

EVOLUTIONARY COMPUTING: STANDARD ASSIGNMENT

TASK II: Generalist Agent - Team 101

Date: 18/10/2021

Christodoulou Alexandros, ID: 2739584
Vrije Universiteit Amsterdam, Faculty Science,
Artificial Intelligence
Amsterdam, Netherlands
a2.christodoulou@student.vu.nl

Christofi Konstantinos, ID: 2742715
Vrije Universiteit Amsterdam, Faculty Science,
Artificial Intelligence
Amsterdam, Netherlands
k.christofi@student.vu.nl

Hadjigeorgiou Christos, ID: 2751135
Universiteit van Amsterdam, Faculty Science,
Forensic Science
Amsterdam, Netherlands
c.hadjigeorgiou@student.vu.nl

Lamkaraf Redouan, ID: 2672658
Vrije Universiteit Amsterdam, Faculty Science,
Artificial Intelligence
Amsterdam, Netherlands
r.lamkaraf@student.vu.nl

ACM Reference Format:

Christodoulou Alexandros, ID: 2739584, Christofi Konstantinos, ID: 2742715, Hadjigeorgiou Christos, ID: 2751135, and Lamkaraf Redouan, ID: 2672658. 2019. EVOLUTIONARY COMPUTING: STANDARD ASSIGNMENT TASK II: Generalist Agent - Team 101: Date: 18/10/2021. In *Proceedings of the Genetic and Evolutionary Computation Conference 2019 (GECCO '19)*. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
GECCO '19, July 13–17, 2019, Prague, Czech Republic

© 2019 Copyright held by the owner/author(s).
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00
<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

The purpose of the research is to evaluate the effect of two Evolutionary Algorithms (EAs) with different initialization strategies and operators regarding the mating population. A generalist agent is trained where the objective function is defined as the weighted sum of the default fitness function of the Evoman framework [4] [7] of the enemies fought and the performance of the agent-based player is assessed while being evolved with increasing generations in order to beat a specific group of enemy players. The differences between the standardized pipelines of EAs outlined in this paper are highlighted in the following research question:

Research Question - "How are the results of Evoman framework generalist agent influenced by an Evolutionary Algorithm with random uniform initialization in comparison to an Evolutionary Algorithm with hybridized version of seeded population and with different strategies regarding mating and survival population?"

To answer the question, three measurable success criteria have been considered: the mean and the maximum fitness of the population of individuals per generation of all runs per training group, the resulting gain of the best individual of all generations per run trained against all enemies per training group and the Wilcoxon test [9] applied on the resulted gain for each training group per algorithm. The first two are presented in line plots while the last one is applied on the arrays of the box plots as shown in the results, see Section 4.

2 RELATED WORK

The Evoman framework has been utilized to simulate and test the player evolution using a Genetic Algorithm, the LinkedOpt Algorithm and the NEAT Algorithm and their results are compared using an objective function [2, 3]. Based on mentioned work, this paper focuses on two GAs where the influence of different initialization is to be explored and different strategies for mating and survival population with respect to the fitness function and compare the results produced with the mentioned papers.

3 METHODS

This section outlines the methodology followed and the reasoning for certain choices. This way, readers can follow and replicate the steps performed obtain similar results and build on using this paper as a reference.

3.1 Budget

Limited time is faced in every assignment which is further amplified by the repetition of experiments due to the inherent stochastic nature of an EA and the random initialization parameter feature of the Evoman framework [7]. As indicated by Fig. 8 about 49 hours are required for the first algorithm and about 25 hours for the second, see Fig. 9 for two group of enemies per experiment where each group has two enemies. This report also indicates the time taken for 3 enemies per group for the second algorithm, see Fig. 10, but due to time restriction this was not possible for the first algorithm. Even though the 3 enemies provide better overall gain, this was not tested further due to time. As indicated by the paper [6], no parameter tuning was performed when choosing certain parameters and could provide better results. More parameter tuning

would have resulted in better results but due to time and hardware limitations, it is left for future scope.

3.2 Algorithm Setup

An EA algorithm consists of 5 phases: representation stage, initialization phase, mating population and survivor selection phase, the crossover and the mutation operators and the termination phase [5, 10].

An EA algorithm uses the concept of Darwinian evolution to increase the average fitness of the population by prioritizing the fitter individuals [10]. Throughout the algorithm, conflicting concepts like exploitation and exploration [10] are evaluated and weighted accordingly as finite amount of resources can be allocated in analyzing the search space. The former relates to finding new unexplored regions to find the global optimum which would optimize the solution with respect to a fitness function compared to the latter which focuses resources on specific hills to discover the optimal point of that region.

