Evolutionary Computing

Task 1: specialist agent - Team 109/Team Inspector Gadget - October 1, 2021

Tommaso Castiglione Ferrari
Vrije Universiteit
Amsterdam
2673807

Tim de Boer
Vrije Universiteit
Amsterdam
2599785

Marcel Stelte
Vrije Universiteit
Amsterdam
2735128

Sander Brinkhuijsen
Vrije Universiteit
Amsterdam
2685424

,
.

1 INTRODUCTION

Evolutionary algorithms (EAs) is the name for a family of population-based search algorithms which, inspired by biological evolution, embrace the notion of natural selection in which individuals with the most favourable traits survive and pass on their genes to further generations.

Games have provided the research field of EAs with a large variety of environments, each with their own difficulties, in which the performance of different algorithms can be assessed and compared [15]. The games included in EA research range from traditional board games like chess [8], battleship [10] and Go [14], to video games like Flappy Bird [1], racing simulators [2], and a variety of Atari games [9]. In 2019, de França et al. [5] introduced the EvoMan competition. They provided a framework for a 2-dimensional action platformer that is based on the video game Mega Man 2. The player controls a character that is able to move left and right, jump, interrupt and shoot straight forward with its arm cannon. The player is ought to defeat all eight different bosses, where each of the bosses has its own unique set of attack moves. Both the player and the bosses start a fight with 100 energy points and either party loses when energy points are reduced to zero. Energy points are lost when getting hit by an attack move from the enemy or when the players collide. The goal of competition is to evolve a controller for the player character that is able to defeat the enemy.

An artificial neural network (ANN) can be used to function as the controller of the player character in the EvoMan framework. The topology and the weights of connections between neurons determine the behaviour of the agent, where the goal is to find the optimal settings that allow the agent to thrive in the environment by winning a specific boss fight. The ANN cannot be easily trained with methods like backpropagation, mainly because of sparse moments of feedback [3, 11]. EAs are known to work well with sparse feedback moments, thus, an alternative way to optimize the ANN is through EAs [7] . This specific subfield of EAs is called Neuroevolution [4].

The goal of this study is to investigate the difference in performance in the EvoMan environment between training the ANN of an agent using the standard Genetic Algorithm (GA), and an agent trained with the NeuroEvolution of Augmenting Topologies (NEAT) algorithm, where both agents will be evaluated using a conservative fitness function. The literature does agree that NEAT is a more advanced version of the standard GA [16, 17], and the baseline values for EvoMan from NEAT show better results when compared with GA [5]. Thus, it is hypothesized that NEAT will outperform GA. Both algorithms will be implemented to evolve a specialized agent controller for the EvoMan framework. Here, specialized indicates that an agent will only be trained to fight one specific boss at the time, and will be retrained for the next boss. A total of three different bosses will be selected and the performance of the two algorithms will be compared per boss.

2 METHODS

Eiben et al. [7] introduce a general procedure most EAs abide to. Firstly, a randomly initialized population is used as first generation. Then, all individuals are let loose in an environment and their performance is evaluated based on a fitness function. Some of the best performing individuals are selected as parents to generate offsprings for the second generation. Children are created by mutating a single parent or combining elements of two parents. The process of evaluating fitness and replacing individuals is repeated until a stopping criteria is reached, for example, max number of generations or no improvement in fitness scores. In the following sections, the specific parameters per algorithm of this procedure will be discussed.

2.1 Genetic Algorithm

One of the most basic Neuroevolution algorithms is the standard Genetic Algorithm (GA) [12]. The GA randomly initializes an ANN structure, the structure used in this experiment has one hidden layer consisting of 10 hidden neurons. The Sigmoid function was used for all relevant layers to determine activation. The algorithm then evolves the weights of the network by creating new generations according to the given parameters. The parameters for selection, crossover, mutation and replacement were chosen by experimentation, and are as follows:

- Parent selection: Tournament selection (n=2)
- Population management: Steady-state model with generation gap of $\frac{148}{150}$. By only keeping 2 parents in the parent selection phase, the other parents are replaced by the offspring.
- Crossover: One-point crossover with $p_c = 0.4$ of copying parent instead of the offspring.
- Mutation: Random mutation with $p_m = 0.4$ where 10% of the weights will be altered through random replacement.

