Evolutionary Computing - Task II: generalist agent
Group 13

October 25, 2019

Floris Fok
2685993

Mick IJzer
2552611

Kim van Putten
2525287

name
student nr

1 INTRODUCTION

Computational intelligence (CI) is a field of research that applies machine learning techniques to games such a chess to create agents capable of autonomously playing the game intelligently [3]. Games often have various valid play strategies and a big challenge is their unpredictable nature. For an artificial intelligence (AI) to appear intelligent, it needs to be able to adapt to constantly changing game states and uncertainties related to for example consequences of the agents actions or a changing game play style of the opponent.

In our previous assignment we used two evolutionary algorithms (EA) to train specialist agents capable of wining a game from a single enemy that they were trained to play against. The EA were trained and evaluated using the EvoMan framework, which provides an environment for evolving game playing agents for action-platformer games inspired by the classic Mega Man II video game [1][2].

The next challenge is to evolve a generalist agent, capable of playing and defeating different enemies with different behaviours and level designs. A good quality generalist agent should be evolved to have a playing strategy that can adapt and be applied to different or changing situations while maintaining good performance (i.e. not dying and defeating the opponent). We implemented an EA and trained it to produce a generalist agent using two different parameter settings. We evaluate and compare the evolutionary progress during training, based on the fitness of the produced population of solutions per generation, and we examine how well the produced solutions with the highest fitness score perform playing against all enemies in the EvoMan framework based how often they can successfully defeat enemies and how much of their own energy (life) points they preserve at the end of a game.

2 RELATED WORK

Evolutionary algorithms can be used for a wide variety of multi-objective optimization problems with complex search spaces [8]. One such application is the use of EA to create enjoyable video game content. A common use for EA in video games is for the creation of high-performing strategies or AI agents such as AI controlled cars in racing games [7] or a player agent capable of beating a platform game [6]. Togelius et al. demonstrated other applications of EA in video games such as generating racing tracks in car racing games tailored to the human player [5]. De Araujo et al. demonstrated how EA can be evolved to create player agents capable of winning against an opponent in the EvoMan framework, but found that EA trained on a single training scenario have a tendency to overspecialize and are often not capable of winning in different scenarios. Agents trained on multiple scenarios could evolve controllers with a generalist play style capable of beating different enemies [2]. However, evolving a very successful generalist remains challenging, and the different agents produced by different EA were yet unable to win against all scenario's.

We take the results by de Araujo et al. [2] as the baseline to compare the performance of our EA implementation to.

3 ALGORITHM DESCRIPTION

The two EA we compare implement the same evolutionary strategies but have a different parameter setting that affects the behaviour of the parent selection, and we suspect will also have an effect on how the population will evolve each generation.

EA	tourn_3	tourn_6
Representation	265 Float Vector	265 Float Vector
Recombination	Whole Arithmetic	Whole Arithmetic
Mutation	Gaussian $\sigma = 0.5$	Gaussian $\sigma = 0.5$
Parent selection	**Tournament $k = 3$**	**Tournament $k = 6$**
Survivor selection	Deterministic $(\mu + \lambda)$	Deterministic $(\mu + \lambda)$

Table 1: Summary of the EAs.

We make use of the provided neural network with 10 hidden neurons. The phenotype of the problem is the neural network, which is able to control the player when receiving the sensor inputs. The genotype is a vector of 265 floats long, which represents the weights of the neural network, and when translated back to the phenotype determine the behaviour of the agent.

During our experiments in task I evolving specialist agents we found the two issues with out EA implementations that we aim to improve upon with our generalist EAs. Firstly, we had set the mutation rate parameter too high, resulting in too much random exploration of the search space and not enough evolution towards quality solutions. Secondly, the EA would occasionally lose their best found solutions thus far during training due to non-deterministic survival selection. To address the lack of push towards quality we opted for a lower mutation rate parameter $p_m = 1/n$, where n is the number of variables per individual, so that on average one variable is mutated per created individual. For the mutation method we opted for Gaussian mutation. To prevent loss of good solutions during training we implemented the deterministic survivor selection $(\mu + \lambda)$ by merging and ranking the population and offspring and selecting μ individuals with the highest fitness score.

