EvoMan Group 5

Evolutionary Computing Task 2: Generalist Agent (October 25, 2019)

Marcus van Bergen
University of Amsterdam
marcus.vanbergen@student.uva.nl
2682023

Jacobus Dijkman
University of Amsterdam
jacobus.dijkman@gmail.com
2686470

Steven Raaijmakers
University of Amsterdam
sjraaijmakers@gmail.com
2655645

Russell Josh Evangelista
Vrije Universiteit
r.j.evangelista@student.vu.nl
2580005

ABSTRACT

In this paper we use two different evolutionary algorithms (EAs) to evolve a neural network which controls the movement of a player in the game EvoMan. The goal is to create a generalist agent which achieves the highest possible gain against eight enemies. The main research question is to find out the effect of a crossover operator in a generalist type evolution. Our EAs have been evolved against three of the eight enemies and are checked for performance against all eight enemies. A comparison with a baseline EA [1] has also been conducted and to take any measurement errors produced by the environment into account, we have repeated the simulations several times. The EA with crossover performed better on average than the EA with crossover and both EAs seemed to perform worse than the EA baseline paper. However, the increase in performance of the EA with crossover seemed to stagnate after about 10 generations. An explanation for this could be premature convergence due to the survival selection operator used. A wrong choice in parameters could also be responsible.

1 INTRODUCTION

In this paper we will compare an EA without recombination (EA1) versus an EA with a crossover operator (EA2) and compare this with the baseline paper [1]. Unlike the previous paper, where the aim was to find the optimal player movement versus a small amount of enemies in EvoMan, this research will focus on creating an agent which performs optimally against all enemies. The research question of this report is as follows: *"What is the effect of a crossover operator on the finding of a generalist agent for the EvoMan game?"*.

1.1 EvoMan

As described in our previous paper, the EvoMan game allows a player to fight different enemies. Each enemy has its own movement and trajectory for the projectiles; some enemies are also fought on a different landscape, with a different amount of gravity, and react differently to player inputs. Table 1 displays a short summary of the different enemies and their behavior.

Table 1: Enemies

Enemy name	Number	Enemy strategy
Flashman	1	Follow player, freeze movements, shoot eight projectiles
Airman	2	Shoot six projectiles, move to and from opposite sides of screen
Woodman	3	Creep to player while attacking. Attack: shot 4 projectiles vertically from screen
Heatman	4	Shoot 3 sequence projectiles to player. When attacked becomes a projectile and fast towards player
The Metalman	5	Level: moves player towards or away from enemy. Shoot randomly and when attacked
Crashman	6	Jump over player and drop bomb on him. If shot, throw extra bomb
The Bubbleman	7	Level: spikes in ceiling. Different physics, when spikes touch instant death. Shoot projectiles at player
Quickman	8	Jump at player and shoot random projectiles

1.2 Generalist Agent

In our previous research we created EAs which were evolved versus only one enemy, the so called specialist agent. In this research we create an agent in such a way that it is able to provide solutions for varying problems that are thrown at it. In the context of the EvoMan game, this translates to a single player character being able to defeat multiple enemies, which is called the Generalist Agent.

2 METHODS

We created two EAs using the python3 DEAP framework [3]. EA1 does not use a cross-over operator, while EA2 uses a whole arithmetic cross-over operator. The player character in both EAs are controlled by a neural network that takes sensory data as input and outputs a certain set of moves for a given time point. The goal of the EAs is to find the optimal weights for the neural network of the player-controller with which the player character is able to defeat the most enemies. This way of evolving a neural network is also referred to as neuro-evolution [2]. The choice of our training set and the operators used are explained below. Table 3 shows an overview of the operator choices used for the two EAs.

2.1 Training set

According to [1] it should be possible for an EA to find a general strategy that is able to defeat several enemies. Knowing this we decided to cluster the enemies based on which enemies are beatable with the same strategy to try to produce an effective and efficient training set of enemies. Thereafter, the best solution is deployed against all 8 enemies to assess its performance. Table 2 shows an overview of the cluster of enemies used for the training set. Enemies 4, 5 and 7 are chosen for the training set. These three enemies should give a good representation and will allow the fitness landscape to be big enough for the player to learn a strategy to defeat multiple enemies. Enemy 6 has explicitly been left out of the training set since this enemy requires a very specific and hard strategy to win against.