It is expected that by more experimentation, the parameters can be further optimized. However, due to the limited time budget, this was not possible for this project.

2.2 NeuroEvolution of Augmenting Topologies

The main idea of the NEAT algorithm is to evolve the ANN's weight values and topology at the same time, and therefore, no longer requiring a fixed topology. This method also prioritizes simpler topologies over complex ones, which might be helpful for efficiency reasons.

The structures start as simple as possible and build complexity through evolution. The network starts with two lists: node genes and connection genes. The node genes provide individual nodes an identifier. The connection genes specify the structure of the network, for example, it denotes what nodes are connected and the weight of the connection. Just like GA, the NEAT algorithm also uses crossover and mutation. However, both operators work differently due to the flexible structure of the network. NEAT's crossover operation relies on global innovation numbers, which is provided to all genes in the connection genes list and allow the chronology of adaptations to be tracked. The connection genes of two parent networks are aligned according to these innovation numbers. A child inherits all matching genes. The genes of the fittest parent are selected in the case of mismatching genes, and a random gene is selected in the event of equal fitness scores. The mutation operator can mutate existing weights or increase complexity of an ANN by adding new node in between two connected nodes or a connection between two existing, unconnected, nodes. Changing the topology

of an ANN can be problematic because the new structure might not immediately result in a better controller. To prevent more complex networks from going extinct right after their creation, NEAT divides its population into different species and only compares controllers from the same species.

NEAT requires a large number of different parameters to be set. The configuration values used were obtained from Drallensmith [6], which experiments with OpenAI's LunarLander environment, a similar experiment, and thus ideal to serve as a baseline for our NEAT parameters.

2.3 Features

The 20 features provided by the EvoMan framework were used as input for the ANNs: 1 for the direction the agent is facing, 1 for the direction the enemy is facing, 2 for the absolute vertical and horizontal distance between the characters, and 16 for the absolute horizontal and vertical distance between the player character and the projectiles.

2.4 Fitness function

The fitness function determines the performance of a controller in the environment. The fitness function used in our experiments is inspired by the fitness function from [4] and can be found in Equation 1, where pe and ee are the player and enemy energy respectively, and α, γ and β are discount factors.

$$F = (100 - ee)^{\gamma} - (100 - pe)^{\beta} - ((\sum_{i=1}^{(t)} 100 - pe_i)/t)^{\alpha} \quad (1)$$

As the authors have experimentally found the optimal values of 1, 2 and 2 for γ, β and α respectively [4], these values were implemented here as well. Also, setting the discount factors to these specific values promote the creation of defensive agents, since taking damage receives a higher penalty than damaging the enemy. The resulting fitness value is then normalized between 0 and 100 for visualization purposes.

2.5 Experimental setup

In this experiment, we assess the performance of both the standard GA and NEAT as specialist agents when fighting the EvoMan bosses AirMan (2), MetalMan (5) and QuickMan (8). Both algorithms maintain a population size of 150 agents throughout the experiments. The evolution cycles stops after reaching 20 generations, as experimentation showed that more generations were not significantly beneficial for the fitness scores.

The experiments consist of 10 independent runs for both algorithms for each of the selected bosses. A line plot is created to shows the average performance per generation by the mean, maximum and their respective standard deviation of the fitness score. Furthermore, the best controllers per run were selected, and used to fight their boss 5 more times. Their average fitness scores is captured in a boxplot. Finally, the best solutions were subjected to a statistical test that will show whether the difference in performance is statistically significant. As this is a paired-sample situation where the distribution of the retrieved solutions is unknown, the paired-sample Wilcoxon test was used.

