Evolutionary Computing Course
Task 1: specialist agent
Team 68
01/10/2021

Michał Butkiewicz
Student nr: 2730938

Antonio Correia
Student nr: 2748054

Gabriel Hoogerwerf
Student nr: 2735225

Andrzej Szczepura
Student nr: 2744488

ABSTRACT

Two Evolutionary strategies are presented and compared based on the in-game performance of an agent whose behaviour is influenced by the best results found in the evolution process. Both strategies make use of the same recombination and mutation techniques but present two different selection methods: respectively (μ,λ)-selection and (μ+λ)-selection.

Performance analysis of different selection types in Evolution Strategies for evolving the behaviour of game-playing agent

1 INTRODUCTION

Evolutionary approaches are often applied to perform the optimization of game-playing learning agents. Specifically, gradientless optimization algorithms are used with a vector of weights as a solution, to be consequently applied to the neural network controlling the in-game behaviour of the agent. Such a vast search space combined with the difficulty of implementing any kind of heuristics or constraints to the solutions make this problem a great benchmark for Evolutionary Algorithms.

In this report, two Evolutionary Strategies are introduced and their overall performances compared. Specifically, our research question revolves around the presence of any statistically significant differences in fitness between the EAs with different types of selection (plus selection and comma selection). Testing and evaluation were performed using the EvoMan framework [2], an environment for evolving game-playing agents, based on the classic 2D action game Mega Man II. The full description of the framework can be found in [2].

2 BACKGROUND AND RELATED WORK

Evolution Strategies (ES) are popular algorithm variants used in Evolutionary Computing (EC). They were first introduced by Rechenberg and Schwefel in the early 1960s [6]. Schwefel introduced two versions of multimembered ES [5], i.e.,

1. the (μ + λ)-ES, in which not only one offspring is created at a time or in a generation, but $\lambda \geq 1$ descendants, and, to keep the population size constant, the λ worst out of all μ + λ individuals are discarded;

2. the (μ, λ)-ES, in which the selection takes place among the λ offspring only, whereas their parents are "forgotten" no matter how good or bad their fitness was compared to that of the new generation. Obviously, this strategy relies on a birth surplus, i.e., on $\lambda > \mu$ in a strict Darwinian sense of natural selection.

The strategy-specific parameters μ and λ are exogenous parameters, which means they are kept constant during the evolution run. Using this notation it is agreed that μ stands for *the* number of parent individuals and λ for the size of the offspring population. Throughout the whole report the terminology introduced by Schwefel [2] will be used, that is **plus selection** denoted by (μ + λ) and **comma selection** denoted by (μ, λ).

3 ALGORITHMS DESCRIPTION

3.1 Algorithms operation outline

To ameliorate the understanding of how the algorithms execute operations we advise consulting the pseudocode presented in *Algorithm 1.* where the difference between the algorithms operations can be easily seen.

```
1: create an initial population of μ random individuals
2: FOR c in generations:
3:         create λ offspring (λ >μ):
4:                 intermediate recombination with n individuals
5:                 gaussian mutation of λ offspring
7:         CASEWHERE experiment condition is:
8:                 (μ+λ)-ES: select μ best individuals from λ+μ
9:                 (μ, λ)-ES: select μ best individuals from λ
10:        ENDCASE
11: END FOR
```

Algorithm 1: Pseudocode for both of the EAs

Firstly the initial population is created, consisting of a number of individuals whose genome is randomly generated. All of the initial individuals are kept and used in the creation of the offspring. In order to generate new individuals both recombination and mutation are implemented. The recombination strategy adopted is intermediate recombination [7]. It's worth mentioning that the mutation step size is also going to be inherited at this point, as the average value of the parents' mutation step size. Once the child is successfully created, its genome is subjected to mutation, according to the inherited and mutated strategy parameter δ (randomly initialized for the very first generation). An array of samples from the Gaussian distribution is created, which is later multiplied by δ and added to the child's genome. The mutation step size itself is also subjected to variation, by being multiplied by N(0.τ), where τ is equal to $1/\sqrt{v}$ with v being the number of variables (genes) in our solutions. This implemented mutation procedure is a standard technique used in Evolution Strategies, described thoroughly in [5][4]. Deterministic selection is performed afterwards, either on the offspring population, or on the combined offspring and parents population, depending on the version of the algorithm. Each chromosome is evaluated and the top μ chromosomes are passed onto the next generation. The cycle is repeated with the resultant population c times - the number of iterations is a predefined hyperparameter.

