The effect of different population selections in two Evolutionary Algorithms

Course: Evolutionary Computing — Assignment: Task 1

H. van Oijen (2590893)
Vrije Universiteit Amsterdam
h.van.oijen@student.vu.nl

T. Loos (2574974)
Vrije Universiteit Amsterdam
t.j.loos@student.vu.nl

T. de Leeuw (2745641)
Vrije Universiteit Amsterdam
t.de.leeuw2@student.vu.nl

Q. van der Kaaij (2557876)
Vrije Universiteit Amsterdam
q.c.vander.kaaij@student.vu.nl

1 INTRODUCTION

For this assignment we've made use of Evolutionary Algorithms (EAs). EAs are algorithms that evolve through the concept of evolution as we see it in nature. Eiben and Smith [3] describe the idea of an EA as follows: you start with a population of individuals that operate within a given environment, and this population evolves through multiple generations to create better individuals. There are two driving forces in an EA: selection and variation. Selection happens on the population level, whereas variation happens on the individual level.

Individuals are evaluated using a fitness function. Some of the best individual candidates are selected, which are used to create offspring. This is the selection part, which pushes towards quality of the population. Through mutation and/or recombination of the parents, offspring with slightly new genotypes are created. This is the variation part, which pushes to novelty of the population. Depending on the EA used, the new population consists entirely of new offspring, or partly of the offspring and partly of the previous generation. This process of selection and variation is repeated until the given number of generations is reached.

Using the Evoman framework, we will test the quality of two different EAs from the DEAP (Distributed Evolutionary Algorithms in Python) framework. The EAs will train an AI controlled player in a 2D shooting game that plays against a static enemy. The objective is to kill the enemy as soon as possible with as much remaining health left as possible. This objective is reflected in the fitness function, which will be elaborated on later. Through the use of a neural network, the EA modifies the weights for the controls of the AI player, which concretely means that it will shoot, jump, or move in a different way than before. By iterating over generations and using the best performing individuals for crossover, the fitness of the population increases.

In this research, we make use of two different EAs: (μ,λ) and (μ+λ). The μ and λ variables represent the number of individuals to select for reproduction and the number of children to reproduce respectively. The difference between the two is that the (μ,λ) algorithm selects a new population from **only** the offspring, while the (μ+λ) selects a new population from the previous generation **and** the offspring. Comparing these two algorithms logically leads to the following research question:

"Will an EA perform better in terms of 'gain' in the Evoman game when the next generation of individuals is chosen solely from the offspring or when the next generation is chosen partly from the current population?"

And looking at high computational time concerning EAs we decided to look at when an EA converges to a population of individuals that maximizes its performance in the Evoman game:

"Which of the two EAs converges faster to a population with individuals that maximize the fitness value?"

2 ALGORITHMS

2.1 Neural Network

The player of the game is controlled by a neural network. The algorithm has 20 input variables that could be of important value to create a good solution. It has a hidden layer with 20 neurons and a output layer of 5 neurons. The 265 weights between the neurons can be changed and delivers different solutions with different fitness levels. The higher the fitness level the better the solution.

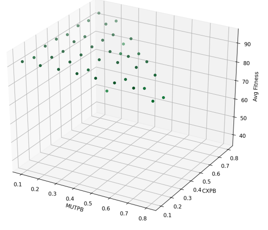

Figure 1: Tuning results of CXPB and MUTPB for μ+λ

2.2 DEAP Algorithm

As mentioned before, the DEAP framework was used to implement the EAs. The individuals of the population exist of values for the weights of the neural network. The EAs that are implemented can be used to change the weights of the neural networks through crossovers and mutations. These recombination techniques cause changes to the player behaviour and thereby the population of solutions and their fitness level. Survival of the fittest, one of the key principles of EAs, causes the fitness level of the population to rise over generations of and thereby the solution to improve. The DEAP framework was chosen because of two main reasons [4]:

- The great compatibility with python
- It offers flexibility and is easy to customize

(μ+λ) vs. (μ,λ)

The differentiation between the algorithms is in the fitness-based survivor selection method. μ stands for the current generation where the size is left to 100, which is the standard value of the algorithm.[4] λ is the group of children that are off-springs from the current generation that underwent crossover and mutation. The group size of λ is 300, which is 3x the size of μ because the literature describes that λ should be larger than μ. But there is a trade-off between λ and computational time, therefore we chose a value of 3. [5] In the (μ+λ) algorithm the best individuals of the children and the parents are selected to continue to the next generation. In the (μ,λ) algorithm only the best individuals of the children are selected to go to the next generation. The best individuals are selected through a tournament selection with tournament size of 3, which is the standard value of the DEAP framework.[2]

2.3 Parameter tuning

To increase the chances of success for both our algorithms, we made use of parameter tuning. The two parameters that we tuned were *CXPB* and *MUTPB*: The probability that an offspring is produced by crossover and that it is produced by mutation respectively. Since the sum of these probabilities must be equal or less than one, we did a constrained grid search for all possible combinations of these parameters with step-size 0.1. For the grid search the population was set to μ=15 and the offspring group size was set to λ=45. We chose to use the tripled value of the population for the

