N-point crossover: Investigating the effects of different values for N on the performance of a generalist agent

Report for assignment 2 of Evolutionary Computing

Group 58

Yuyu Bai | 2732696
Willem Gerritzen | 2522283
Arne Sailer | 2748382
Pieter Meulenbelt | 2708245

Task 2: Generalist agent

Artifical Intelligence
Vrije University Amsterdam
2021.10.1

1 INTRODUCTION

Variation and selection are the two fundamental forces that form the basis of evolutionary systems [2]. In the field of evolutionary computing, these are therefore important concepts to gain understanding of. In this paper we will focus on the role of variation in evolutionary algorithms. Crossover/recombination is the variation process in an evolutionary algorithm (EA) where offspring new solutions (offspring) are created from the genotypes of the parents in an existing population.

An interesting aspect of crossover in EA's is that there are multiple methods available from which you can choose. In this paper, we are interested in the crossover operator for the genome representations in a genetic algorithm. This operator combines the genetic information of the selected parents to create new offsping by randomly choosing points in both parents chromosomes to swap the information when creating new offsping. This selection influences the variation by creating unique children for the next generation while preserving the similarity of the parents.

In this paper we will focus on two types of crossover; two-point crossover, which selects two points to be swapped, and 5-point crossover, where 5 points are selected for swapping information. Using multiple crossover points causes more disruption in the creation of new offspring.

In this paper we are interested in the influence of the amount of crossover points in a genetic algorithm. Because crossover is an exploratory mechanism, by searching in the space between two relatively fit individuals, we expect that a higher number of crossover points would result in a better resulting fitness in the later generations by being able to more extensively search the space between parents in creating offsping.

2 METHODS

In this paper we have created a basic genetic algorithm. The Algorithm first creates a random population with the size of 100. This population gets evolved in the following way: the offspring is created by using the crossover method on the parents. This offspring is mutated with the mutation method. Then the selection methods is applied to the parents and the offsprings. The new population is then created by adding the selected offsprings to the selected parents.

Evoman framework. In this study the Evoman framework was used as the testbed for testing the evolutionary algorithm. We used this framework to train a generalist agent that is trained to fight with all bosses. A more precise explanation about this framework is provided in a previous research by da Silva Miras de Araujo and de Franca [1].

Crossover method. In this experiment, the representation method is a Real-Valued Representation where parents are selected by their highest fitness value. An analogous operator used for bit-strings can be used here, but now split between floats. In this experiment we have chosen two different settings for the crossover function. The first setting is selecting two points where the chromosome is broken into, where the offsprings are created by assembling the alternative segments of the parents. The second setting is an extension of the first one, but now 5 points in the parent chromosomes are selected to assemble the offspring. The limitation of this kind of recombination factor is that it can not introduce new values but only give us new combinations of existing values.

Mutation method. For the mutation we used the same mutation method as our assignment 1. There is a 80% chance that an offspring will get mutated by a random number from a Cauchy distribution. The new values will be introduced by mutation operation.

Selection method. For the selection method we used a tournament selection. This selection method makes the individuals randomly compete with each other. This selection is the most widely used selection for genetic algorithms due to its simplicity and the ease to control selection pressure [2].

The percentage of winners that are used for the new population is introduced as the KEEPWINNERS value in our assignment1. This value is represented by a value between 0 and 1, going from selecting 0 to 100% of the winners. Altering this value influences the selection pressure which we define by takeover time. So a lower value increases the exploration, and higher value the exploitation of the algorithm. For this selection method we used a deterministic parameter control method to control this KEEPWINNERS value. The value of a parameter is altered by some deterministic in rule predetermined manner without using any feedback from the search.

Fitness function. We used the default Evoman fitness function as shown below.

$$f' = \bar{f} - \sigma \tag{1}$$

where f' is the consolidated fitness by $\bar{f}$ and σ, that are the mean and the standard deviation of fitness against the all enemies in chosen group for training. And the fitness for each enemy is defined as below:

$$f = \gamma * (100 - e_e) + \alpha * e_p - \log t \tag{2}$$

t to represent the number of time steps of game run. e_e and e_p are the energy measure after a game, values can be from 0 to 100. Constants γ and α assume values 0.9 and 0.1 respectively.

Parameter Control and Experimental Setting. In our evolutionary algorithm, the setting of the KEEPWINNERS value affects the population's diversity in the algorithm, and is a compromise between exploration and exploitation. In the beginning, We want our EA to keep the exploration high, so it doesn't converge too quickl. To do this, the KEEPWINNERS is set to a relative lower value. With the continuous evolution of EA, the KEEPWINNERS is increase gradually to accelerate the speed to reach the optimal point. In our experiment, the KEEPWINNERS value increase from 0.2 to 1 in a step of 0.2 change at every 10 generation. Another important parameter in our experiment is the N-POINT. In order to investigate the influence of n-point value, we choose one even N-POINT 2 and one odd N-POINT 5 to generate 2 EA.

4 Group of enemy, [2,4],[2,6],[7,8] and [1,5] respectively, were used to find the generalist EA which can have a excellent performance among all enemies.