3.2 Parameters

Table 1: Parameters set for the experiments.

Population Size (μ)	Offspring size (λ)	Number of generations (c)
100	400	20

Table 1. one shows the parameters adopted for the algorithm. The proportion between number of offspring and population size sets the selective pressure quite high, as it is advisable with evolutionary strategies [5].

4 EXPERIMENTAL SETUP

The simulation was initialised in individual evolution mode for a single static enemy. To train the specialist agents, the EAs tried to find solutions for each one of the enemies separately (chosen enemies: 2,4,7). Default fitness function from the EvoMan framework was used. Thus, solutions that will take too long to defeat the enemy will have a lower chance to move on to the next generation. The neural network demo_controller.py was used to perform the game simulations. Both algorithms were run 10 times for each of the chosen enemies (independently) and the results were subjected to statistical analysis.

5 EXPERIMENTAL RESULTS

5.1 Statistics

Independent-samples t-tests were conducted to compare the average mean values of 5 fitness values of the best individual among all generations for each run in the plus selection and the comma selection conditions for each enemy.

5.1.1 Enemy 2. No significant difference was observed in the results for plus selection (M=92.23, SD=1.42) and comma selection (M=89.86, SD=8.38) conditions for Enemy 2; t(9.5199)=0.88125, p = 0.3999

5.1.2 Enemy 4. A borderline significant difference was observed in the results for plus selection (M=73.88, SD=16.19) and comma selection (M=60.1, SD=13.72) conditions for Enemy 4; t(17.528)=2.0585, p = 0.05473

5.1.3 Enemy 7. No significant difference was observed in the results for plus selection (M= 71.37, SD=13.04) and comma selection (M=72.66, SD=12.39) conditions for Enemy 7; t(17.953)=-0.2276, p = 0.8225

5.2 Graphic representation of results

5.2.1 Line plots. Line plots show graphically mean maximum fitness values (dotted lines) and average mean fitness values (solid lines) for both EAs (red - **plus selection**, EA 1, blue - **comma selection,** EA 2) for corresponding enemies. Generation values represented in X-axis and on mean Fitness Y-axis..

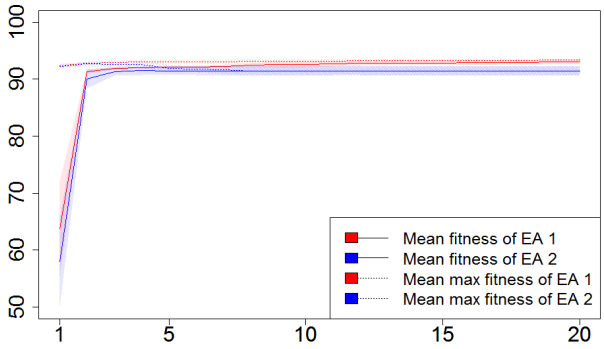

Figure 1: Line plot for enemy 2.

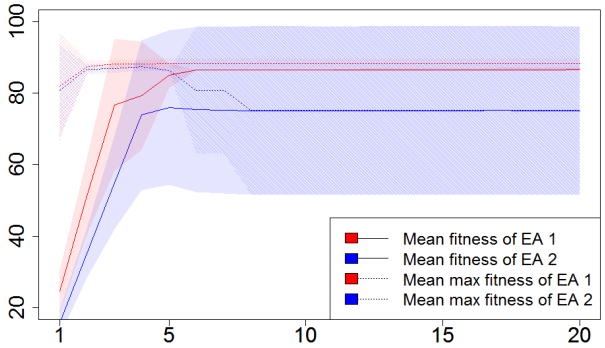

Figure 2: Line plot for enemy 4.