3 RESULTS

The line plots of the experiments can be found in Figure 1. Both algorithms produce agents that are generally able to play well against all agents, as shown by the average maximum fitness values (red line) being close to the highest obtainable fitness score, 100. Furthermore, NEAT consistently outperforms GA, especially against boss two and eight. In fact, both considering the average of the maximum fitness values as well as the average of the mean fitness values there is strong evidence of NEAT's dominance over GA. Moreover, not only the mean fitness values are higher, but they also have a lower standard deviation.

Next, Figure 2 shows the boxplots of both algorithms per boss [1], and table 1 shows the p-values of the paired-sample Wilcoxon test per boss. By analyzing the p-values obtained, we can verify that there is a significant difference in performance between NEAT and GA for the second and eight bosses. Conversely, the p-value for the fifth enemy does not show a significant difference in performance.

Enemy	GA	NEAT	Wilcoxon p
2 (AirMan)	97.562	**99.708**	0.004
5 (MetalMan)	97.600	**98.868**	0.557
8 (QuickMan)	94.429	**98.200**	0.01

Table 1: Mean fitness and statistical values of 5 runs from the best individuals

4 DISCUSSION

The results show that, while the best individuals of both NEAT and GA are capable of defeating all enemies, NEAT consistently outperforms the standard GA technique, expected outcome based on previous research [3–5]. This effect can be explained by the flexibility of NEAT, having a non-fixed topology enables the algorithm to find a non-obvious structures, as well as having a higher magnitude of evolution power. Conversely, GA has to fully rely on the predefined structure, which might have a sub-optimal topology for this specific task.

As discussed in Section 2.4, the fitness function incited controllers to focus on survival rather than defeating the enemy. It was noticed that some agents were able to obtain a high fitness score while having a low individual gain [2] due to not inducing damage to the enemy, while avoiding taking damage at all costs. Manual inspection of controller behaviour showed that the player agent can get stuck in a sequence of damage avoiding actions. The sequence would be repeated until the maximum number of game ticks is reached, leaving both the player and the boss at maximum energy, thus reaching a high fitness score. The fitness function can be modified to include a time penalty, which would incentive the agent to defeat the enemy instead of just surviving it. However, this might compromise the conservative nature of the agent, causing the agent to take more damage as well.

Due to the time constraints for this project, selection of hyperparameters for both GA and NEAT was sub-optimal. An interesting

[1] The vertical axis of the boxplots have been scaled according to the min and max values for visualization purposes

