The effect of different population selections in two Evolutionary Algorithms

Course: Evolutionary Computing — Assignment: Task 2

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 are expanding the research we did in the first assignment [4]. We've explored EAs, how they work and how to apply them in the Evoman framework. We tested two different EAs: (μ,λ) and (μ+λ). The (μ,λ) algorithm selects a new population from **only** the offspring, leading to higher variation. On the other hand, the (μ+λ) selects a new population from the previous generation **and** the offspring. We concluded that there was no significant difference in performance of the two EAs. The mean gain however was better for the (μ+λ) algorithm when training against a single enemy.

In this research, we will again test two different algorithms from the Distributed Evolutionary Algorithms in Python (DEAP) framework: (μ+λ) since it had better mean gain, and a Covariance Matrix Adaptation Evolution Strategy (CMA-ES).

One of the problems of extensively researching the optimal solutions in the first assignment was the long running time for parameter tuning and running the EA itself, which increases even more with higher population. Since CMA-ES works better with a lower population size and updates the parameters itself [2], it would be a good contender against the (μ+λ) algorithm.

Not only do we research a different algorithm, the goal will also be different. In the first assignment we trained an EA to compete against a single enemy, whereas now the EA will have to generate a solution that will play against multiple enemies.

Considering these changes in the setup, we ended up with the following research question: "Which of the two algorithms, (μ+λ) or CMA-ES, will perform better when training a generalist agent in the Evoman framework against all enemies?". Since performance is broadly interpretable, we've conducted two sub-questions:

- Which of the two algorithms will perform better in terms of "gain"?
- Which of the algorithms will perform better in terms of running time?

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 deliver different solutions with different fitness levels. The higher the fitness level the better the solution.

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 crossover and mutation. 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 [1]:

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

2.3 (μ+λ)

The (μ+λ) algorithm is named after the fitness-based survivor selection method it uses. μ stands for the current generation which has a size 100. λ is the group of children that are off-springs from the current generation that underwent crossover and mutation. The size of λ is set to 300, which is 3x the size of μ because the literature describes that when λ needs to be bigger than μ. But there is a trade-off between λ and computational time, therefore we chose a value of 3.[3] The (μ+λ) algorithm selects the best individuals of the children and the parents to the next generation, where in other algorithms like (μ+λ) this is not the case. The selection procedure that is used to select the individuals is the tournament method. In this method, x number of random individuals compete where one is selected to advance to the next generation. This is done until the next generation is full of best individuals. The amount of random individuals is determined by the tournament size, which is set to a value of 3.

2.4 CMA-ES

A CMA-ES is used to control the evolution parameters during the evolution [2]. This algorithm has several advantages over the previously explained (μ+λ) algorithm:

- Relatively "parameter free" and does not require extensive parameter tuning
- Works well with a lower population size

The CMA-ES starts with a single individual, an NxN covariance matrix, and the initial mutation step size σ. At every generation, λ=N offspring are sampled from a multivariate gaussian distribution and each individual is evaluated. The individuals are sorted based on their fitness and then used to update the distribution parameters; The parent for the next generation is the weighted mean of the μ best individuals from the current generation and the covariance matrix is updated. This process repeats until the desired number of generations is reached.

2.5 Parameter Tuning

2.5.1 MUTPB & CXPB:. To improve the performance in terms of gain, parameter tuning was used. The first parameters that were tuned were MUTPB & CXPB. These parameters stand for the probability that an offspring is produced by mutation and the probability that its produced by crossover respectively in the (μ+λ) algorithm. Since the sum of these probabilities must be equal or less than one, a constrained grid search was performed 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. The tripled value of the population was chosen for the size of the offspring group.[2] The number of generations was set to NGEN=10 to keep time for the tuning process reasonable. 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 and MUTPB. These averages were plotted in a 3d grid. Figure 1 represents the parameter tuning grid of the

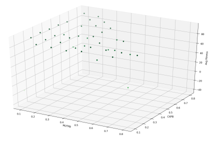

Figure 1: Tuning results of CXPB and MUTB for enemy 2 and 5.

(μ+λ) algorithm. The optimal value for MUTPB and CXPB are 0.1 and 0.3 respectively.