Table 2: Enemy clustering

Cluster of enemies	Strategy
1,2,3,5	Jump & shoot
4, 8	Corner Jumping & shoot
7	Jump with release timing & shoot

Table 3: Algorithm composition

	EA1	EA2
Phenotype	Neural network	Neural network
Genotype	Biases of NN	Biases of NN
Representation	$[v_i \dots v_n] \rightarrow [w_i \dots w_n]$	$[v_i \dots v_n] \rightarrow [w_i \dots w_n]$
Gain function	$\sum_{i=1}^{n}(p_i - e_i)$	$\sum_{i=1}^{n}(p_i - e_i)$
Recombination	*none*	Whole Arithmetic
Mutation	Gaussian mutation	Gaussian mutation
Parent selection	Tournament	Tournament
Survivor selection	Best N_{pop}	Best N_{pop}
Initialization	Random vector	Random vector
Termination	45 generations	45 generations

2.1.1 Gain. To assess the overall success of an individual across all enemies in the training set we use the gain measure:

$$gain = \sum_{i=1}^{n}(p_i - e_i), \tag{1}$$

where n is the number of enemies, p_i the remaining health of the player versus enemy i while e_i describes the remaining health of enemy i.

2.1.2 Phenotype. The phenotype of each individual in the population is a weighted neural network with 20 input nodes and 1 bias node, 10 hidden nodes and 1 hidden bias node and 5 output nodes. The values of the input nodes are given by the sensory input of the player controller. The player controller receives 20 different sensory inputs. The sensors give information on the vertical and horizontal distance between the player and the enemy; the directions the player and enemy are facing, and the vertical and horizontal distances between the player and a maximum of ten enemy projectiles. The output nodes of the neural network is the list of instructions returned by the player controller which in turn determines the player movement at time t [4].

2.1.3 Genotype. The attribute of an individual is a weight $w \in \{-1, \dots, 1\}$ which maps to each corresponding edge of the neural network. In total there are 265 edges between the nodes. This means that the genotype of each individual consists of an array of 265 floating point numbers in which element i contains the weight of edge j [4].

2.1.4 Representation. Given that our phenotype is the neural network with weights, our representation is a neural network where the genotype is mapped to the weights as follows: $G_i \longrightarrow P_j$.

2.1.5 Parent selection. The parents are selected using a tournament selection with replacement. Since our problem will most likely have multiple optima, we choose to keep the selection pressure quite low. We have opted for a $k = 2$ tournament size, which is repeated 100 times. This creates a list of 100 individuals (the same size of the total population size), listed in the order by which they won in the 100 tournaments.

2.1.6 Recombination. The crossover operator that EA2 utilized was a whole arithmetic crossover operator for floating point representations. This choice was motivated by the wish to keep the balance of exploration and exploitation of our neural network. This type of crossover should induce variation to keep diversity, while also keeping the genetic changes relatively low compared to, for example, blend crossover.

$$\begin{gathered} \mathbf{z_1} = (1-\alpha)\mathbf{x} + \alpha\mathbf{y} \\ with\ \alpha \sim \text{Uniform}(0,1) \end{gathered} \tag{2}$$

Every couple that is selected to mate creates one child. Because the number of parents is 100, this means that there will be a total number of 50 children created. Each mating session has a crossover probability of 0.5 ($P_{cx} = 0.5$). When no crossover takes place, a clone of one of the parents will be produced. For EA1, $P_{cx} = 0$ such that this mating always produces a clone of one of the parents.

2.1.7 Mutation. After the mating, all individuals have a chance to undergo a nonuniform mutation with a Gaussian distribution with $\mu = 0$, $\sigma^2 = 1$. To not radically change the weights of the neural network at once, we have opted for a mutation probability P_{mut} that is set to 0.2. That means that every allele of an individual will mutate with a probability of 20%.