[2] Individual Gain = player energy - enemy energy

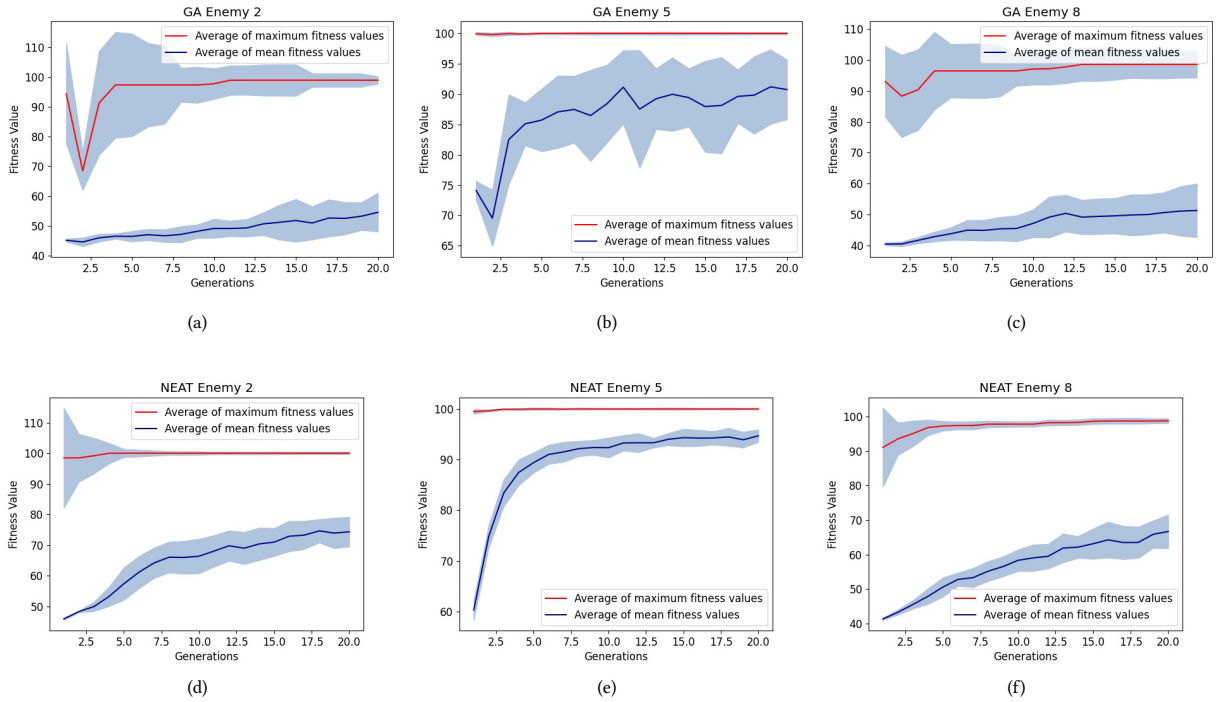

Figure 1: Lineplots of mean and max fitness values of individuals over generations for GA (top row) and NEAT (bottom row), against enemy 2 (left), 5 (middle), and 8 (right).

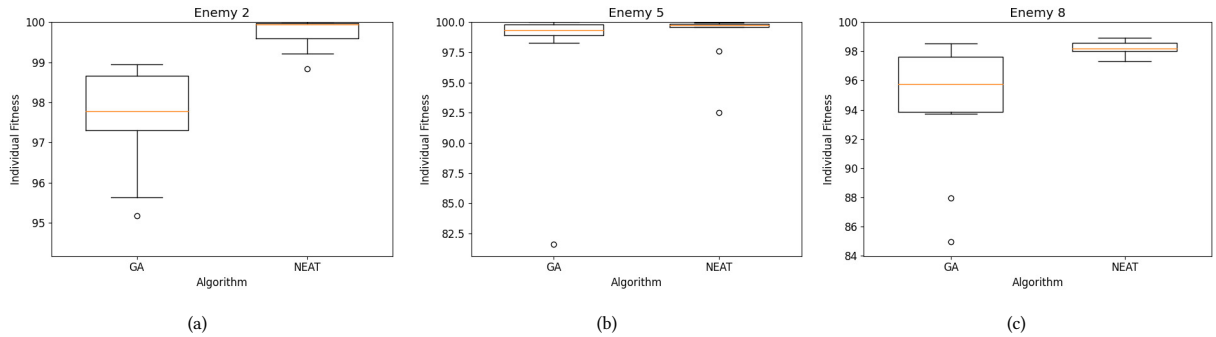

Figure 2: Boxplots of fitness for the best individual per run for GA and NEAT, against enemy 2 (left), 5 (middle), and 8 (right).

topic for future research could be to apply hyper-parameter optimization techniques like F-Race, Revac and ParamILS [13] for fine-tuning the two techniques' parameters. Moreover, considering different fitness functions that would still focus on the player's health as first objective, while also keep in consideration and penalize the time passed, could prove potentially beneficial on the overall performance.

5 CONCLUSIONS

This paper investigated the performance of the standard GA with the NEAT algorithm for the EvoMan framework. In general, the adapting nature of NEAT outperforms the fixed structure of standard GA. The conservative fitness function enabled both algorithms to evolve agents that were capable of evading the enemy, however, this occasionally prevented the agent from damaging the enemy. It is important to find a fitness function that balances avoiding and doing damage.