An EA should have a good balance between push towards novelty and push towards quality. We decided to experiment with parent selection to see how it can affect the quality-novelty balance. From literature we found that tournament selection is often a good choice for the parent selection method [4]. Since a lower mutation rate will result in a lower population diversity and $(\mu + \lambda)$ selection also decreases diversity due to high selection pressure we vary the tournament sample size k, but keep it low since a high k further increases selection pressure. We selected $k = 3$ and $k = 6$ as the difference between our EAs to compare. A low $k = 3$ results in close to uniform random selection, whereas $k = 6$ is less likely to select individuals with low fitness scores to reproduce. With both EA $\mu * 1/6$ pairs of parents are selected and each pair produces 2 offspring, therefore $\lambda = 1/3 * \mu$. The implemented tournament selection is deterministic and selects based on the highest fitness scores. The EAs use whole arithmetic crossover to produce children. The α recombination parameter is randomly drawn each time.

A summary of the two EAs is shown in table 1

3.1 Fitness

Both parent selection and survivor selection are based on the fitness of the individuals in the population, and the fitness score of a solution also serves as an evaluation metric in our experiments. The fitness of an individual is evaluated per simulated game based

Enemies	2,5,6
Hidden Layers	10
Mutation rate p_m	$1/n \approx 0.39\%$
Population size	50
Generations	50

Table 2: Parameter setup.

on the end statistics with the following function:

$$fitness = 0.9*(100-enemylife)+0.1*playerlife-log(time) \quad (1)$$

The fitness assigns a high weight to the enemy life. The lower the life of the enemy, the higher the score. This is due to the preferred outcome of a won game in which the enemy life is 0. The weight of the player life is significantly lower to deter endless dodging and avoidance tactics. The time is subtracted so that a faster win will result in a higher score.

The final fitness score assigned to an individual is a normalized score of the fitness values among multiple games, calculated as the mean of all the fitness scores minus the standard deviation of the fitness scores.

4 EXPERIMENTAL SETUP

The parameter setup chosen for our experiments are shown in table 2. Due to the computational cost of training process it is not feasible to train our EAs against a large set of enemies. De Araujo et al. [2] trained their generalist agents against sets of 2 and 3 different enemies. Because we want to be able to fairly compare our experiment results to their results as baseline we decided to train our EAs on a same set of enemies, namely enemy 2, 5, and 6.

4.1 Training experiment

For our experiment we train each EA 10 times on the same three training enemies. The population size and number of generations were selected based on the estimated run times. Through test runs we found that a population size of 50 and 50 generations cost approximately 2.5 hours on a personal computer (Intel i7 quad-core). In total, running 10 training runs on 3 enemies per EA cost 50 computational hours, but the work was divided to three computers and several training runs were run concurrently to bring testing time down to less than a day.

The measured statistics during evolution are the best, worst and average fitness of each generation, as well as the mean player life points left over at the end of a game per generation.

4.2 Evaluation experiment

After the evolution process, we selected the best found solution per training run for each EA based on the highest fitness score. These evolved agents were evaluated 5 times against all 8 enemies in the EvoMan framework, measuring the total gain, calculated as:

$$g = \sum_{i=1}^{n}(p_i - e_i) \quad (2)$$

where n are the number of different enemies, p_i is the health of the agent left over at the end of a game i, and e_i is the remaining health for the enemy at the end of game i. The average gain was calculated per run against all 8 enemies, thus resulting in 5 average gain scores per agent.