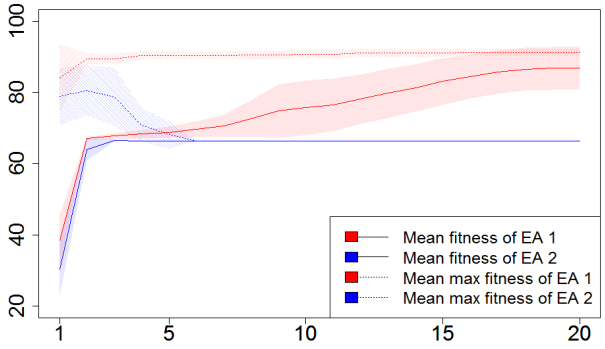

Figure 3: Line plot for enemy 7.

5.2.2 Boxplots. Boxplots show graphically the means of 5 fitness values of the best individual among all generations for each run for both EAs (red - **plus selection,** EA1, blue - **comma selection,** EA2) for each enemy. On the Y-axis there are mean fitness values.

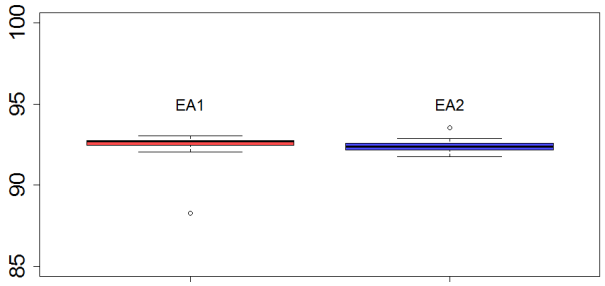

Figure 4: Boxplot for enemy 2

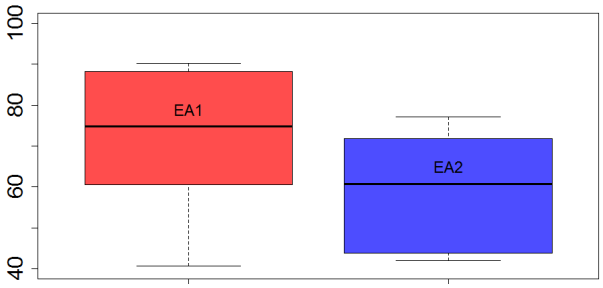

Figure 5: Boxplot for enemy 4

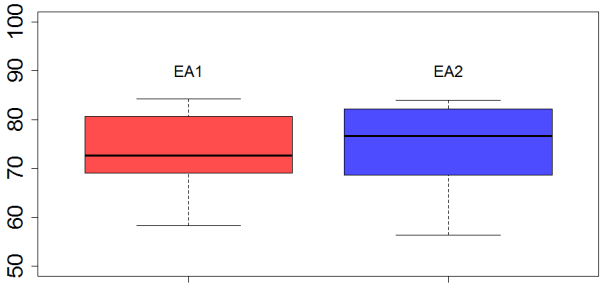

Figure 6: Boxplot for enemy 7

Table 2: Best individual final energy comparison of this research EAs to the best one from baseline paper.

Enemies	NEAT	$(\mu+\lambda)$ EA	(μ, λ) EA
AirMan (2)	94	88	90
HeatMan (4)	94	75	69
BubbleMan (7)	78	77	68

Table 2. reports the final energy of the very best agents at the end of experiment for both algorithms and the overall best algorithm (NEAT) results from the baseline [2] paper for each enemy for comparison purposes.

6 ANALYSIS AND DISCUSSION

From the line plots it can be seen that both algorithms perform similarly (against every different enemy), by increasing their mean fitnesses of every individual steeply during the first few generations and achieving stable values for the rest of the simulation. One exception for this is EA 1 performance against enemy 7, where the mean fitness value has a slower but constant increase. Standard deviation, especially in Enemy 4, is more noticeable for EA 2. This can be due to the fact that every generation is filled completely with new individuals and therefore higher diversity. Predictably, the mean maximum fitness values in Comma Selection often experienced drops from generation to generation, which is due to the fact that possible best individuals in parents were not taken in consideration for the next generation. As for $(\mu + \lambda)$ selection, as it guarantees the survival of the best individual found and since it preserves the best individual, such selection techniques can be called *elitist*. Due to the fact that parents can survive an infinitely long time span, elitism can be noticed in plus strategies, and overall it may be the factor for slightly better results than in comma selection condition.