6 TASK DIVISION

- Tommaso and Marcel: implementation of algorithms.
- Tim and Sander: literature research and report.

REFERENCES

[1] André Brandão, Pedro Pires, and Petia Georgieva. 2019. Reinforcement Learning and Neuroevolution in Flappy Bird Game. In *Pattern Recognition and Image Analysis*, Aythami Morales, Julian Fierrez, José Salvador Sánchez, and Bernardete Ribeiro (Eds.). Springer International Publishing, Cham, 225–236.

[2] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. 2009. Evolving Competitive Car Controllers for Racing Games with Neuroevolution *(GECCO '09)*. Association for Computing Machinery, New York, NY, USA, 1179–1186. https://doi.org/10.1145/1569901.1570060

[3] Karine da Silva Miras de Araujo and Fabrício Olivetti de Franca. 2016. Evolving a generalized strategy for an action-platformer video game framework. In *2016 IEEE Congress on Evolutionary Computation (CEC)*. 1303–1310. https://doi.org/10.1109/CEC.2016.7743938

[4] Karine da Silva Miras de Araújo and Fabrício Olivetti de França. 2016. An electronic-game framework for evaluating coevolutionary algorithms. *arXiv preprint arXiv:1604.00644* (2016).

[5] Fabrício Olivetti de França, Denis G. Fantinato, Karine Miras, A. E. Eiben, and Patrícia Amâncio Vargas. 2019. EvoMan: Game-playing Competition. *CoRR* abs/1912.10445 (2019). arXiv:1912.10445 http://arxiv.org/abs/1912.10445

[6] Drallensmith. 2019. neat-python-openai-lander. https://github.com/drallensmith/neat-python.

[7] Agoston E Eiben, James E Smith, et al. 2003. *Introduction to evolutionary computing*. Vol. 53. Springer.

[8] D.B. Fogel, T.J. Hays, S.L. Hahn, and J. Quon. 2004. A self-learning evolutionary chess program. *Proc. IEEE* 92, 12 (2004), 1947–1954. https://doi.org/10.1109/JPROC.2004.837633

[9] Matthew Hausknecht, Joel Lehman, Risto Miikkulainen, and Peter Stone. 2014. A Neuroevolution Approach to General Atari Game Playing. *IEEE Transactions on Computational Intelligence and AI in Games* 6, 4 (2014), 355–366. https://doi.org/10.1109/TCIAIG.2013.2294713

[10] Tomáš Kancko. [n.d.]. Reinforcement Learning for the Game of Battleship. ([n. d.]).

[11] Joel Lehman and Risto Miikkulainen. 2013. Neuroevolution. *Scholarpedia* 8, 6 (2013), 30977.

[12] Tom V Mathew. 2012. Genetic algorithm. *Report submitted at IIT Bombay* (2012).

[13] Elizabeth Montero, María-Cristina Riff, and Bertrand Neveu. 2010. An evaluation of off-line calibration techniques for evolutionary algorithms. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. 299–300.

[14] Norman Richards, David E. Moriarty, and Risto Miikkulainen. 1998. *Applied Intelligence* 8, 1 (1998), 85–96. https://doi.org/10.1023/a:1008224732364

[15] Sebastian Risi and Julian Togelius. 2017. Neuroevolution in Games: State of the Art and Open Challenges. *IEEE Transactions on Computational Intelligence and AI in Games* 9, 1 (2017), 25–41. https://doi.org/10.1109/TCIAIG.2015.2494596

[16] Kenneth O Stanley and Risto Miikkulainen. 2002. Efficient evolution of neural network topologies. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, Vol. 2. IEEE, 1757–1762.

[17] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation* 10, 2 (2002), 99–127. https://doi.org/10.1162/106365602320169811