2.5.2 μ & Tournament size: Besides tuninng MUTPB & CXPB for (μ+λ), an attempt has been made to optimize μ and Tournament size. As Tournament size cannot be greater than μ, a constrained grid search was performed for all possible combinations of these parameters with step-size of 20. The combinations were selected from the ranges 20 to 160 for μ and from 10 to 150 for tournament size. Not every value between the ranges was tested due to lack of computational power and time. For the grid search the MUTPB & CXPB were set to their earlier found optimal values of 0.1 and 0.3 respectively and λ was always set to the triple value of μ. The number of generations was set to NGEN=10 to keep time for the tuning process reasonable. After 10 generations the maximum of each generation was used to calculate the average of the maxima over 10 generations for each combination of μ and Tournament size. We found a highest fitness value of 93.33 with a value of 140 for μ and 90 for Tournament size. But given that the grid search gave an average value of 92.24 and the values don't show a clear trend, the results were found not significant enough to use. Therefore, another search was conducted with tournament size ranging from 1 to 10 with a μ of value 100. But this did not lead to significant results either. Based on this research it was decided to keep the values of μ and Tournament size on their standard values of 100 and 3 respectively.

2.5.3 CMA-ES:. As described in Hansen[2], the values of the selection parameters are relatively unimportant to the evolution process. Therefore we chose the default values for the selection parameters λ (=4+3*ln(n), where n is the size of an individual), and μ (=λ/2). The parent weights are superlinear by default. Due to time limitations, we did not tune any of the other strategy parameters. However, the default values for these parameters are good choices in most setups with a relatively easy objective function[?]. The initial step size σ has to be chosen carefully. We found that setting $\sigma < 2.0$ leads to low variation and slow convergence to the maximum fitness. The average fitness achieved with a low initial step size is lower compared to higher initial step sizes ($\sigma > 2.0$). We also found that for $\sigma > 4.0$, the results of the evolutionary process did not change significantly. Therefore we chose to set the initial step size to 4.0.

2.6 Experimental Setup

To answer our research question, we train both EAs on two different groups of enemies and test the best individuals of each EA against all 8 enemies. The (μ + λ) strategy again starts with μ = 100 and λ = 300. The CMA strategy starts with λ = 20 and σ = 0.4. During the training phase, we select individuals based on the equally weighted average fitness of the individual against the enemies in the training group. The fitness function is now defined as:

$$fitness = \frac{\sum_{k=1}^{K}[(0.9*(100-e_k)+0.1*p_k-log(t_k)-e_k]}{K} \quad (1)$$

Where p is player life, e is enemy life, t is time in seconds and K is the number of enemies. The standard deviation of the fitness values of one individual against the enemies in the training is subtracted from the average fitness, to account for individuals that only perform well against one of the enemies in the training group. 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 per enemy group for both EAs. To test if our individuals are able to beat multiple enemies, we let each best individual play against all 8 enemies 5 times. We calculate the average gain over these 5 runs, where gain is specified as:

$$individual\ gain = \sum_{k=1}^{K}[player\ life_k - enemy\ life_k] \quad (2)$$

where K is the number of enemies. We compare the difference between the average of 5 runs of the 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. 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.

After comparing the performance of both EAs, we also compare running time. One of the downsides of the (μ + λ) strategy was the high amount of computational time to run the algorithm. If both algorithms have equal performance in terms of gain and fitness, we favor the algorithm with the shortest time to run.

3 RESULTS

3.1 Algorithm Evolution

When investigating Figure 2 and 3, one can observe a difference in conversion of the mean avg fitness value for both (μ + λ) and CMA-ES. What can be seen is that the evolution against the enemy set of 6 and 8 seems to be less successful after 20 generations. Also, after 20 generations it seems not to be totally converged to its maximum mean avg level. When looking at the mean avg fitness value of (μ + λ) in the evolution against enemy set 2 and 5, one can see that it converges to its maximum mean avg level after 9 or 10 generations. When comparing the increase of the mean avg lines of both algorithms one can see that the CMA-ES mean avg lines fluctuates. The fluctuation in the CMA-ES mean avg fitness value can be observed in the evolution against both enemy sets. Another results is that the mean avg performance of (μ + λ) converges to a higher fitness value when evolved against both enemy sets.

3.2 Performance Against All Enemies

The boxplots in Figures 4 and 5 indicate that the overall performance of the both algorithms yields a negative gain. A negative gain indicates that the algorithms on average lose against the enemies.

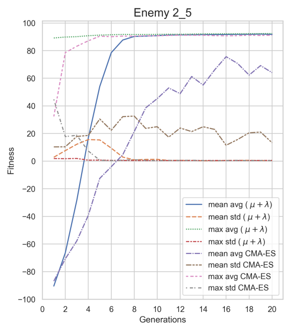
Figure 2: Evolution against enemies 2 and 5.

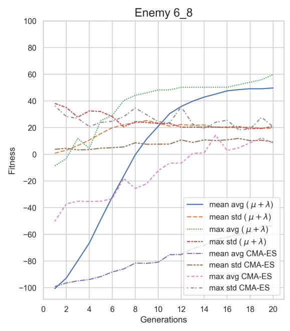
Figure 3: Evolution against enemies 6 and 8

enemy set	u(+)	$\mu(C)$	$\sigma(+)$	$\sigma(C)$	t	p
2, 5	-191.28	-235.47	42.19	75.08	-1.54	0.14
6, 8	-261.54	-217.43	79.25	63.56	1.30	0.21

Table 1: Statistic of compared gains of $(\mu + \lambda)$ (+) and CMA-ES (C) against all enemies.

