Evolutionary Computing - Task I: specialist agent
Group 13

October 4, 2019

Floris Fok
2685993

Mick IJzer
2552611

Kim van Putten
2525287

name
student nr

1 INTRODUCTION

Learning from real world situations is hard due to the ever changing environment and the chaotic events happening constantly. Most models are not suited to adapt to this constantly changing environment. To overcome this challenge we use a simulated game to measure the robustness of an Evolutionary algorithm, an algorithm that was designed to have highly adaptive properties. We study the effect of implementing two evolutionary algorithms (EAs) and train and evaluate them in a game simulation framework called EvoMan. The EvoMan framework is an environment for evolving game playing agents for action-platformer games inspired by the classic Mega Man II [1][2]. The game contains 8 final boss battles which the AI can learn to play and try to beat.

Our aim is to study the effects of different crossover strategies on the performance of the EA's in winning the game. The two variations of EA are trained to be specialist agents. That is, each EA is trained against a single boss enemy per experiment such that the resulting best found solutions are tailored to one enemy only. Our expectation is that a specialist agent trained on one enemy level should be able to produce a decent game play strategy against this enemy, but that this solution will perform less well against the other enemies, similar to the findings by Togius et al. [4] and de Araujo et al [2].

2 RELATED WORK

Evolutionary algorithms can be used for a wide variety of multi-objective optimization problems with complex search spaces [6]. One such application is the use of evolutionary algorithms to create enjoyable video game content. A common use for evolutionary algorithms in video games is for the creation of high-performing strategies or AI agents such as AI controlled cars in racing games [5] or a player agent capable of beating a platform game [4]. 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 [3]. 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 [2].

3 ALGORITHM DESCRIPTION

We implemented two evolutionary algorithms (EA) that use different reproduction crossover operators but are functionally the same in every other aspect.

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.

Algorithm 1: General algorithm of the EA

```
Result: A set of evolved solutions
Initialize(population size);
for each generation do
    parents=parent_selection(population);
    for each pair(parents) do
        children=sex(parent 1, parent 2);
        mutate(children);
        evaluate(children);
        new_population=population+children;
        population=survival_selection(new_population);
    end
end
```

Both EAs start by initializing a population consisting of random solutions (individuals). For each generation, a set number of pairs of parents are selected. Parents are selected on their fitness score via the roulette wheel method, where individuals with a higher fit have a larger probability to be selected. This way, the most fit individuals are most likely to produce offspring, but less fit individuals also have a chance to reproduce. Each pair of parents produces two offspring, which undergo random mutation to increase diversity and push towards novelty and hopefully allow to explore outside local optima. The mutation operator looks at every weight in an offspring, if the weight is selected for mutation it will be changed into a random number between -1 and 1. The chance of mutation, μ, is kept constant throughout generations. The offspring gets evaluated for their fitness, and a new population is formed by combining the previous generation with the newly produced children and performing a selection process to reduce the population size. Survivor selection is done with the same selection method, roulette wheel weighed by fitness. Thus, some individuals of the previous generation might survive, as might less fit children.

3.1 Crossover methods

The two EA differ in their crossover methods. The first EA, referred to as the 'parts' variation, uses recombination. A crossover point is randomly chosen, and the parents are split into a heads and tail part at this crossover point. The crossover point is randomly generated for each parent pair. The parts of the parents are copied to the children, such that the first child inherits the head of parent 1 and the tail of parent 2, and vice versa for the second child.

The second EA, referred to as the 'fraction' variation, creates children by calculating a weighted mean average of its parents. A weight is randomly selected, and values of the first child is calculated as $weight * parent1 + (1 - weight) * parent2$, and the second child is calculated as $(1 - weight) * parent1 + weight * parent2$.

3.2 Fitness

The fitness of all individuals are calculated with EvoMan's default fitness function based on the end statistics of a game:

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

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 fitness score is used by the selection process in both parent and survival selection, and also serves as an evaluation metric of the experiment.

4 EXPERIMENTAL SETUP

Both EAs are trained against the same three enemies. For statistical certainty we trained each EA against each enemy ten times, for a total of sixty tests. All parameters are kept the same for both EA against all enemies. The parameters are shown in table 1.

