# Comparing the Generalisability Between a Fixed-Topology Genetic Algorithm and NEAT When Training a Generalist Agent in the EvoMan Framework

**Evolutionary Computing**

**Group 52**

**Task 2: Creating a Generalist Agent**

**19th October 2024**


| Tomasz Kubrak | Rafal Kukielka | Kieran Keesmaat | Jan Kaleta-Slyk |
|---|---|---|---|
| 2745607 | 2816023 | 2843427 | 2828677 |

# 1 INTRODUCTION

Many optimization problems have been tackled with Evolutionary Algorithms (EA) since their development. For example, in the gaming realm, EAs have been applied to various game formats where they have evolved expert game-playing strategies [4]. However, games are becoming more intricate and require these algorithms to generalize to increasingly complex and unseen situations. This is not only crucial in gaming, but also in other AI fields such as robotics or autonomous driving, where there are unpredictable changes in the environment.

This search for generalisable algorithms motivated the current paper, which compared the ability of two NeuroEvolution techniques to evolve generalisable agents within the EvoMan framework, a video game environment designed to challenge agents against eight distinct enemies [2]. A neural network was used to control the agent, where the inputs were game state sensors, and the output was the agent's actions. Both algorithms incorporated NeuroEvolution, which is the process in which the weights and/or topology of a neural network are evolved over time using a fitness function as feedback [6]. The first algorithm was a fixed-topology Genetic Algorithm (GA), only evolving network weights, and the second was a NeuroEvolution of Augmenting Topologies (NEAT), evolving both the weights and network topology.

The fixed-topology GA was chosen as a benchmark for its relative simplicity, computational efficiency, and previous use in neural network optimization [3][5]. NEAT was chosen as previous research had suggested that NEAT outperforms weight-only evolutionary approaches, possibly due to its mechanisms for crossover, incremental growth, and speciation [6]. Our main research question is, does the increased dynamic ability to evolve network topology alongside its weights, as seen in NEAT, lead to better generalisable performance against unseen enemies compared to a fixed-topology GA approach? The main hypothesis is that the NEAT algorithm will be better able to generalize and, therefore, outperform the fixed-topology GA when tested against all enemies.

# 2 METHODS

To test the hypothesis we developed a neural network using the Fixed-Topology GA and the NeuroEvolution of Augmenting Topologies (NEAT) based on the paper by Stanley and Miikkulainen [6]. Each algorithm was trained on two different groups of three enemies over 30 generations and was subsequently tested against all 8 enemies to see how well they generalized. This was repeated for 10 separate runs per algorithm. To find suitable training groups, we tested 56 possible group combinations with varying parameter values using the Fixed-topology GA structure. The two groups that turned out to be most suitable were: (2, 5, 6) and (2, 6, 8). We measured the performance of the agents against each enemy using an individual gain score (agent life – enemy life) and compared the scores between the algorithms. To explore the optimal set of parameters for the algorithms, we performed a hyperparameter tuning with Optuna.

## 2.1 Fixed-Topology Genetic Algorithm

### *2.1.1 Representation and fitness function*

The first algorithm is a fixed topology, fully connected neural network. It consists of 20 input neurons (numbers of sensors in the environment), a single hidden layer with 10 neurons and an output layer with 5 neurons corresponding to the five actions the player can take. Sigmoid activation function is used in both the hidden and the output layers of the network. The genotype of each individual is represented as a vector of real-valued numbers which are the weights of the neural network. They are initialized randomly in the range of [-1,1]. The genotype is then mapped into the phenotype by decoding the vector of weights into the neural network that controls the player in a game. Player's performance against each opponent is evaluated using the following fitness function defined in the paper by Da Silva Miras De Araujo and De Franca [1]. Where $h$ denotes the *health* of player and enemy after each round.