However, it is hard to see if and when a win against an enemy occurs.

When investigating the boxplot in Figure 4 it seems that the CMA-ES algorithm performs better against all enemies as the median of the gain is higher than the median gain of $(\mu + \lambda)$. in Figure 5 the median gain of $(\mu + \lambda)$ is higher compared to the CMA-ES algorithm. A Shapiro-Wilk test on these results provides no evidence that the samples are not normally distributed. When examining the statistical results in Table 1, one can see that the p-values, when comparing the mean of both gain samples, have a p>0.05. Therefore, there is no evidence that one of the two algorithms outperforms the other. Note that $(\mu + \lambda)$ and CMA-ES are abbreviated with (+) and (C) respectively in Table 1.

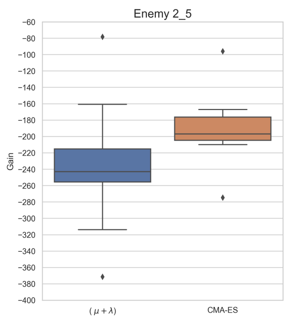
Figure 4: Boxplot results against all enemies: $(\mu + \lambda)$ and CMA-ES are trained on enemies 2 and 5.

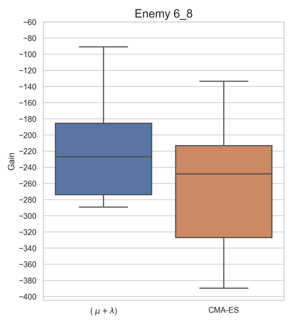
Figure 5: Boxplot results against all enemies: $(\mu + \lambda)$ and CMA-ES are trained on enemies 6 and 8.

algorithm	enemy set	time (m)
$(\mu + \lambda)$	2, 5	341
CMA-ES	2, 5	53

Table 2: computational time of evolution of both algorithms against enemies 2 and 5

3.3 Computational Time

The results of the computational time experiment are shown in Table 2. The Table indicates that the total computational time for the CMA-ES was ≈ 4 times less than for $(\mu + \lambda)$.

4 DISCUSSION & CONCLUSION

4.1 Discussion

The comparison of $(\mu + \lambda)$ and CMA-ES in terms of computational time is questionable. The design of the CMA-ES algorithm is responsible for the lower computational time. The CMA-ES algorithm starts with a population of $\mu = 1$, where $(\mu + \lambda)$ starts with a population of $\mu = 100$ at generation 0. Furthermore, $(\mu + \lambda)$ relies on tournament selection and two-parent crossover, while CMA-ES uses a weighted combination of the best individuals to create a single parent for the next generation. These design characteristics result in different optimal settings for the population size. Therefore, the comparison is questionable but it does give an insight in the gain that can be yielded in a particular time span.

4.2 Conclusion

When examining the results there is no evidence to conclude that either $(\mu + \lambda)$ or CMA-ES outperforms the other in the current experimental setup. The CMA-ES is faster in this setup, although the computational time comparison is questionable. The resulting yielded gains do not show a significant difference when comparing the samples of $(\mu + \lambda)$ and CMA-ES. Although the results of the avg mean during the evolution of $(\mu + \lambda)$ against both enemy sets show that $(\mu + \lambda)$ converges faster to a higher fitness value than CMA-ES, there is no evidence to conclude that $(\mu + \lambda)$ performs better in terms of gain against all enemies.

5 CONTRIBUTION

5.0.1 Tiddo Loos. Coding part: Parameter Tuning, Running and plotting the results of the $(\mu + \lambda)$ algorithm. Report: Responsible for the Results and Discussion & Conclusion sections.

5.0.2 Hidde van Oijen. My contribution in the coding part mostly consists of performing the parameter tuning. In the report, I mainly focused on the explaining the algorithms and parameter tuning.

5.0.3 Tom de Leeuw. Coding part: Setting up and running CMA-ES. Report: Experimental setup, CMA-ES parameter tuning.

5.0.4 Quinten van der Kaaij. Coding part: Parameter tuning. Report: Introduction

REFERENCES

[1] 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.

[2] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation* 9, 2 (2001), 159–195.

[3] 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.

[4] Q. C. van der Kaaij, T. de Leeuw, T. Loos, and H. van Oijen. 2021. The effect of different population selections in two Evolutionary Algorithms. *Course: Evolutionary Computing* Assignment: Task 1 (2021).