Generalist Agent for Evoman Framework. Neuroevolution Of Augmented Topologies

Evolutionary Computing- Group 21

Wafaa Aljbawi
Victor Retamal Guiberteau
Janneke van Baden
Félix Nastar

1 INTRODUCTION

Computational Intelligence can be used to design and develop Artificial Agents (AA's) with biological inspiration to perform predefined tasks. An example of such a task which is playing a videogame. Evoman game is part of a framework specifically created to test Evolutionary Algorithms (EA's). The Evoman framework provides us with 8 different challenges represented by eight different enemies in the game [2]. AA will take the control of the player and will compete with every enemy in a non-static environment.

Neuroevolution makes it possible to find a neural network that optimizes our objective with the generalist agent by evolving weights and/or the architecture of a neural network.

Our objective is to balance the minimization in the energy of the adversary in the EvoMan game, the maximization in the energy of the current player and the minimization in the time taken to end the game to generate an optimal generalist agent to beat the Evoman game.

In multi-objective problems as the one stated before, the main problem is how to develop an AA to generalise a behaviour for challenges for which it is not specifically trained for. This research investigates the generalisation of two EA's, shown in Table 1, when trained to perform neuroevolution in the Evoman framework. The two EA's will be tested against a specific baseline paper from [3]. A comparison between the two EA's will be explained.

1.1 Research question

In this research paper, we aim to answer the following research question:

- *Which neuroevolution strategy between, evolving weights with genetic algorithm (DEAP) and neuroevolution of augmented topologies (NEAT), is best suitable to produce a optimized solution for a generalist agent in the Evoman framework?*

The findings of experiments performed to answer this question are presented in Section 3.

2 METHODOLOGY

2.1 Training and testing phase

In the training phase 25 generations, with a population with 100 individuals, were used per run. Two groups of enemies were selected to perform the training of the EA's. Group one with enemies [7,8] and group 2 with enemies[2,7,8]. We selected the enemy group based on the enemy player attack pattern and the differences in the environment. The patterns and environments are as described in [2]. For each group of training enemies, experiments conducted and repeated 10 times. The statistics from these experiments are reported. Using the best individuals of each run, determined by fitness, the testing phase was performed. This consisted of five individual runs of each best solution, of which the mean gain is reported.

2.2 Design of Evolutionary Algorithms

Two evolutionary algorithms, one using DEAP [5] and one using NEAT [1] were implemented for this paper. A sketch of the algorithms can be found in Table 1, and its numeric parameters in Table 2 and Table 3.

Parameter	Algorithm 1	Algorithm 2
Representation	Weights of a neural network	NEAT encoding
Crossover	Blend Alpha Crossover (+limit)	Competing Conventions (NEAT version)
Mutation	Uniform Mutation	Add Node, Add Connection
Parent Selection	Roulette Wheel Selection	Tournament
Survival Selection	μ, λ	Elitism

Table 1: Symbolic Parameters

Parameter	EA1
Number of hidden neurons	10
Population size	100
Parent pool size	400
Offspring per parent pair	2
Crossover parameters	$\alpha = 0.5$ $prob = 0.5$
Mutation probability	0.25

Table 2: Numeric Parameters EA1

Parameter	EA2
Population size	100
Compatibility threshold	3
Excess coefficient	0.1
Disjoints coefficient	1
Weight mutation prob	0.1
Probability to add new node	0.2
Probability to add new connection	0.5%

Table 3: Numeric Parameters EA2

2.2.1 *Fitness Function.*

The fitness an individual has, represents how well it satisfies the criteria the algorithm is optimizing for [6]. The following equation shows how the fitness is calculated for one game [3]:

$$fitness = \gamma * (100 - e_e) + \alpha * e_p - logt$$

where

- e_e *and* e_p *are the energy of enemy and player, respectively*
- t *is the number of time steps the game took*
- γ *and* α *are constants assuming values 0.9 and 0.1, respectively*

There are some adjustments if multiple enemies are used to test an individual. The results of the games are then such that: $\overline{f} - \sigma$, in which $\overline{f}$ stands for the mean of the fitnesses of the games (which were played with different enemies) and σ stands for the standard deviation [3].

2.2.2 *Algorithm 1: Genetic Algorithm.*

The DEAP framework was used to implement the first algorithm. DEAP is a well-designed EC library that provides useful tools for

a rapid development of unique evolutionary algorithms (EA) [5]. In comparison to the other black-box frameworks, DEAP is simple to use and perfect for making EAs explicit and data structures transparent [4].

The initial individuals are represented by an array consisting of random uniform numbers within the bounds [-1,1], where each number corresponds to a weight value from the "standard neural network" that contains 20 input nodes, 10 hidden nodes (in a single layer), and 5 output nodes. To initialize an individual, a "*create_gene*" function is called n = *n_vars* times repeatedly using "tools.initRepeat",where *n_vars* are the number of weights from the previously mentioned neural network.

Similarly, a population alias is defined to allow population initialization, in this case with n = 100 individuals. Then, the fitness of each individual in the population is evaluated and stored.

