Evolutionary Computing

Task 2: generalist agent

Group 14
Alexander Uitendaal, 11027835
Milou Fuhri Snethlage, 10770747
Bart van Laatum, 11027835
Kas Sanderink, 10814418
02-10-2019

ABSTRACT

To research the application of Evolutionary Computing (EC) in an Electronic Gaming context, two Evolutionary Algorithms (EA's) are implemented. he EA's are used to evolve a neural network which controls an agent to compete in a Megaman-based EC-oriented framework named Evoman. Evoman contains eight enemies which must be defeated by one neural network that is evolved by an EA. The neural network is trained against two of the eight enemies. There is a significant differences between the performance of the EA's used in this report. Furthermore, are both EA's outperformed by the baseline paper.

1 INTRODUCTION

Computational Intelligence can be used to train Artificial Agents (AA's) to perform some predefined task [1]. An example of such a task is playing a video game. In this paper, we will train an AA to play a specific game called Evoman. This game is part of a framework that was created specifically to test and improve Evolutionary Algorithms (EA's). The Evoman framework provides 8 distinct challenges in each of which an AA has to defeat a specific enemy in a non-static environment. In this report, the agent is trained against multiple enemies at once. The agent will therefore be referred to as the Generalist Agent.

A specific problem in every multi-problem environment is how to train the AA such that his learned behaviour can be generalised to problems which it is not specifically trained for. This possible generalisation feature of the solutions found is also investigated in this report. In this research, two EA's called 'A' and 'B' will be used to train a 'Generalist Agent'. The two EA's will be tested against a specific baseline EA taken from [1]. Furthermore, a comparison will be made between 'A' and 'B'. In section 2, a precise formulation of the hypotheses will be given. Furthermore, the EA's will be explained and their use will be motivated. In section three and four, the results will be discussed and a conclusion will be drawn.

2 METHODOLOGY

2.1 Experimental setting

The AA's are trained within the Evoman framework. As mentioned in the introduction, the electronic environment provides 8 distinct challenges in the form of different enemies. In the case of Algorithms A and B, we will train a Generalist AA. This means that the AA is trained against multiple enemies. In this research, both Algorithm A and B are trained against enemy 2 and 6.

Since we will be producing a generalist controller, we have to adjust a few parameter settings in the Evoman environment so that the AA can play against multiple enemies in one run. The multiple-mode' parameter in the environment class is set to 'yes'. This enables the controller to be developed according to a fitness function that represents the weighted-average of the fitness acquired by playing against both enemies. The training phase will be conducted by training the AA against enemy 2 and 6. These two enemies were strategically chosen. In de Araújo and de França [1], every enemy has a short problem description. From observing enemy 2 and 6, we find that two strategies to defeat both enemies independently could be very different based on that the static attack of both enemies and the feature of the specific games are very different. By training the agent on these enemies, the resulting increase in the complexity of the fitness landscape could lead to a solution that focuses less on one specific strategy, thereby reducing the chance that we over-fit. This would possibly lead to a solution that can be better generalised.

The created two Evolutionary Algorithms follow a general design. Both make use of the evolutionary framework DEAP [3]. Our Evolutionary Algorithms use some built-in features of DEAP. Some functions are self-written according to our perception of how the operators and the corresponding values should be.

2.2 Hypotheses

As mentioned, Algorithms 'A' and 'B' will be used to train the generalist agent. The measurements of performance of both Algorithms in the training phase will be the mean/standard deviation of the Fitness per generation averaged over all the simulations and the remaining AA's life point. For comparison between the EA's, the best solution from each independent run per EA is taken and used to perform 5 independent runs against all enemies. The eventual performance measure that will allow comparison between the Algorithms 'A' and 'B' is that the best controller from each algorithm is taken for each of the simulations, and used to test against all enemies. For every independent test, the gains are reported. The gains are calculated as follows: $g = \sum_{i=1}^{n}(p_i - e_i)$. The results will be presented in a box-plot. Each point in the box-plot per EA will be the mean gain for each best solution for an independent run. To compare the EA's to the baseline EA taken from de Araújo and de França [1] the best gain found by a solution of 'A' and 'B' combined is compared to the best gain from one of the EA's, NEAT, used in the baseline paper (TABLE VII).