$$fitness = 0.9 * (100 - h_{enemy}) + 0.1 * h_{player} - log_{time}$$

### *2.1.2 Crossover, mutation and selection*

For the crossover mechanism, we examined three types of recombination: *discrete*, *intermediate*, and *blend*. After testing, the *intermediate crossover* approach yielded the best results and was thus selected. In this method, each offspring inherits genetic material as a blend of both parents' genes. For each gene, the offspring's value is calculated as a weighted combination of the corresponding genes from both parents, ensuring that the child's genes fall somewhere between the parents' gene values. A parameter *alpha* is applied to control how much genetic material is inherited from either of the parents. We opted for *tournament selection* to choose parents for recombination by randomly sampling five individuals and choosing the one with the superior fitness score. The new generation retains the fittest individuals based on the *elitism_rate*, with the rest produced through recombination and mutation. The *elitism_rate* defines the percentage of top individuals to preserve in each generation. We chose the *uniform mutation* variant with a mutation probability of 0.05. The selected gene is then replaced by a random value in the range [-1, 1] denoted by *domain_lower/upper_bound*.

| Hyperparameters | Values |
|---|---|
| *n_hidden_neurons* | 10 |
| *Population_size* | 154 |
| *alpha* | 0.12 |
| *tournament_size* | 5 |

| | |
|---|---|
| $p_{mutation}$ | 0.05 |
| *domain_upper_bound* | 1 |
| *domain_lower_bound* | -1 |
| *elitism_rate* | 0.24 |

**Table 1: Fixed-Topology GA parameters**

## 2.2 NEAT

### *2.2.1 Representation and fitness function*

To implement the second algorithm we used the NEAT Python library and based our approach on the paper by Stanley and Miikkulainen [6]. In contrast to the fixed-topology neural network, NEAT allows the network's structure to evolve. Initially, the network begins with a simple configuration and gradually adds neurons and connections between them increasing complexity. The genotype consists of the particular weights and connections between neurons encoded as real-valued numbers, whereas the phenotype is the network that controls the player in the game. Similarly to the fixed-topology GA, the network uses 20 input and 5 output neurons. Sigmoid activation function is used both in the hidden and output layers and the weights are randomly initialized in the range [-1, 1]. This time however, new neurons and connections were added with each new generation in a feed-forward fashion, meaning that there were no recurrent connections. The same fitness function as described in section 2.1.1 was used to allow for clear comparison.

### *2.2.2 Speciation, crossover, mutation and selection*

One of the core features of NEAT is *speciation* which is a process of grouping individuals in the population into species based on their genetic makeup. Speciation is used to let the new network structures evolve independently without being eliminated before they have the chance to influence the evolution. It is achieved by measuring the so-called *compatibility distance* to assess the genetic similarity between neural networks. Individuals with a compatibility distance below the *compatibility_threshold* are grouped into the same species. In other words, members of the same species are genetically similar to each other but differ from individuals in other species. The crossover, selection, and mutation occur within the species rather than the whole population. NEAT uses alignment crossover of two parents' genomes to create the offspring. This approach helps to deal with different sizes and structures of parents' genomes, combining them meaningfully. It involves retaining the matching genes that exist in both parents and have the same innovation number with 0.5 probability of inheriting that gene from one of the parents. The disjoint genes that exist in the middle of the genome of one parent but not the other and the excess genes that also appear in only one of the parents but are located in the end of the genome are inherited based on the fitness of the parents. If the parents have the same fitness, the offspring inherits disjoint and excess genes from both parents. On the other hand, if one parent is fitter than the other, the offspring inherits these genes only from the fitter one. This strategy ensures that the evolving generations produce more complex neural networks retaining the beneficial mutations and structural information of the parents.

There are three types of mutations that we employed in our NEAT algorithm: node, connection, and weight. Node mutation refers to adding and removing neurons to the network structure. New nodes are added by splitting the existing connection between two nodes and inserting a new neuron in between, increasing the network's structure. The frequency of this process is determined by the *node_add/delete_probability*. Connection mutation involves adding and removing connections between existing nodes. It is determined by the rate of *connection_add/delete_probability* parameter. Weight mutation is used to fine-tune the importance of connections between nodes without changing the network's structure. We used two parameters: *weight_mutation_rate* and *weight_mutation_power* to affect the mutation. The former determines the probability that a particular connection will be mutated, whereas the latter controls the magnitude of change during the mutation. Additionally, we implemented *weight_replace_rate* parameter to allow the algorithm for randomly resetting a weight by adding a new random value in its place. This was done to omit local optima.