The evolutionary uses fitness-proportionate selection to choose parents. In this algorithm, roulette-wheel selection is used. The number of selected parents is four times the population size. Crossover and/or mutation can be performed on these parents, to generate offspring.

For the operators changing the parents to be offspring, uniform mutation happens to an offspring individual with probability 0.25, and in that individual, to a gene with probability 0.1, and blend–α crossover happens to an offspring individual with probability 0.5. As using the blend–α method can result in weights outside of the [-1,1] bounds, weights that are outside the bounds get changed to the closest bound, so either -1 or 1.

For all the changed individuals, the fitness is recalculated. Afterwards, the (μ, λ) strategy is used to select which individuals survive and form the next generation. The parents never survive. Of the $populationsize * 4$ offspring, only the best "$k = population\_size$" can survive, and go on to be the next generation. Therefore, an element of elitism is introduced, so it keeps the best individuals and it is less likely to get stuck in an optimum.

Using the Hall of Fame functions in DEAP, the best individual, based on fitness, of the entire run gets stored and always kept. This way, even if the individual gets lost due to the (μ, λ) survivor selection, it gets stored and kept.

2.2.3 Algorithm 2: NEAT.

NEAT is an Evolutionary algorithm used to evolve the best topology and corresponding weights of a neural network for a specific task [7]. It works by altering both the weights and structure of the network to find a balance between the fitness of evolved solutions and their diversity. NEAT encoding is based on node tuples (A,B) with the information flowing from A to B and a set of flags indicating the state of the connection. An Identifying Number per chromosome is added to track innovation between species. The mutation operator in NEAT can add a node and/or new connections. The crossover operator creates a neural network with the architecture of the fittest parent and the weight inherited from both parents. With every generation more complex architecture are generated, but this new complex structures will have a low fitness in their first appearance. To solve this problem NEAT tracks genes with history markers to allow crossover among topologies, applying speciation to preserve innovations, and developing topologies incrementally from simple initial structures. This distance between two network encoding δ can be measured as a linear combination.

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 * W \tag{1}$$

Where E is the number of excess genes, D disjoint genes and W is the average weight differences of matching genes.

The reproduction mechanism of NEAT is based in explicit fitness sharing, preventing an unique species to take over the evolution even if every member of the species perform extremely well. The adjusted fitness $f_i^{'}$ for organism i is calculated according to its distance δ from every other organism j in the population

$$f_i^{'} = \frac{f_i}{\sum_{j=1}^{n} sh(\delta(i,j))} \tag{2}$$

Where

$$sh(\delta(i,j)) = \begin{cases} 0, & \text{if} \delta(i,j) > \delta_t \\ 1, & \text{otherwise} \end{cases}$$

and δ_t is a threshold to cluster compatibility. Each species is assign an specific number of off-springs based on the sum of the adjusted fitness $f_i^{'}$ of their members. The total number of off-springs n_k allotted to species k is:

$$n_k = \frac{\bar{F}_k}{\bar{F}_{tot}} |P| \tag{3}$$

Where $\bar{F}_k$ is the average fitness of population $|P|$ and $\bar{F}_{tot} = \sum_k \bar{F}_k$

The lowest performing fraction of each species is eliminated. The parents for next generation are chosen randomly among the remaining individuals (uniform distribution with replacement). The highest performing individual in each species carries over from each generation. Otherwise the next generation completely replaces the one before [7]. To implement our version of NEAT, we utilized NEAT-python [1], a framework based in [7] to implement a version of NEAT.

3 RESULTS AND DISCUSSION

In Figures 1 and 2, the results of the training for an enemy group consisting of enemies 7 and 8, and the results of the training for an enemy group consisting of enemies 2, 7 and 8 are given respectively.

It can be seen that both the mean and the best fitness of the evolutionary algorithm using DEAP are higher than the mean and the best fitness using the NEAT algorithm. Moreover, the best fitness of the NEAT algorithm seems to have multiple peaks, whereas the best fitness of the algorithm using DEAP steadily increases.

An explanation for this is that the DEAP algorithm uses (μ, λ) selection, and thus has an element of elitism. It always retains the best offspring solutions, which can also contain non-mutated individuals that have not gone through recombination. Due to its fitness-proportionate parent selection, this probability is also higher for the individuals which have a higher fitness.

The mean fitness therefore increases rapidly for EA type 1, but it seems that the mean fitness only starts to increase steadily for EA type 2 at the end of the 25 generation run. For further research, it may be interesting to compare these two algorithms with runs with more generations. However, due to limited computational power, this could not be done for this paper.