Regarding the performance of the best individuals presented in Figure 4., $(\mu + \lambda)$ EA performs marginally better than (μ, λ) EA and both of them have managed to win against Enemy 2. On the other hand, whiskers of $(\mu + \lambda)$ EA in Figure 5. shows the best individuals being able to win with enemy 4 but not consistently throughout the 5 tests for each run. Besides this notion, it is also worth underlining that a borderline significant difference between the two algorithms was observed. Thus, the $(\mu + \lambda)$ EA can be expected to perform better than the (μ, λ) EA. Figure 6 shows EA2 having an average better fitness for it's best individual but overall performance is in the same range.

7 CONCLUSIONS

This study aimed at exploring differences between selection schemes and their influence in evolving a game-playing agent. The outcome of the experiment points to the differences in the behaviour of the two algorithms, such as the ability of EA1 to improve, even if sometimes modestly, the mean fitness of the population, or the tendency of the EA2 max mean fitness to drop throughout the time of the runs. The results also show how *plus* selection had overall slightly better performance than *comma* selection, both being able to provide often similar peak solutions (although the differences were not statistically significant). As also stated in [5], the difference between these two approaches probably fades away when the population is large enough.

What could be improved in future experiments is the variety in generations of solutions. Thus the next goal is to work on the ability of EAs to maintain a diverse set of points that could help to escape from locally optimal regions. Baseline paper [2] provided the evidence that there are multiple possible winning strategies characterizing a multimodal search space thus comma selection approach could pose a big advantage for exploring different strategies for a player agent to adopt.

8 CONTRIBUTION INFORMATION

A.Sz. with the help of M.B. developed theoretical formalism. A.Sz. plotted the analytic calculations. A.C. and G.H. and M.B. performed the numerical simulations. A.C. and G.H with the help of A.Sz. and M.B wrote the code. All authors contributed to the final version of the manuscript.

9 BIBLIOGRAPHY

[1] A. Ahrari and O .Kramer, *Finite life span for improving the selection scheme in evolution strategies*, *Soft Computing*, *21*(2):501-513, 2017.
[2] K. D. S. M. de Araujo and F. O. de Franca, *Evolving a generalized strategy for an action-platformer video game framework*, In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1303-1310, 2016.
[3] T. Bäck and F. Hoffmeister, *Basic aspects of evolution strategies*, *Statistics and Computing*, *4*(2): 51-63, 1994.
[4] H.G. Beyer, *Evolution strategies, Scholarpedia*, 2(8), 2007.
[5] H. G. Beyer and H. P. Schwefel, *Evolution strategies–a comprehensive introduction*, *Natural computing*, *1*(1): 3-52, 2002.
[6] A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing, Springer, 2003.
[7] D. B. Fogel and H. Beyer, *A Note on the Empirical Evaluation of Intermediate Recombination*, in *Evolutionary Computation*, 3(4), pp. 491-495, 1995.

10 APPENDIX

Appendix 1

ACM Reference format:

M. Butkiewicz, A. Correia, G. Hoogerwerf, A. Szczepura. 2021. Performance analysis of different selection in Evolutionary Strategies for evolving the AI-player behaviour. In *1st. Assignment of Evolutionary Computing Course, Amsterdam, The Netherlands, 2021*, 3 pages.

Appendix 2
Classes and Functions

In order to achieve a flexible and easily manageable code, two classes have been implemented in the *Chrom.py* file, respectively:

Chrom() class represents an instance of a candidate solution, along with attributes related to its performance.
Population() class represents a list of chrom() objects, as well as attributes and functions related to the population.

1: ***testing_pop***(pop) will evaluate every individual in *pop*, providing info about its performance on the game:fitness,player life, enemy life, time.

2: ***reproduction***(pop) will create an Offspring ***Population,*** giving a *pop* of parents**.** Two parents will be picked randomly from pop. With these two, we will perform **crossover**. In the end we **mutate** all the members of the offspring population.

3: ***crossover***(parent1, parent2) will create one offspring, giving two parents. The offspring *genome* and the *mut_step* will be equal to the average of its parents, respectively.