Fitness based tournament selection was used to choose individuals with higher fitness to reproduce. *Elitism* was set to 2 meaning that the top 2 individuals from each species were carried over to the next generation without any changes. The *survival_threshold* was set to 0.2 ensuring that only the top 20% of individuals inside each species are allowed to reproduce.

| Hyperparameters | Values |
|---|---|
| *n_hidden_neurons* | 10 |
| *Population_size* | 117 |
| *Generations* | 30 |
| *compatibility_threshold* | 1.0 |
| *node_addition/deletion_probability* | 0.375/0.894 |
| *connection_addition/deletion_probability* | 0.895/0.447 |
| *weight_mutation_rate/power* | 0.108/0.983 |
| *weight_replace_rate* | 0.1 |
| *surival_threshold* | 0.2 |
| *elitism* | 2 |

**Table 2: NEAT parameters**

# 3 RESULTS AND DISCUSSION

The average best and mean fitness (over the 10 runs) per generation for each algorithm and training group is plotted in Figures 1 and 2. There is not much difference between the algorithms, with the plots revealing a slightly higher average fitness for the GA compared to NEAT for both training groups. Comparatively, NEAT slightly outperforms GA for the max fitness from training group 2, while their max fitness becomes almost indistinguishable after generation 20 from training group 1.

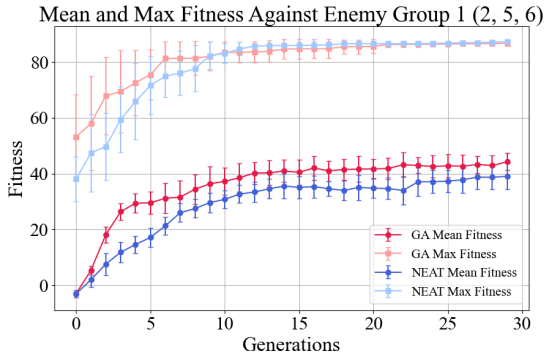


**Figure 1: Average Mean and Max Fitness over 30 Generations**

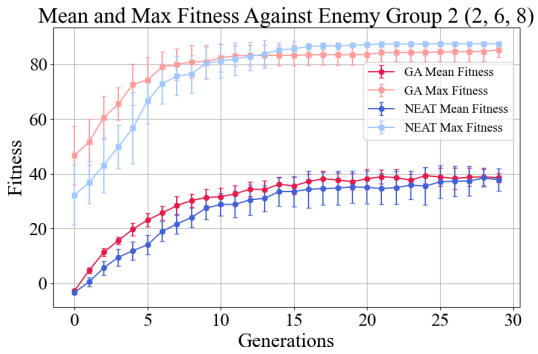


**Figure 2: Average Mean and Max Fitness over 30 Generations**

To investigate the difference in performance, the best individual from each run was tested against all 8 enemies and the average individual gain (agent life – enemy life) was calculated. The boxplots in Figure 3 reveal that for training group 1, the GA achieved significantly higher individual gains (M = -6.04, SD = 4.77) compared to NEAT (M = -19.36, SD = 9.2) as revealed by Mann-Whitney U-test (U-statistic = 7, $p < 0.001$). Furthermore, Figure 4 shows that for training group 2, the GA once again outperformed NEAT, (M = -10.68, SD = 9.02 and M = -15.45, SD = 10.32 respectively) however this result was not found to be significant using a Mann-Whitney U-test (U-statistic = 38, p = 0.25).

Interestingly, although the GA had better performance on average, Table 3 reveals that NEAT produced the very best overall individual, with an average individual gain over all enemies of 4.4, compared to 2.8 from the best GA individual. This appears to be more of an outlier, however, and does not offset the overall trend.

Contrary to the hypothesis, these results show that the fixed-topology GA was better able to generalize compared to NEAT. A potential reason for this may have been due to the way in which training groups were selected for. The GA architecture was used to find groups that could generalize well and these groups were then also used for both the GA and NEAT when training the final generalist agent. In hindsight, it now appears clear that this could have added a bias that inherently favored the GA as it was in a sense pre-tuned to generalize well based on these training groups. In the future, a more careful approach to training group selection should be taken to ensure that there is no unfair advantage to one algorithm over the other. This result further highlights the difficulty of objectively comparing algorithms as already noted in the literature [1].