3.3 General Algorithm Considerations

The first step of any EA is to choose the representation between the phenotype and genotype and will determine the quality of the solution. The representation in this paper relates to a one-to-one mapping where the phenotype is the neural network that controls the player agent and the genotype is a 1D-array of 265 elements-neurons that is fed to the neural network [3] and an additional value extending the individual that corresponds to the value of mutation step size which is unique to each one. The individuals correspond to single solutions where a fitness function evaluates how good an individual is. As the agent is a generalist, it takes a number of enemies as a list to train against with the fitness function to be maximized:

$$fitness' = \bar{f} - \sigma \quad (1)$$

with the default fitness function used as defined in the original paper [7].

A modification is performed to Eq. 1 where the subtraction of standard deviation is disregarded as it was favouring smaller values and a weight is introduced as shown below:

$$fitness' = 0.6 * f_{e_1} + 0.4 * f_{e_2} \quad (2)$$

with f_{e_i} being the fitness obtained from the enemy i in the training groups of size 2.

The same parameters are used with the baseline paper [3] to be able to compare the results namely a population of 100 individuals evolved for 30 generations with a repetition of 10 times to compensate the stochastic nature of the EA.

3.4 Initialization Phase

This section is different for the two algorithms as the first loads 10 specialized agents who are specifically trained for each of the 8 enemies and to reach 100 individuals, the remaining individuals are drawn from random uniform distribution with limits of -1 and 1 while the second algorithm creates a population of 100 individuals drawn from the same random uniform distribution

Afterwards, the population is evaluated to get the mean fitness and the maximal fitness of the population of each generation. Then

the iterative process of evolutionary operators: selection of mating population and of survivor population, recombination and mutation begins until the termination conditions are met.

3.5 Evolution Algorithms

3.5.1 Mating / Parental Selection Phase.

In the parental selection phase, the first algorithm makes use of a Memetic algorithm which performs a local search operator namely bit flip to a number of elements of the array of an individual on all individuals except the best of the population as described in [6]. As described in the paper, simulated annealing is considered, the temperature is inversely proportional to the difference of maximum fitness and the average fitness of the population where initially the difference is larger than in subsequent generations. This impacts the selection pressure as worse modified individuals can be accepted according to the Boltzmann distribution and in the initial generations favour the exploration phase while in later generations the exploitation is preferred. Similarly k-tournament selection is used to create the mating population of the size of the original population and selection pressure is increased as generations increase which in turn increase the value of k.

On the contrary, the second algorithm makes use of the Fitness Proportional Selection (FPS) mechanism is selected to pick the parents which will be part of the mating population. This process considers the fitness value of the individuals as indicated by Eq. (3).

$$P_{FPS}(i) = \frac{f_i}{\sum_{j=1}^{\mu} f_j} \quad (3)$$

It is incorporated with sigma scaling which provides information about the average and the standard deviation of the array of individuals for better representative selection [10] and is implemented as indicated by the formula in page 81 [10].

Normalization of array is performed with the function 'Min-MaxScaler' of the python library 'sklearn.preprocessing' [8] and values below a threshold are given the value of the threshold. This way, worse individuals can still be selected with a small as the EAs are stochastic algorithms [1].

3.5.2 Survivor Selection Phase.

In this section, the first algorithm replaces the entire population with the new offspring created by the mating population compared to the second algorithm which makes use of sigma scaling FPS again for stochastic selection of individuals and prioritizes the worse individuals in terms of fitness. The modification is that the range of the array is computed and each fitness value is reduced by that value. By taking the absolute value of the entire array as there are already negative values to subtract from, the originally small values will be prioritized as they become the large ones. Afterwards, a division by the total fitness of the array will scale the values and these weights are used to sample the individuals.

3.5.3 Crossover/Recombination Operator.

For the crossover step, the whole arithmetic recombination is implemented for both algorithms where two parents produce two children given an α parameter that distinguishes the two offspring. The α parameter is drawn randomly from uniform distribution ranging from 0 to 1 and limits the possible region of the offspring in the search space to the region bounded by its parents with the

equation implemented taken from the respective section of page 66 of the book [10].

For the first algorithm, crossover procedure is probabilistic with a probability of 80% as indicated in the paper [6] while for the second algorithm it is guaranteed. If the random probability drawn from uniform distribution is greater than the defined threshold then the two parents become the two offspring.