To analyze the EA, we will look at both the average fitness and the highest fitness in the 50th (last) generation in the 10 simulations. For each enemy and n-point value, as well as the trend over the simulated generation to see how well the variations of the algorithm preforms on each group of enemy. This will give us insights in how the n-point values influence the fitness of the population and try to find what the difference between odd and even n-point crossover operation.

Budget. It takes around 1 hours to run 1 attempts of this evolutionary algorithm for one group of enemy and one type of crossover value. So for 2EA and 4 group of enemy, it takes 80h in total.

3 RESULTS AND ANALYSIS

In this paper we have trained an evolutionary algorithm method where we experimented with the crossover method.

Figure 1 depicts how the average and best performance vary over time for the crossover n-values of 2 and 5, respectively, for enemies (1 and 5), (2 and 4), (2 and 6), and (7 and 8). We trained each enemy combination 10 times to ensure that no random factors influenced the result.
The following are some of the findings drawn form the graph:

- The first 10-20 generations where the KEEPWINNERS value is lower than 0.5 the fitness of the population is getting worse or just slightly better for both crossover settings. It is visible in every graph that the mean fitness values increases a lot more after every 10th generation, since this is the time when the KEEPWINNERS value increases. With in every 10 generations the mean fitness shows the usual logarithmic curve that are typical for fitness values in evolutionary algorithms.
- The combination of enemy (7 and 8) is the most straightforward to beat. A "good" individual (fitness > 60) was found immediately. Enemy (2 and 6) was also not too hard to beat for the algorithm. Here it took less than 20 generations, where KEEPWINNERS value was still pretty low, to reach a fitness of over 60. For n = 2 both of them got to a fitness of over 80.
- The n-value of 2 can achieve a better fitness values than n = 5 for the most group of enemies, for enemy (1,5),(2,4) and (2,6), but only for enemy (7 and 8) n = 5 is slightly better after 50 generations than n = 2. This result is different from what we expected. And according to some previous research, whether the value of n is odd or even will have a greater impact on the result. Is there are some relation between genes from the front and end, the related genes will never appear in offsprings if n is odd. Following is a simple demonstrate of what happened when n is an odd number.

After a visual inspection of the results, a statistical analysis confirmed what the graphs show. For the agent trained with enemies 1 and 5 (M = 76.7, SD = 0.874), there was no significant difference between both n-point values in best fitness values, $t(509) = 0.296$, $p > .05$. The same was true the agent trained against enemies 7 and 8 (M = 81.4, SD = 0.540), $t(509) = 0.372$, $p > .05$. However for the agents trained against enemies 2 and 4 (M = 61.2, SD = 1.42) and enemies 2 and 6 (M = 49.7, SD = 1.51), the 2-point crossover function scored significantly higher than the 5-point crossover function with $t(509) = 2.31$, $p = .021$ and $t(509) = 8.97$, $p < .001$ respectively.

Figure 2 illustrates the performance of the top individuals for n = 2 and 5. We tested each experiment ten times to ensure that no random factors influenced the results. And result for statistical analysis are shown in table below. Assume a 5% Significance level, there are no significant difference between the best individual between 2 crossover method.

Table 1: Performance of our Best Solution

enemy	statistic t	p-value
1,5	-1.52	0.14
2,4	1.96	0.06
2,6	0.54	0.6
7,8	-0.41	0.68

3.1 Compared to previous work

Maris had already carried out these experiments in her previous work. Using the Evoman Framework, she tested every available combination of the eight enemies with the neat algorithm.[1]. Although we are able to beat every of enemy combination Maris achieved a higher fitness then us. It may because of the neat algorithm not only evolves the weights of the Neuron Network, but also the structure of the Network.

3.2 Competition

The of the best solution we found is one solution from EA trained with enemy 1, 5, and n value is 5.

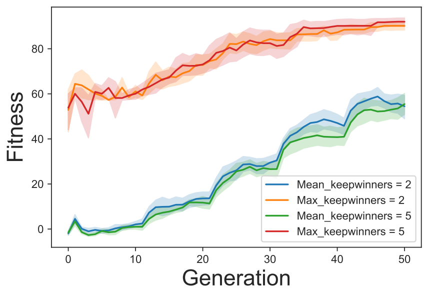
Table 2: Performance of our Best Solution

Enemy	Player energy	Enemy energy
1	90	0
2	80	0
3	0	20
4	0	70
5	94.48	0
6	0	100
7	71.8	0
8	0	10

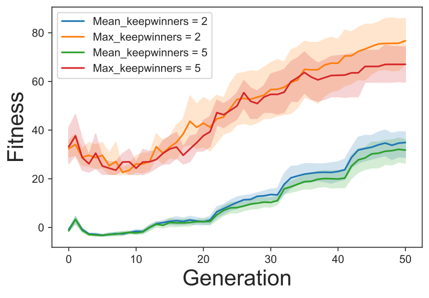
4 CONCLUSION

As explained at the start of this work, we investigated the impact of using a 2-point crossover function versus a 5-point crossover function. Because of a more extensive search, we expected to observe a positive correlation between the n-point value and the resulting fitness, especially in later generations.