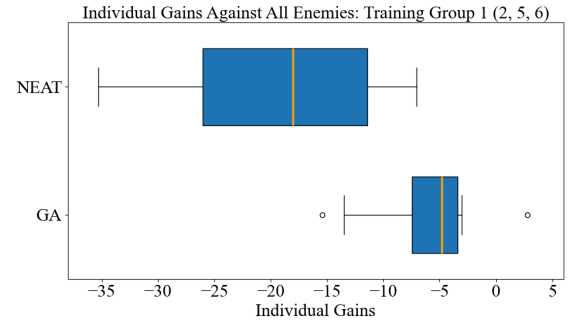


**Figure 3: Individual gains of GA and NEAT against all enemies**

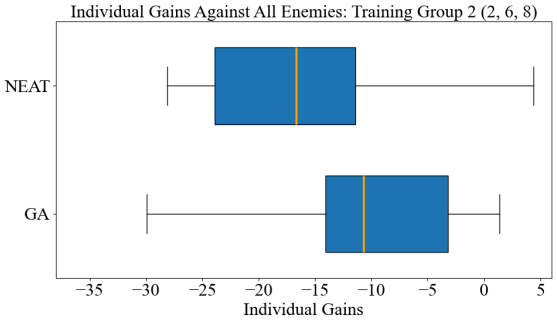


**Figure 4: Individual gains of GA and NEAT against all enemies**

| Enemy nr: | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|
| NEAT | h_player | 0 | 62 | 0 | 0 | 61 | 21.4 | 0.4 | 30.4 | 21.9 |
| | h_enemy | 90 | 0 | 30 | 20 | 0 | 0 | 0 | 0 | 17.5 |
| | gain | -90 | 62 | -30 | -20 | 61 | 21.4 | 0.4 | 30.4 | 4.4 |
| GA | h_player | 0 | 68 | 0 | 0 | 47.2 | 55.6 | 0 | 11.2 | 22.8 |
| | h_enemy | 90 | 0 | 30 | 20 | 0 | 0 | 20 | 0 | 20 |
| | gain | -90 | 68 | -30 | -20 | 47.2 | 55.6 | -20 | 11.2 | 2.8 |

**Table 3: Energy points of agent and enemy for best performing NEAT and GA**

## 4 CONCLUSIONS

In this project, we compared the performance of a Fixed-Topology GA with a NEAT algorithm for evolving generalist agents in the EvoMan framework. Contrary to expectations, the GA showed a better ability to generalize to new enemies compared to NEAT. A potential reason for this is introduced bias when selecting training groups, which future research is invited to investigate further. This paper brings to light the importance and difficulty in designing unbiased experimental comparative studies, and hopefully raises awareness of this critical challenge for future researchers in the field.

## REFERENCES


[1] Beiranvand, V., Hare, W., & Lucet, Y. (2017). Best practices for comparing optimization algorithms. doi:http://dx.doi.org/10.14288/1.0379904

[2] de Araújo, K. da S. M., & de França, F. O. (2016). An electronic-game framework for evaluating coevolutionary algorithms. https://doi.org/10.48550/arxiv.1604.00644

[3] Gupta, J. N. D., & Sexton, R. S. (1999). Comparing backpropagation with a genetic algorithm for neural network training. Omega (Oxford), 27(6), 679–684. https://doi.org/10.1016/S0305-0483(99)00027-4

[4] Lucas, S. M., & Kendall, G. (2006). Evolutionary computation and games. In IEEE computational intelligence magazine (Vol. 1, Number 1, pp. 10–18). IEEE. https://doi.org/10.1109/MCI.2006.1597057

[5] Sexton, Randall & Dorsey, Robert. (1998). Toward global optimization of neural networks: A comparison of the genetic algorithm and backpropagation. Decision Support Systems. 22,. 171-185. 10.1016/S0167-9236(97)00040-7.

[6] Stanley, K. O., & Miikkulainen, R. (2002). Evolving Neural Networks through Augmenting Topologies. Evolutionary Computation, 10(2), 99–127. https://doi.org/10.1162/106365602320169811