3.5.4 Uncorrelated mutation with one sigma.

For this section, the same procedure is kept for both algorithms where an individual will undergo mutation process if the random uniform probability drawn of that individual with limits between 0 and 1 is lower than the mutation probability of 5% as indicated by [6].

Each individual chromosome is extended by a σ value which will dictate the mutation step size for the log-normal mutation operator. This mutation step size is multiplied by an exponential term that is drawn randomly from normal distribution with parameters of $\mu = 0$ and $\sigma = 1$, multiplied by the learning rate which is inversely proportional to the square root of the size of the individual as indicated by the Eq. (4.2) and (4.3) of page 58 [10]. Likewise, σ is bounded by a threshold value to prevent it from having a negligible effect as indicated by the same page. New individuals created are assigned a σ value of 1.

3.5.5 Stagnation Test.

Stagnation test has been implemented where if the maximum value of a population does not increase, stagnation counter is incremented as a way to monitor diversity. If the counter reaches a predefined value then stagnation function is implemented in the next generation. This means that for 15 generations the maximum value has not been improved and diversity must be injected by randomly replacing 20% of the population using random uniform distribution of limits -1 and 1. .

3.6 Termination Phase

The termination condition chosen for this report is the total number of generations which is 30. According to the book [10], another termination condition is stagnation where if the average fitness of the population does not improve after a number of iterations, the algorithm could terminate as there is no further benefit of evolution. Instead, a different idea with stagnation is applied to inject diversity and let the algorithm proceed until the generation limit mentioned above. (see Sec. 3.5.5).

4 RESULTS AND DISCUSSION

As it can be observed at the line plots and the box plots, the results obtained by this paper are comparable with the baseline paper [3] as the same groups of enemies have been tested. The choice of group (2,6) was made due to the fact that it was highlighted in the baseline paper that it was one of the best pairs to use for training, while the enemy group (3,5) was selected as 3 is not beaten by any algorithm while 5 is relatively easy enemy [3]. Comparatively with the baseline paper, the results obtained are significantly worse which can be explained due to the limited budget and the lack of the parameter tuning.

4.1 Line Plots

The line plots below, see Fig. 1 and Fig. 2, illustrate the average and maximum fitness of all 10 runs for each of the 30 generations per enemy group for the two algorithms. Both algorithms show that the second group of enemies (3,5) show better results regarding the mean and maximum fitness of the population, in contrast with the first group of enemies (2,6). In addition, both enemy groups line plots for the first algorithm show a fluctuation. Although for the first group of enemies, algorithm A has a higher starting point regarding the mean fitness due to the smart initialization, eventually after 30 generations the mean of their fitness is about the same. However this is not the case for the second group of enemies as while the algorithm A has a better start, eventually Algorithm B outperforms it after 30 runs. Furthermore, the maximum fitness of the second group of enemies for the first algorithm has better results rather than the same group for the second algorithm. For the first group of enemies, both algorithm indicate a fluctuating line plot regarding the maximum fitness.

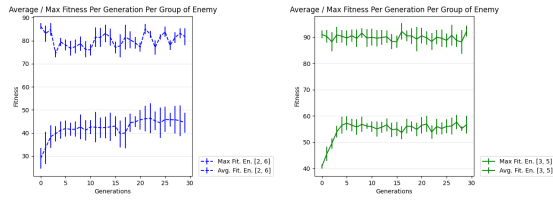


Figure 1: Line plot for average and max fitness of individuals per generation for all runs for first algorithm .

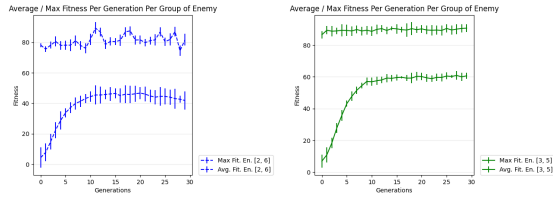


Figure 2: Line plots for average and max fitness of individuals per generation for all runs for second algorithm .

4.2 Box Plots

The box plots below, Fig. 3, is generated per enemy group per algorithm where the mean of 5 independent test runs performed by the best individual of each of the 10 runs throughout all 30 generations. [7]. Keeping the research question in mind, the results from the plot in comparison to each algorithm per enemy group, the average of the first algorithm on the first training group outperform the mean of the same group of second algorithm while for the second training group it is reversed as. It is important to keep in mind the additional overhead imposed by the local search of the

first algorithm as it required approximately 49 hours compared to 25 hours of the second, see Fig. 8 and Fig. 9.