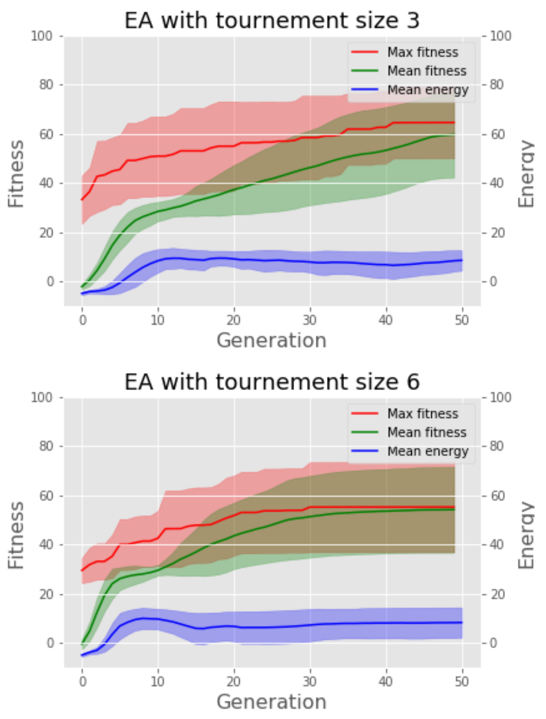

Figure 1: Mean and max fitness with standard deviation plotted for tourn_3 (top) and tourn_6 (bottom), including the mean energy with standard deviation on the same scale in blue.

5 EXPERIMENTAL RESULTS

The graphs in figure 1 show the average and standard deviation of the mean and maximum fitness and energy points of the agents over the generations of the training experiments. We can see that for both EA the mean fitness converges to the maximum fitness, which means that the population in both EA converges towards a (local) optimum. Furthermore, the difference in selection pressure between the two EA becomes clear in the graphs. The mean fitness in the EA with tournament size 6 nears the maximum fitness after 30 generations while the mean fitness of the EA with tournament size 3 is still relatively distanced from the maximum fitness after 50 generations, which means the population has more diversity still. The mean energy levels show a similar pattern in both EA. Both EA found roughly equally best and worst solutions based on the fitness score.

The results of the evaluation of the 10 best agents produced by each EA are shown in the two plots in figure 2. We see that overall the EA with tournament size 3 produced better performing generalist agents in 10 training runs than the EA with tournament size 6. The average gain from the EA with tournament size 3 (M = -199.27, SD = 98.43) was significantly higher than the average gain from the EA with tournament size 6 (M = -249.52, SD = 88.56), t(98) = 2.68, p = .009, 95% CI [13.09, 87.41].

Table 3 shows the mean and standard deviation of the mean gains scores, energy points of the agent and enemies and achieved number of wins of the best solution produced. We chose agent8 evolved by tourn_3 as the absolute best based on it achieving the highest

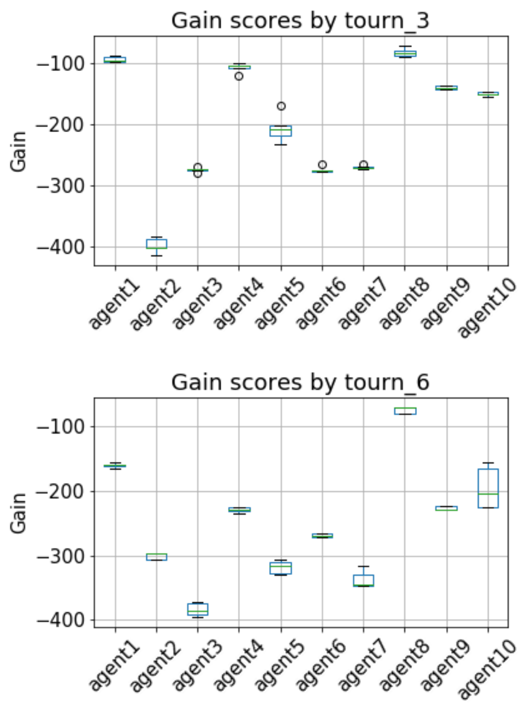

Figure 2: Boxplots showing the mean gain scores per evaluation run per agent produced by the tourn_3 EA (top) and tourn_6 (bottom).