Enemy levels	1,5,7
Playermode	AI
Difficulty level	2
Hidden Layers	10
Mutation rate μ	0.3
Population size	25
Generations	50

Table 1: Parameter setup

Due to time limitations and the computational intensity of the algorithms we were unable to perform extensive parameter tuning. Therefore, to select a mutation rate μ we ran a small test (population size 10, 75 generations) with both EA against a single enemy with varying mutation rates. We hoped to see how different mutation rates would affect the performance of learning of the EA based on the best and average fitness of the generations, but perhaps due to the small scale of the test we observed very little difference. We decided on $\mu = 0.3$ in the hope that this would give the EA reasonable diversity and allow the EA to explore outside local optima and reach a global optimum.

The population size and number of generations were selected based on the estimated run times. Ideally we would have preferred a larger population size and larger number of generations, but we were limited in computational resources and through testing we found that a population size of 25 and 50 generations cost approximately 40 minutes per test on a personal computer (Intel i7 quad-core). In total, the run time of the total experiment cost 37.5 computational hours. Since all the test are run with the same parameters the negative effects would harm all equally.

The measured statistics are the best, worst and average fitness of each generation.

We also selected the best found solution for each EA against each enemy based on the highest fitness score, and let these solutions play against all 8 enemies to compare the performance in different play level settings. The environment parameter setup is kept the same with the enemy controlled by the default AI and the difficulty level set to 2.

5 EXPERIMENTAL RESULTS

Figure 1 shows the average best fitness per generation for both EAs per enemy level. We see that for enemy level 1 the EA with part recombination appears to outperform the EA with weighted mean (fraction) crossover. It appears that the fraction EA would lose its best found solution during survival selection phase and be unable to re-reach this earlier found optimum. A similar phenomena of losing a best solution can be observed for both EA against enemy 7, although the negative impact appears to be much smaller. Both EA appear to perform really well against enemy 5 and find a well working play strategy in the first few generations of training. Enemy level 1 is the only enemy where the average increases, enemy 5 has a small deviation and enemy 7 looks like a random walk.

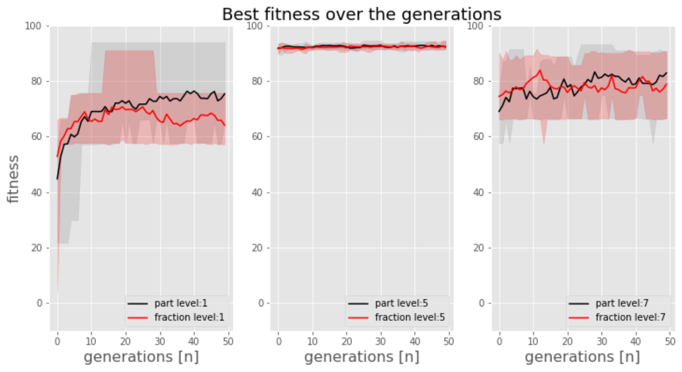

Figure 1: Average best fitness achieved over the generations for enemy level 1, 5, and 7.

Figure 2 shows the median and standard deviation of highest fitness score reached per experiment. The EA with recombination strategy 'parts' (M = 75.39, SD = 11.55) outperformed the EA with strategy 'fraction' (M = 64.09, SD = 5.78) on enemy level 1, t(13.25) = -2.77, p = 0.016. No significant differences were found between the the strategies on the other enemy levels, t(18) < 1.21, p > 0.241.

We selected the best found solutions per EA per enemy based on the highest fitness score, and ran each solution against all 8 enemy levels 5 times. We measured the average fitness of each solution against each enemy, the results of which are plotted in

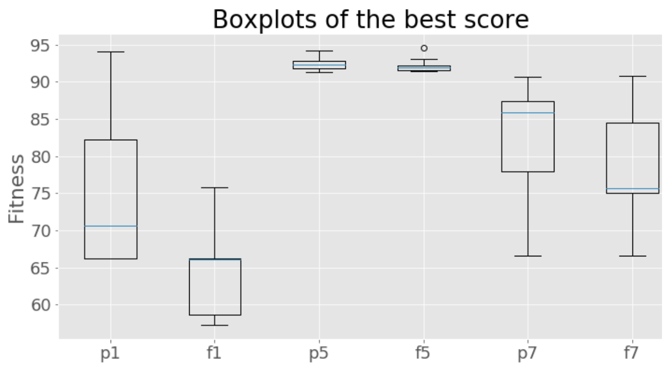

Figure 2: Boxplot of best fitness achieved at the end of the generations per training experiment. Plotted on the x-axis are the different EA (p - parts, f - fraction) trained against enemy 1, 5 and 7.