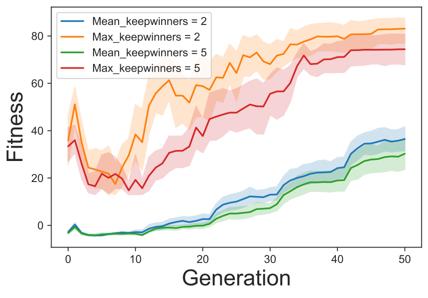
As the results have shown, these expectations were partially correct, as only half of our agents were able to show a significant difference between a 2- and a 5-point crossover. Although this would imply previous research to be correct, it does raise question as to why not all agents were able to show a significant difference. Clearly, the choice of enemies to train against (chosen as they



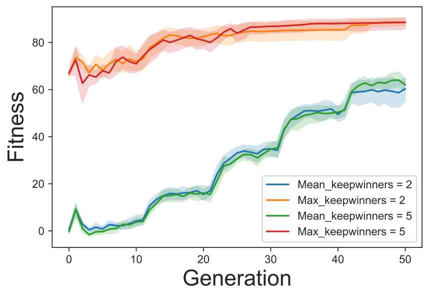
(a) Enemy 1 and 5



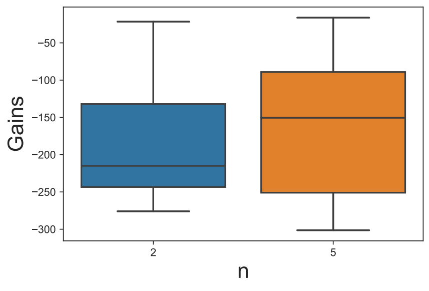
(b) Enemy 2 and 4



(c) Enemy 2 and 6

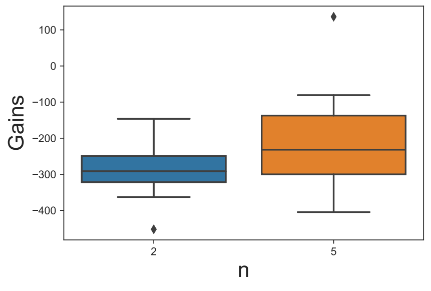


(d) Figure1 Enemy 7 and 8

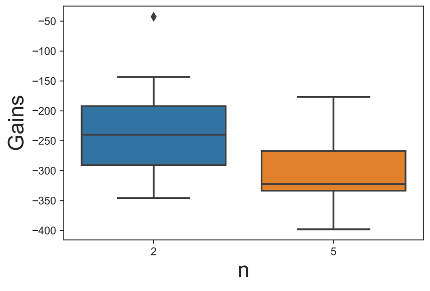


(e) Figure 2 EA trained for Enemy 7 and 8

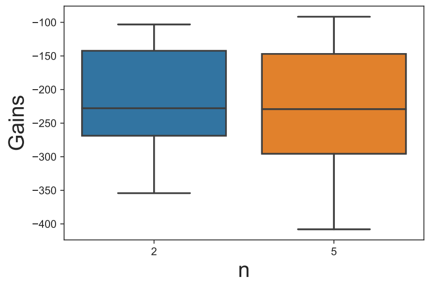
Fig. 1: Change curve of the average fitness, max fitness and standard deviation value per generation



(a) EA trained for Enemy 1 and 5



(b) EA trained for Enemy 2 and 4



(c) EA trained for Enemy 2 and 6

Fig. 2: Performance of the best individual for n = 2 and 5.

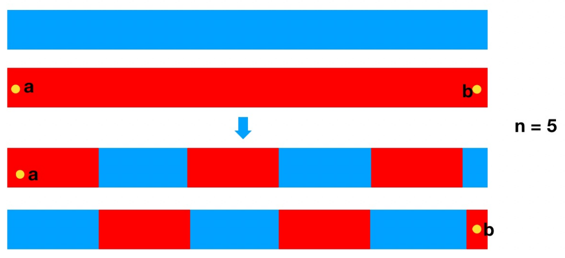


Fig. 3: N is a odd number

were the pairs to show the most promising results in the papers introducing the assignment) may have been a factor.

However, given how many parameters and variables can influence EAs, it would be of interest for future research to dig into the specifics of how each enemy influences the agents to gain a better insight into the dynamics at play, before building the findings presented here.

WORK ALLOCATION

Yuyu Bai: Designed the mutation algorithm, Wrote the Method part and helped with the plot .
Arne Sailer: Developed the selection method, structured the code and wrote the analysis part.
Willem Gerritzen: Developed the crossover function and the testing framework. Wrote the conclusion, conducted the statistical test.
Pieter Meulenbelt: Wrote the introduction and structured the paper.

REFERENCES

[1] da Silva Miras de Araujo, K., and de Franca, F. O. Evolving a generalized strategy for an action-platformer video game framework. In *2016 IEEE Congress on Evolutionary Computation (CEC)* (2016), pp. 1303–1310.

[2] Eiben, A.E., S. E. *Introduction to evolutionary computing*, 2 ed. Springer, Berlin, 2015.