Further conclusions are drawn using the statistical test of Wilcoxon which considers the difference between the arrays of the box plot per enemy group of the two algorithms [9]. Two-tailed test with a 5% Confidence Interval has been applied to check whether there is difference in the distributions of the arrays. All the tests conducted have indicated a p-value and median-value of more than 5% which means that the null hypothesis cannot be rejected. This implies that the average performance of the two algorithms does not provide any indications about being different. Specific details regarding the calculations of each test can be seen in Appendix A

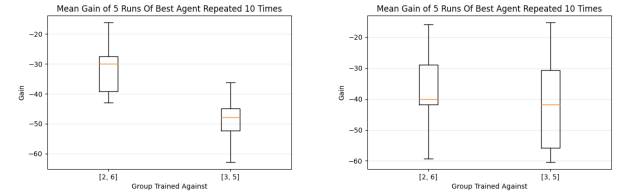


Figure 3: Box plots for average of best individual over 5 runs of best individuals per enemy group for the two algorithms.

4.3 Table

This section relates to the table of average results obtained of the best individual of the experiment which is trained by the second algorithm which is the Memetic algorithm and the second group of enemies: 3 and 5 as indicated below:

Table 1: Average life points of best individual of the experiment and each enemy for 5 runs.

	Enemies Fought Against							
	En. 1	En. 2	En. 3	En. 4	En. 5	En. 6	En. 7	En. 8
Agent Life Points	0.00	0.00	14.80	0.00	85.36	0.00	0.00	0.00
Enemy Life Points	72.00	82.00	42.00	86.00	0.00	94.00	78.00	88.00

5 CONCLUSION

This paper has as a main goal to explain the procedure of developing an EA, the differences between two manually-developed EAs and to compare them with the results of the baseline paper [3]. In general, while the first algorithm with the smart initialization seem to outperform the second one with the random initialization, the Wilcoxon test hints to no difference in the overall performance. As a team, we have divided the work that has to be done for the Task II as follows: Christos Hadjigeorgiou is responsible for the coding part, Alexandros Christodoulou and Konstantinos Christofi for the literature review and everyone contributing equally to the report.

A APPENDIX: WILCOXON TEST

This appendix provides further information about the exact computations performed on the difference of arrays per enemy for the box plot to draw statistical conclusions.

As it can be seen by Fig. 4, each array in the list corresponds to the difference of arrays between the two algorithms for the corresponding enemy group.

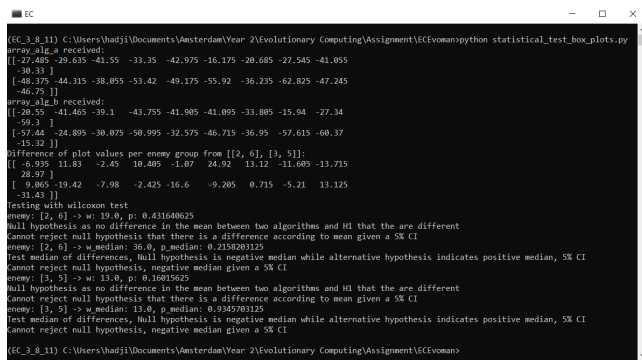


Figure 4: Test performed for each enemy considering the arrays used to create the box plot.

As it can be seen by Fig. 4, each array in the list corresponds to the difference of arrays between the two algorithms.

B APPENDIX: ENEMY GROUPS [1,2,3] AND [4,6,7] ALGORITHM B

This appendix provides information about an enemy group which contain 3 enemies rather than 2 about the second algorithm.

Average / Max Fitness Per Generation Per Group of Enemy

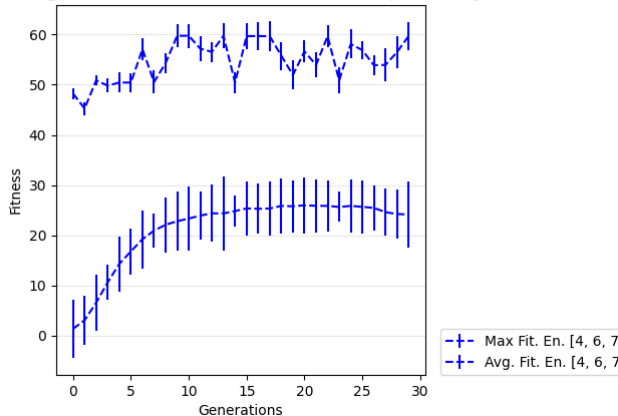


Figure 5: Enemy group 4,6,7 algorithm b line plot

As indicated by Fig. 10, it took 13 more hours for the second algorithm to finish the runs with a group of 3 enemies, so there was not much time to run the group of 3 enemies for the first algorithm due to budget constraints.