Best agent	tourn_3 agent8
Gain	−82.84 ± 6.62
Energy agent	18.395 ± 0.83
Energy enemy	28.75 ± 0.0
Number of wins	3 ± 0.0

Table 3: Mean and standard deviation of the mean gain scores, agent energy, enemy energy and number of wins obtained over 5 experiment runs.

average mean gain score during the evaluation experiments, which means that it is scored as the most general agent, while achieving the most wins. tourn_6 agent8 scores a slightly higher mean average gain, but is only capable of beating 2 enemies. Compared to the results of the baseline paper our best evolved agent performs slightly worse. The NEAT agents evolved by de Araujo et al. [2] that was trained on the same enemies (2,5,6) achieved a very similar gain but was able to beat one more enemy, and with more energy point remaining. We believe the better performance is because the NEAT algorithm implements a niching technique that helps maintain a better diversity of the population.

6 ANALYSIS AND DISCUSSION

The goal of this experiment was to develop a generalist agent that would perform decently against multiple enemies, and compare the performance of the two EA against each other and the baseline. After training the different EA multiple derived agents were able to defeat 3 enemies. However, it became clear that one EA performed statistically better than the other EA, namely the EA with tournament size 3 outperformed the EA with tournament size 6 and is able to produce agents with a higher gain score on average. Based on this it can be derived that with the currently used parameters an EA with less selection pressure in the parent selection is beneficial for the performance of the EA.

An interesting observation is that both EA have a tendency to converge to a local optimum, and the more a generation has converged to a local optimum, the lower the change change the population will explore outside the local optimum hill and find a higher local optimum in the search space. We believe that both EA could improve their performance with a slightly higher push towards novelty, for example by changing to a survival selection methods with a lower selection pressure or implementing niching techniques to preserve diversity.

Another observation we made is that the fitness function tends to penalize winning strategies that do not perform well across all enemies. Agents that perform uniformly mediocre score a higher fitness that agents that are, for example, capable of winning from half the enemies and lose fast from the rest. If the goal is evolve an agent to win in as many scenario's as possible then it might be beneficial to relax the generalisation a little. A suggested way of doing this is to implement an adaptive fitness function that starts to reward winning against enemies once the population consists of multiple decently performing agents.

7 CONCLUSION

We implemented and experimented with two EA with two different parameter settings to evolve generalist agents that try to find a strategy to win from every enemy by learning from a set of 3 enemies. Our implemented EA were not able to improve on the baseline algorithms by de Araujo et al. [2], but our experiments did demonstrate the effect of selection pressure and the importance of a good balance between pushing towards novelty and pushing towards quality and how it influences the learning capabilities of the EA. A lower selection pressure produced on average better evolved strategies, capable of beating 3 enemies.

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] 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.

[3] Simon M Lucas. 2008. Computational intelligence and games: Challenges and opportunities. *International Journal of Automation and Computing* 5, 1 (2008), 45–57.

[4] Pandey H. M. Mehrotra D. Shukla, A. 2015. Comparative review of selection techniques in genetic algorithm. *International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE) (pp. 515-519). IEEE* (2015).

[5] Julian Togelius, Renzo De Nardi, and Simon M Lucas. 2007. Towards automatic personalised content creation for racing games. In *2007 IEEE Symposium on Computational Intelligence and Games.* IEEE, 252–259.

[6] Julian Togelius, Sergey Karakovskiy, Jan Koutník, and Jurgen Schmidhuber. 2009. Super mario evolution. In *2009 ieee symposium on computational intelligence and games.* IEEE, 156–161.

[7] Niels Van Hoorn, Julian Togelius, Daan Wierstra, and Jurgen Schmidhuber. 2009. Robust player imitation using multiobjective evolution. In *2009 IEEE Congress on Evolutionary Computation.* IEEE, 652–659.

[8] Eckart Zitzler. 1999. *Evolutionary algorithms for multiobjective optimization: Methods and applications.* Vol. 63. Citeseer.