2.1.8 Survivor selection. After the offspring is added to the population, the least fit individuals are removed from the new population according to the $(\mu+\lambda)$ scheme: the offspring is evaluated and added to the population, and from this combined population, the 100 (population size) individuals with the highest fitness are selected as population for the next generation.

2.1.9 Initialization. All individuals in the initial population are initialized with an array of 265 floating point numbers. These numbers were randomly chosen between -1 and 1 with a uniform probability.

Table 4: Average fitness over five runs for the individual in EA2 with highest gain.

Enemy	1	2	3	4	5	6	7	8
Fitness	-80.	58.	-30.	-30.	59.44	-60	-10.	22.6

2.1.10 Termination. We terminate all of our experiments after 45 generations due to computational limits. Since we want to compare both algorithms, we do not include any other stop conditions. By doing so we can easily compare the performance of our algorithms against each other.

3 RESULTS

The goal of both EAs is to find the individual with the highest gain across the test set among all generations. For statistical correctness we ran ten evolutions for both EAs fighting against enemy 4, 5 and 7, resulting in ten highest-gain-individuals per EA. These evolved individuals then fought against all eight enemies, which was was repeated 5 times. For every repetition we kept track of the gain and the amount of games won. This eventually resulted in an average gain and an average amount of games won for every of the ten individuals per EA. A boxplot of these average gains is shown in figure 1a. The average gain over the generations for both EAs is shown in figure 1b.

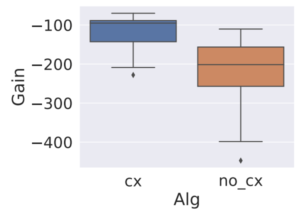
Out of the ten highest-gain-individuals per algorithm we pick the best individuals. We pick the individuals in both algorithms with the highest gain and the most wins as “the best”. For EA1 the individual with the highest gain is the same individual as the one with the most wins. However, for EA2 these individuals differ. The fitness versus every enemy for all four winners is shown in figure 2.

Out of these four, we pick the individual in EA2 with the highest gain as our winner, since it has the highest overall fitness. Its numerical values are shown in table 4 (which correspond with the blue bars in figure 2).

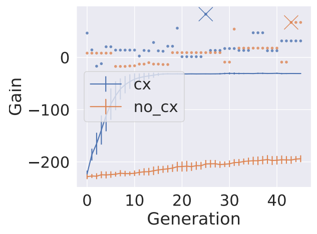
4 DISCUSSION

From figure 1b, we can see that the average test-set gain of the population seems to increase significantly more when using crossover. However, since the goal of the EAs is to find the highest performing individual across all generations, it is more interesting to look at the difference in gain between the best performing individuals of both EAs. In figure 1a, we can see that also the individuals with the highest gain across all enemies score better with crossover than the individuals without crossover. This suggests that the use of a crossover operator has a positive effect on the finding of a generalist agent for the EvoMan game. This is contradictory to the results of our first report [4].

These contradictory results could be explained by the different choice of crossover operator used in this report, in combination with the different choice of selection and mutation operators used. Another possible explanation could be that we did not use parameter tuning for the EAs in this report, due to a limited computational budget. Therefore, it could be that the parameters used in this report were not optimal for the EAs, and that the optimal parameters



(a) Average gain for the best ten individuals runs per EA



(b) Test-set gain evolution over generations.

Figure 1: Gain measures. a) boxplot showing the gain across all enemies for the 10 individuals that got the highest fitness in their run. For each of these individuals, the gain is tested 5 times for all enemies and the averages of these 5 experiments are illustrated in the boxplot. b) The lines illustrate for both EAs the gain versus three enemies (4, 5 and 7) which is averaged over ten runs. The error bars denote the standard deviation. The dots indicate the maximum fitness per generation within all ten runs.

