ten times. This means that we had 300 runs, 150 runs with Creep Mutation and 150 runs with Random Resetting, over 3 different bosses. For each boss we performed a T-test, from this we can conclude that there is no difference in fitness between the two mutation operators. We did, however, see a difference in the mean of average fitness's during the runs. Meaning that even though there is no significant difference in the best solution the average of the solution from the runs were higher for the Creep Mutator Operator. This can be beneficial if you are not specifically looking for the optimal solution, but the average solutions are of importance as well.

4 LIMITATIONS AND DISCUSSION

As the simulation duration did not depend on processing power but on the duration of the game, how long the player needed to defeat the boss. we could not do all the experiments that we initially wanted. We would have liked to do runs with different parameters for the mutation power, rate, and probability. But due to time constraints we were not able to. We mostly kept to the standard parameters of NEAT, this could mean that with different parameters different results will occur. However, we are still pleased with our result. As a difference in the average mean of the solution could be interesting if you are for example looking for creativity or just not always want the best solution.

REFERENCES

[1] Eiben, A. E., & Smith, J. E. (2015). Introduction to Evolutionary Computing (2de editie). Springer Medizin Verlag.

[2] K. O. Stanley and R. Miikkulainen, "Efficient evolution of neural network topologies," Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600), Honolulu, HI, USA, 2002, pp. 1757-1762 vol.2, doi: 10.1109/CEC.2002.1004508.

[3] NEAT Overview. (2017, 30 juli). NEAT-Python. https://neat-python.readthedocs.io/en/latest/neat_overview.html

[4] K. da Silva Miras de Araujo and F. O. de Franca, "Evolving a generalized strategy for an action-platformer video game framework," 2016 IEEE Congress on Evolutionary Computation (CEC), Vancouver, BC, 2016, pp. 1303-1310, doi: 10.1109/CEC.2016.7743938.

[5] Hassanat, A.; Almohammadi, K.; Alkafaween, E.; Abunawas, E.; Hammouri, A.; Prasath, V.B.S. Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach. Information 2019, 10, 390.

[6] Jacques sdsadahdasgdhasdhagda.

[7] Bruce P. Douglass. 1998. Statecarts in use: structured analysis and object-orientation. In *Lectures on Embedded Systems*, Grzegorz Rozenberg and Frits W. Vaandrager (Eds.). Lecture Notes in Computer Science, Vol. 1494. Springer-Verlag, London, 368–394. DOI: http://dx.doi.org/10.1007/3-540-65193-429

[8] Ian Editor (Ed.). 2008. *The title of book two* (2nd. ed.). University of Chicago Press, Chicago, Chapter 100. DOI: http://dx.doi.org/10.1007/3-540-09237-4