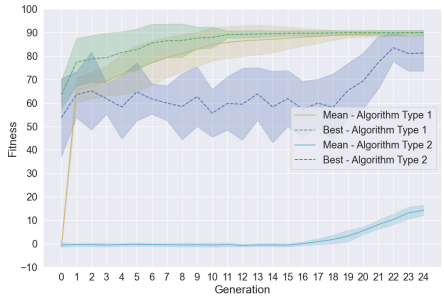

Figure 1: Performance of EA by Type, for Enemy 7 and 8

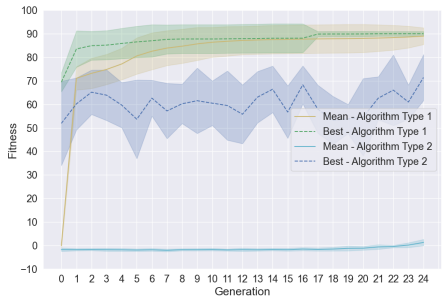

Figure 2: Performance of EA by Type, for Enemy 2, 7 and 8

Figure 3 shows box-plots of the gain of the best solution of every run. This best solution was chosen by fitness.

In the box-plots, it can be seen that the median and the 3rd quartile of the mean gain of EA2 lie below the 1st quartile of the mean gain of EA1. This seems to indicate that the performance of EA1 is better than the performance of EA2.

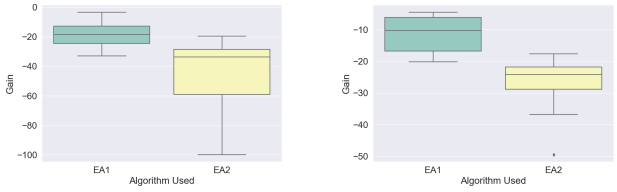

Figure 3: From left to right: Performance of Best Solutions for Training With Enemies (7, 8), and (2, 7 and 8)

To confirm this, significance tests need to be performed. Firstly, it needs to be determined whether the data in the box-plots is normally distributed. Therefore, a Shapiro-Wilk test for normality has been performed. In the cases both the distributions for EA1 and EA2 for a particular enemy group were normal, an unpaired t-test was performed. Otherwise, a Mann-Whitney U test was performed. The results are given in Table 5.

The significance tests in Table 5 show that there is indeed a significant difference between the performance of EA1 and the performance of EA2. EA1 has a higher gain than EA2, for both enemy groups.

Table 4: Results Shapiro-Wilk Test for Normality and Significance Test

Enemy	p-value Shapiro	Test	p-value
7, 8	EA1 - 0.97| EA2 - 0.10	Unpaired t-test	0.01
2, 7, 8	EA1 - 0.16 | EA2 - 0.03	Mann-Whitney U	0.00

Comparison with the baseline paper can be done by taking the best solution from the baseline paper NEAT algorithm trained with (1,5,6) and comparing it with the best solutions generated by EA1 and EA2 shown in table 5. The baseline NEAT version outperforms our versions. This difference could be due the optimization and parameter tuning utilized in the baseline paper with LinkedOpt and the different selection of enemies groups to train the algorithm.

Table 5: Best solution comparison with NEAT baseline paper best solution

Gain±std Baseline NEAT	Gain±std EA1	Gain±std EA2
33±9	-11±6	-27±10

Lastly, the best individual was chosen, based on the mean fitness found in the experiment results that were shown in the box-plots. This individual was created by the DEAP EA and trained on an enemy group consisting of enemy 2, 7 and 8. The mean of the player and enemy energy of five games for every individual enemy is shown in Table 6. This shows that the solution is able to beat 4 of the 8 enemies.

Table 6: Mean Player and Enemy Energy for Best Solution

Enemy	1	2	3	4	5	6	7	8
MEP	0.0	64.0	0.0	0.0	45.4	0.0	24.9	27.0
MEE	78.0	0.0	56.0	70.0	0.0	22.0	36.0	6.0

4 CONCLUSIONS

Our research shows that evolving the weights of the neural network is a better approach for neuroevolution in multi-objective problems when comparing it with NEAT algorithm for the implementations described in this paper. The genetic algorithm has less parameters to tune than the algorithm using the NEAT framework, which could be an explanation of why it outperforms it.

However, as shown in the baseline paper, further research can be done regarding parameter tuning and enemies groups selection to optimize the performance of NEAT.

5 CONTRIBUTIONS

Félix worked on the NEAT algorithm, its report part and conducted experiments. Wafaa implemented DEAP algorithm, wrote methodology in the report, conducted experiments. Victor implemented NEAT and wrote the report. Janneke implemented DEAP algorithm, created plots, performed statistical analysis and wrote the discussion.

REFERENCES

[1] LLC CodeReclaimers. 2018. NEAT-Python Documentation. (2018).

[2] 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).

[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)*. IEEE, 1303–1310.

[4] François-Michel De Rainville, Félix-Antoine Fortin, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. Deap: A python framework for evolutionary algorithms. In *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*. 85–92.

[5] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary algorithms made easy. *The Journal of Machine Learning Research* 13, 1 (2012), 2171–2175.

[6] Haneet Kour, Parul Sharma, and Pawanesh Abrol. 2015. Analysis of fitness function in genetic algorithms. *International Journal of Scientific and Technical Advancements* 1, 3 (2015), 87–89.

[7] K.O. Stanley and R. Miikkulainen. 2002. Efficient evolution of neural network topologies. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, Vol. 2. 1757–1762 vol.2. https://doi.org/10.1109/CEC.2002.1004508