In summary, the measures used in this report that determines which Algorithm is best revolves around the gains of the best individual solution found by the EA's under scrutiny. Furthermore, the same performance measure is used to compare algorithm A and B against one of the baseline algorithms from paper [1]. The relevant algorithm used for comparison is 'NEAT'. For enhancing the validity of the comparison, the enemies on which 'A' and 'B' were trained are the same as the enemies for which 'NEAT' was trained in [1]. Note that the NEAT represents a state of the art Algorithm. For a precise formulation for the hypothesis in this report, see below:

$$H_1 : mean - gain_A = mean - gain_B$$
$$H_2 : mean - gain_{baseline} = mean - gain_A$$
$$H_3 : mean - gain_{baseline} = mean - gain_A$$
$$H_4 : best - solution_{baseline} = best - solution_{AorB}$$

2.3 Evolutionary Algorithms

The general setup for both algorithms can be found in Table 1 and 2. Every table is divided into two parts: symbolic and numeric parameters. Both EA'S have the same configuration of symbolic parameters. Symbolic parameters refer to the operators used to provide diversity and selection such as blend cross-over and Uniform parent selection respectively. The differences between the EA's are in the numeric parameters. Numeric parameters refer to the specific values of parameters that are linked to a specific symbolic parameter. For instance, if the symbolic parameter is blend-crossover, the corresponding numeric parameter is α.

Furthermore, some numeric parameters correspond to probabilities. Some of these probabilities are set and do not vary over the course of the EA. Both algorithms have a probability of 0.8 for a specified parent-couple producing offspring through recombination and 0.1 for a specified parent to undergo mutation. [2] shows a range of optimal values for both numeric parameters, where we picked the values for our EA's from. The structure of the tournament operator is different for both algorithms. However, the tournament size is 3

for both EA's. This value was chosen to prevent selection pressure from being too high and thus diversity from decreasing too fast. The take-over time according to the formula from Eiben et al. [2] is now approximately 5 generations. Lastly the blend parameter, α is 0.5 which is noted as a good value for the blend cross over in [2].

2.4 Algorithm A

The features of the first algorithm are displayed in table 1. As explained before there are a few difference between the algorithms. The most obvious difference is that *A* uses tournament with replacement as survival selection method. This is a widely used selection mechanism with very promising results [2]. Because the algorithm makes use of a generational model, we make an offspring pool 3 times bigger than the population size. This number is chosen from the literature [2]. The old and recent trend for the size of the offspring pool is $\lambda = 3$ and $\lambda = 7$ times the size of the population. Algorithm A selects couples of 2 in total and in total the estimated size to go through recombination and mutation is 1.5 x population size.

Table 1. Evolutionary algorithm A

Symbolic Parameters	
Representation	Neural Network
Initialisation	Random number generator
Parent selection	Uniform parent selection
Recombination	Blend Crossover
Mutation	Self-adaptive,
Population management	Generation model
Survival selection	Tournament with replacement
Termination condition	Maximum generations reached
Numeric Parameters	
Nodes in neural network	256
Population size	50
Recombination probability	0.8
size selected parent pool	3 * Population size
Offspring per Recombination	2 children per parent couple
Blend crossover param. 'a'	0.5
Child mutation probability	0.1
Gene mutation probability	0.05
Tournament size	3
Number of generations	20

2.5 Algorithm A

Algorithm B bears many resemblances when compared to Algorithm A. This can be observed by comparing the symbolic parameters. Differences between the EA's can be found in the way the offspring is created. Algorithm B picks *N* parents by the uniform parent selection, where *N* is the population size. Each pair of parents will create four children with the recombination probability. By producing multiple offspring, more diversity is created. By using the blend crossover operator, a broader area of the search space can be covered and through creating multiple offspring per parent more space around and between the parents is covered. This in contrast with Algorithm A, that creates more varying parent couples to reproduce. Furthermore, algorithm B uses the tournament selection without replacement. Therefore, each child in the offspring can only be chosen once. Resulting in more diversity in the next generation, since there cannot be any duplicates in the next generation. Selection by tournament without replacement results in a decrease in selection pressure.

Table 2. Evolutionary algorithm B

Symbolic Parameters	
Representation	Neural Network
Initialisation	Random number generator
Parent selection	Uniform parent selection
Recombination	Blend Crossover
Mutation	Self-adaptive,
Population management	Generation model
Survival selection	Tournament without replacement
Termination condition	Maximum generations reached
Numeric Parameters	
Nodes in neural network	256
Population size	50
Recombination probability	0.8
Size selected parent pool	1 * Populations size
Offspring per Recombination	4 children per parent couple
Blend crossover param. 'a'	0.5
Child Mutation probability	0.1
Gene Mutation probability	0.05
Tournament size	3
Number of generations	20

3 RESULTS

In this section, the found results of the different EA's will be presented. At first, the training process is reviewed. The average learning curve of the algorithms over ten independent runs is plotted, afterwards the algorithm's solutions are compared when competing against all eight enemies.