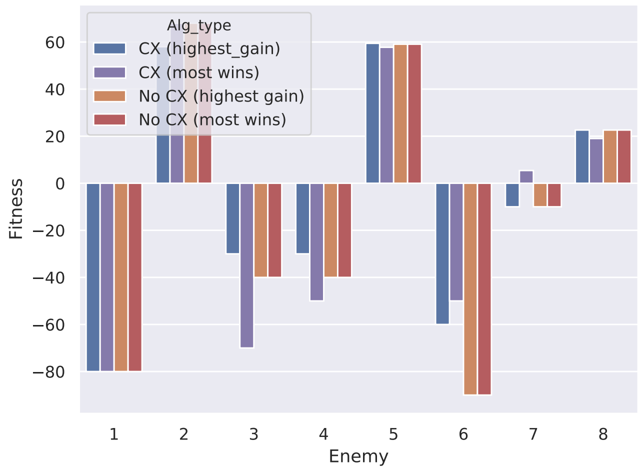


Figure 2: Average fitness over 5 runs per enemy.

for EA1 (without recombination) would actually give better solutions than the optimal parameters for EA2 (with recombination). In contrast, the results could also be explained by the fact that the EAs have to solve a multimodel problem, with many strategies possible to beat the different enemies. Because of this, there are likely many local optima that the EAs can converge to. Therefore, it is important that the EA is able to explore the search space before it starts exploiting a certain local optimum. Because crossover increases exploration of the search space, cross-over could cause EA2 to find higher optima than EA1. Figure 1b also shows that the gain of EA1 seems to still climb linearly at the end of the 45 generations. Mutation could, therefore, cause EA1 to eventually reach, and perhaps surpass EA2, but this would take many more generations.

When observing the change in gain of EA2 (with crossover) in figure 1b, we see that the rate of change seems to stagnate around 10 generations. This could be a result of the survival selection operator used. Because only the best individuals are chosen from the $\mu + \lambda$ group of individuals, this could reduce early diversity and lead to premature convergence. As an alternative to this choice in operator, it is worth investigating whether a probability dependent selection mechanism, like rank-based selection, would counteract this premature convergence and could generate a higher gain.

Figure 2 shows that the highest gain individuals for both algorithms have very similar fitness scores against all enemies. This is illustrated by figure 1b: although the mean gain of the population seems to be very different between the two EAs, the highest gain in their populations seem to be more similar. When instead looking at the individual with the highest number of wins for each algorithm, we see that EA2 is able to beat 4 enemies on average, while EA1 could only beat 3 enemies on average. Since the baseline paper is able to beat 5 enemies and the highest total gain found in this paper was -65, the results of this report show that the combination of EA operators used in this report is less effective in the finding of a generalist strategy than the baseline paper. This could in part be attributed to the fact that the baseline paper uses a probability based survivor selection. Like previously discussed, this could counteract the premature convergence that happens for EA2.

5 CONCLUSIONS

We investigated the effect of the cross-over operator on the performance of an evolutionary algorithm (EA) for a generalist agent. The goal of the EAs was to find a strategy that could defeat the most enemies in the EvoMan game. We found that the EA with crossover performed better on average than the EA without crossover. However, when comparing the EA with crossover to the baseline paper [1], it seems that this EA performs worse than the EA used in this paper. This means that this combination of parameters and EA operators is not a viable alternative to the combination used in the baseline paper. Parameter tuning and parameter control could be implemented to further improve the EAs to see if the EAs can outperform the baseline and to see if they will eventually drift apart in performance. Finally since our problem most likely consists multiple optima in the fitness landscape, we suggest using a probability based survivor selection method, to counteract premature convergence to a local optimum solution.

REFERENCES

[1] 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.

[2] Dario Floreano, Peter Dürr, and Claudio Mattiussi. 2008. Neuroevolution: from architectures to learning. *Evolutionary intelligence* 1, 1 (2008), 47–62.

[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.

[4] Russel Josh Evangelista Jacobus Dijkman Marcus van Bergen, Steven Raaijmakers. 2019. Evolutionary Computing Task 1: Specialist Agent. (2019).