enemy	p1	f1	p5	f5	p7	f7
1	**94.02**	75.75	56.60	57.75	3.31	12.79
2	30.30	21.23	48.08	21.27	39.07	**75.03**
3	38.75	-6.11	21.17	**65.76**	30.78	21.17
4	11.97	2.98	3.00	2.85	21.75	21.40
5	91.24	92.47	93.82	**93.92**	82.74	89.63
6	-5.40	-5.56	2.85	12.89	13.01	-4.90
7	13.89	13.89	-2.71	-3.26	90.67	**90.80**
8	21.48	21.55	47.88	21.39	3.52	**57.00**
mean score	28.89	20.06	30.58	30.69	40.22	**50.02**

Table 2: Average fitness scores achieved by the best specialist solutions against each enemy level. Bolded are the highest fitness scores achieved per enemy level.

table 2. The mean and standard deviation of the average results per EA are plotted in figure 3. We see that the best solution produced by the fraction EA trained against enemy 7 appears to perform best against all the enemies compared to the other specialist solutions.

Our hypothesis was that an EA trained against a single enemy would likely perform poorly against the other enemies, which appears to be supported by the plot. The highest scores correspond with the enemy level the solutions were trained on, with a notable exception for the best solution of the fraction EA trained against enemy 1, which achieved a win against enemy 5. The mean average best score for each EA solution is overall quite low, and no differences in performance between were found, F(42, 5) = 0.22, p = 0.951.

6 ANALYSIS AND DISCUSSION

Our hypothesis was that specialist EA trained on a single enemy level would be able to find a well performing strategy against that specific enemy, but that the training would overfit to the single enemy and the solution would perform worse against the other enemies. The results found by our experiments appear to support this hypothesis, with a notable exception for the fractions EA trained against enemy 1, which was able to win against enemy 5. But we suspect that this is due to the fact that enemy level 5 is not hard to win against since the EA trained against enemy 5 were able to find a winning strategy almost immediately during the training process.

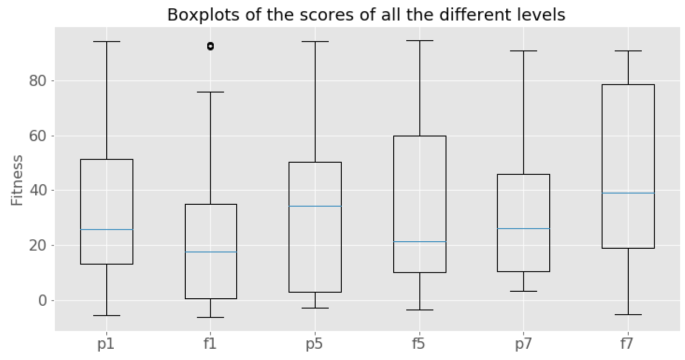

Figure 3: Boxplot of the average fitness score of each best solution against all enemies.

Overall, the parts EA that generates offspring by donating a heads and tails part from both parents to the children appears to perform better in the specialist scenarios, though the difference in performance is only statistically significant against enemy 1.

7 CONCLUSION

In accordance with the findings by Togius et al. [4] and de Araujo et al. [2] our experiment shows that specialist EA trained on a single enemy tend to overfit and perform poorly against other enemies or in different game level designs.

Both EAs appear to be prone to losing their best found solution during the training process, which leads up to believe that our EAs push too much towards novelty and sacrifice the push toward quality due to a high mutation rate and the probabilistic nature of the survivor selection process. Going forward we would like to improve the performance of the EAs with more extensive fine-tuning of the mutation rate parameter, and experimenting with different selection processes that are more elitist or deploy a "comma strategy" (μ, λ) rather than a "plus strategy" $(\mu + \lambda)$.

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] 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.

[4] 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.

[5] 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.

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