3.1 The Training Phase

For the training phase, we report the average/standard deviation of the mean and maximum fitness and energy points (life) of the agents averaged over1 10 runs per generation. See Figure 2. Whereas the mean fitness per generation develops very similarly for 'A' and 'B', and the best fitness is close around generation 20, the average life points of the AA for 'A' grows with respect to 'B' from generation 15. However, the algorithms do not seem to have converged to their maximum. Therefore, it would be interesting to investigate the algorithm's behaviour after generation 20.

3.2 Comparing 'A' and 'B'

To compare A and B, a box-plot is constructed in Figure 2. For both 'A' and 'B', the mean-gain per independent simulation is represented in box-plot 'A' and 'B'. A T-test is performed to answer $H_1 : mean - gain_A = mean - gain_B$. Both the one-sided and two-sided t-test have a p-value of 0.00. This means that we reject the first hypothesis and suggest the mean-gain of B is better than the mean-gain of A.

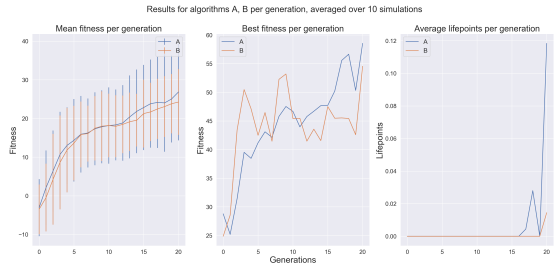

FIGURE 1: The left-hand graph displays a the mean fitness per generation averaged over 10 simulation for algorithm A and B, the error bars represent the standard deviation per simulation, The middle graph displays the best fitness per generation and the right-hand graph the average life points.

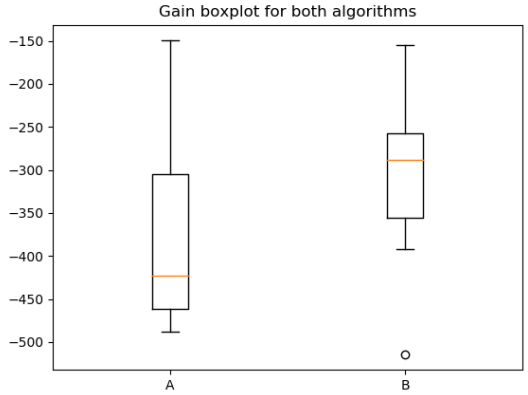

FIGURE 2: The left-hand graph displays a the mean fitness per generation averaged over 10 simulation for algorithm A and B.

3.3 Comparing 'A' and 'B' to NEAT

In order to answer $H_2 : mean - gain_{baseline} = mean - gain_A$ and $H_3 : mean - gain_{baseline} = mean - gain_A$ we compare the mean-gain of A (H2) and B (H3), represented in the box-plot with the mean-gain of NEAT where the AA is trained on enemy 2 and 6. Both t-tests give a p-value 0.00. Additional one-sided t-tests gave significant results for the mean of NEAT being higher than the mean of both 'A' and 'B'. Therefore, we can reject hypothesis 2 and 3. The mean-gain of NEAT is higher than the mean gain of 'A' and 'B'.

To answer $H_4 : best - solution_{baseline} = best - solution_{AorB}$ we look at the sum of the remaining life points of the best solution of A and B combined and compare that to those of NEAT (See Table 3). Table 3 shows that the sum of the remaining life points of NEAT is higher.

Table 3. Best solution life points

Enemy	Life point B	Life points Neat
1	0	0
2	68	74
3	0	0
4	0	0
5	36.4	43
6	21.4	88
3	0	0
4	0	4

Table 4. Best solution life points

Gain (std) Neat	Gain (std) A	Gain (std) B
52 (14)	-374.7 (109.9)	-308.9, 100.5

4 CONCLUSION, DISCUSSION

During this research two EA's where developed to train a Generalist Agent. The performance of both Algorithms 'A' and 'B' was compared to each other and to the baseline, state of the art Algorithm NEAT that trained the AA against the same enemies. Four hypotheses were formulated. We showed that Algorithm 'A' and 'B' were significantly different from each other. We also showed that the performance of both 'A' and 'B' was lower than the performance of NEAT.

Lastly, we showed that for the best solution of A/B compared to the best solution of NEAT, NEAT outperformed our EA's on the sum of the remaining life points. Thus, the behaviour of the AA trained by NEAT better generalises to other problems than the AA trained by 'A' and 'B'.

One shortcoming in our experiments was the amount of computational power that was at hand. Due to this shortcoming, only a small part of the search space was covered. In further research with more computational power, one could consider the following adjustments to get better results. The population size could have been bigger, allowing for a faster and broader search through the search space. Furthermore, our termination condition could be adjusted to the moment that the algorithm does not improve anymore since the plots from the training phase indicate this possible room for improvement.

REFERENCES

[1] 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).

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

[3] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research* 13 (jul 2012), 2171–2175.