	CXPB	MUTPB
(μ,λ)	0.2	0.2
(μ+λ)	0.4	0.2

Table 1: Optimal values for CXPB and MUTB

size of the offspring group [3]. The number of generations was set to NGEN=10 to keep time for the tuning process reasonable. CXPB and MUTPB for (μ,λ) and (μ+λ) algorithms were optimized separately to make sure both algorithms work optimally and thus the effect of how to produce offspring is measured most precisely. After 10 generations the maximum of each generation was used to calculate the average of the maxima over 10 generations for each combination of CXPB en MUTPB. These averages were plotted in a 3d grid. Figure 1 represents the parameter tuning grid of the (μ+λ) algorithm. Note that the same grid was created for (μ,λ). From the grids we subtracted the optimal values for CXPB and MUTPB for both algorithms (Table 1).

2.4 Experimental setup

To answer our research question, we test both EAs against 3 different enemies and examine which EA performs best. For both EAs, we begin with a population size of μ = 100 individuals and evolve the population for 20 generations, with λ = 300 offspring in each generation. After 20 generations we terminate the evolution and select the individual with the highest fitness across all generations. We use 20 generations to limit computation time, as for all three enemies the maximum fitness is reached in less than 20 generations. This is tested by running 3 short experiments with μ=15 and λ=45 before we ran the actual experiments. We use the following fitness function to evaluate the individuals:

$$fitness = 0.9 * (100 - e) + 0.1 * p - log(t) - e \qquad (1)$$

Where p is player life, e is enemy life and t is time in seconds. Subtracting enemy life from the fitness function results in a penalty if the player is defeated. This experiment is repeated 10 times, resulting in 10 best individuals. The average of the mean and maximum fitness for each generation across 10 runs are plotted in a single line graph for both EAs. To test if our individuals are able to consistently beat the enemy, we let each individual play against the enemy 5 times. We calculate the average gain over these 5 runs, where gain is specified as:

$$individual\ gain = player\ life - enemy\ life \qquad (2)$$

We compare the difference between the individual gain for our 10 best individuals of both EAs in a box plot, showing the minimum, first quartile, median, third quartile and maximum of the individual gain distribution. The individual gain reported here is the average of 5 runs for each individual. To be able to test if the difference in individual gain is statistically significant, we perform a Shapiro-Wilk test on the distribution of the individual gain. If the sample of individual gains is normally distributed, we perform a two-sample t-test on equality of the mean individual gain of the EAs. If not, we perform a Wilcoxon rank on equality of the mean individual gain.

enemy	mean(+)	mean(,)	std(+)	std(,)	t-stat	p-value
2	60.80	62.32	21.38	16.22	25.5	0.84
5	65.92	72.24	18.80	21.47	19.0	0.39
8	60.31	22.22	7.94	37.03	7.0	0.04

Table 2: Wilcoxon signed rank test results

3 RESULTS

3.1 Algorithm Evolution

Figure 2 presents the yielded fitness of both algorithms during evolution battling the three enemies. The figures for evolution with Enemy 2 and Enemy 5 display an increase in the mean avg fitness which flattens around 6 or 7 generations. The mean avg of both algorithms with Enemy 8 reaches its maximum after 14 generations, Although the increase in mean avg fitness is small after generation 8. When The mean average fitness for (μ+λ) and (μ,λ) meet the max avg lines it means that that the EA evolved to its maximum. In previous generations an individual that (accidentally) achieved a high fitness was kept for evolution.
A notable result presented in Figure 2 is that the meas std of (μ,λ) is bigger during the evolution with all three enemies. This indicates that the resulting fitnesses achieved during the the evolution process are more widely spread in relation to the mean fitness value. The difference in mean std can be explained by the nature of the (μ,λ) algorithm, as the population is solely selected from the offspring and therefore a more varied population of individuals after each generation.

3.2 Best Individuals

In Figure 3 the three boxplots present the distribution of the average gain individual of the best individuals that played against the three enemies. The individuals that played against enemy 2 seem not to differ much from each other, although the (μ+λ) distribution has a higher positive skew. For the individuals that played against enemy 5 it seems that (μ,λ) has a more constant gain than (μ+λ) has. Although, (μ,λ) has two outliers that indicate a performance that was not in line with the average performance. When examining the performance of the best individuals against Enemy 8 it seems that (μ+λ) outperforms (μ,λ). However, a higher average individual gain has been achieved with (μ,λ), but if one looks at the average performance the mean represents a higher gain for (μ+λ).

The sample of (μ,λ) was not normally distributed after conducting a Shaprio-Wilk test, therefore a Wilcoxon signed rank test was executed to investigate if the differences between the mean of the gain of the best individuals differ significantly. The results of the Wilcoxon signed rank test are presented in Table 2. From Table 2 one can conclude that there is a significant difference between the mean of the gain of (μ+λ) and (μ,λ) as $p<0.05$. Also, the std(+) for enemy 8 is relatively high, which is in line with the boxplot in Figure 3 at enemy8 for algorithm μ,λ where the boxplot has a relatively wide wide spread over the y-axis compared to the other boxplots.

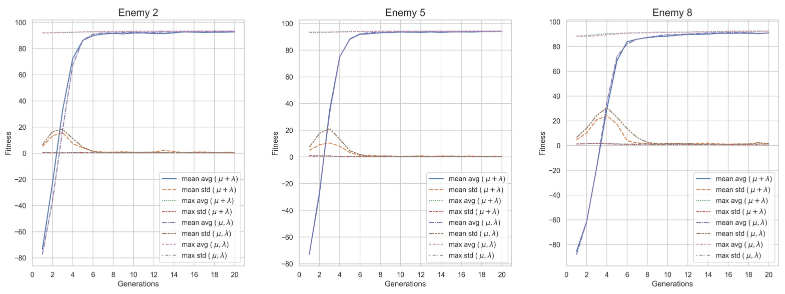

Figure 2: Linecharts with average development over different runs of mean and standard deviation of the fitness against enemy 2, 5 and 8

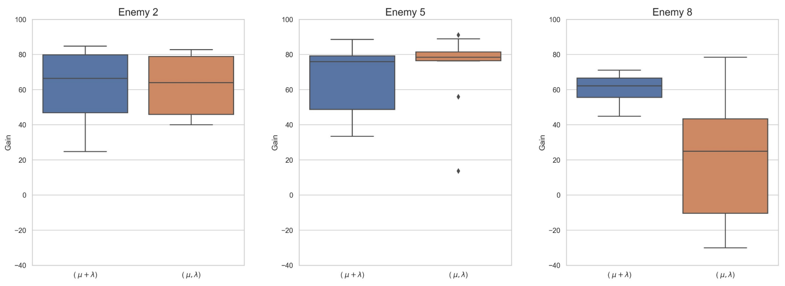

Figure 3: Boxplot with average gain of best solutions against enemy 2, 5 and 8 tested 5 times

4 DISCUSSION & CONCLUSION

4.1 Discussion

Consistent with the baseline paper [1], we succeed in defeating three enemies by training individual players with an EA. For each enemy, the algorithms find a best individual that is capable of beating the enemy. Only for enemy 8 our results differ from the baseline paper, since the (μ,λ) strategy produces some individuals that are not able to consistently beat the enemy. Due to a lack of time we did not tune other parameters than CXPB and MUTPB, and used the standard DEAP parameter values. Therefore, we can not conclude that the chosen parameters result in the best possible outcome of the algorithms. We added a penalty to the fitness function by subtracting remaining enemy life, which is not present in the baseline paper, but did not examine whether this penalty leads to faster convergence or a higher fitness level.

4.2 Conclusion

Based on the results, we concluded there is no significant difference in the performance of (μ+λ) and (μ,λ) against enemies 2 and 5. However, based on the statistics of the Wilcoxon signed-rank test, we concluded that there is a significant difference of the mean gains of (μ+λ) and (μ,λ) when the best individuals play against enemy 8. we could not conclude if one of the two EAs converged earlier or later to its maximum, since they converge very similarly. However, it can be seen that both EAs converge after more generations when playing against enemy 8, compared to enemy 2 and 5. However, this experimental set-up was not designed to investigate the overall difference in performance of the EAs on the the three enemies. Recommended, is to investigate this in more detail in further research.

5 CONTRIBUTION

5.0.1 Tiddo Loos. Concerning the coding part of the assignment, I contributed to the parameter tuning experiment and running a part of the main experiment. Also, I coded the program for the statistical results. In the report I mainly focused on the results and the conclusion section.

5.0.2 Hidde van Oijen. In the coding part, I mostly worked on retrieving the results and tuning the parameters. Next to that, I helped merging the Evoman and DEAP framework. In the report, I mainly focused on explaining the different algorithms and parameters.

5.0.3 Tom de Leeuw. In the coding part of the assignment, I mainly focused on the DEAP framework. I also coded the linecharts and boxplots. In the report I contributed to the experimental setup and the discussion.

5.0.4 Quinten van der Kaaij. Concerning the coding part, I mainly focused on researching implementations of the DEAP framework and the parameter tuning. For the report my main contributions were the parameter tuning and the introduction.

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)*. 1303–1310.

[2] François-Michel De Rainville, Félix-Antoine Fortin, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. Deap: A python framework for evolutionary algorithms. In *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*. 85–92.

[3] A. E. Eiben and J. E. Smith. 2015. *Introduction to Evolutionary Computing* (2nd. ed.). Springer, New York, NY.

[4] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary algorithms made easy. *The Journal of Machine Learning Research* 13, 1 (2012), 2171–2175.

[5] Thomas Jansen, Kenneth A. De Jong, and Ingo Wegener. 2005. On the Choice of the Offspring Population Size in Evolutionary Algorithms. *Evolutionary Computing* 13, 4 (2005), 413–440.