If however it is not allowed, please disregard this appendix.

Average / Max Fitness Per Generation Per Group of Enemy

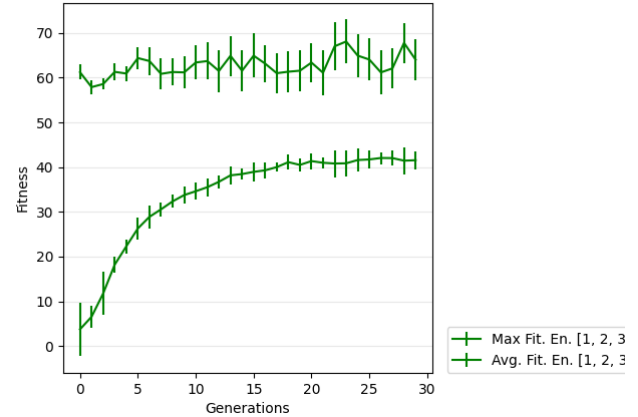


Figure 6: Enemy group 1,2,3 algorithm b line plot

Mean Gain of 5 Runs Of Best Agent Repeated 10 Times

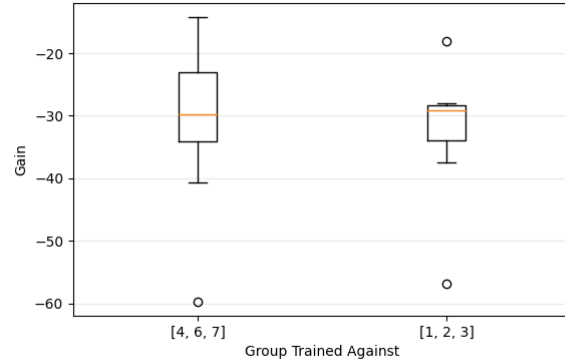


Figure 7: Both enemy groups box plots

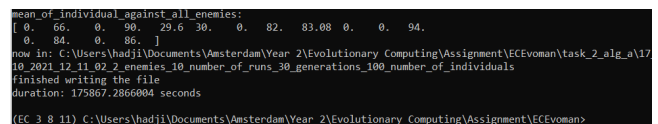


Figure 8: Time taken for for algorithm a with enemy group of 2 enemies, 175867.2866004 seconds, which is 48.85 hours

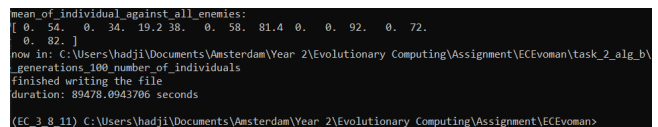
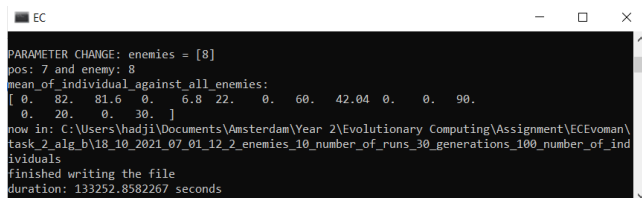


Figure 9: Time taken for for algorithm b with enemy group of 2 enemies, 89478.0943706 seconds, which is 24.85 hours



```
EC
PARAMETER CHANGE: enemies = [8]
pos: 7 and enemy: 8
mean_of_individual_against_all_enemies:
[ 0. 82. 81.6 0. 6.8 22. 0. 60. 42.04 0. 0. 90.
 0. 20. 0. 30.]
now in: C:\Users\hadji\Documents\Amsterdam\Year 2\Evolutionary Computing\Assignment\EC\Evoman\
task_2_alg_b\18_10_2021_07_01_12_2_enemies_10_number_of_runs_30_generations_100_number_of_ind
ividuals
finished writing the file
duration: 133252.8582267 seconds
```

Figure 10: Time taken for for algorithm b with enemy group of 3 enemies, 133252.8582267 seconds, which is 37.01 hours

REFERENCES

- [1] James E. Baker. 1987. Reducing Bias and Inefficiency in the Selection Algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*. L. Erlbaum Associates Inc., USA, 14–21. <https://doi.org/10.5555/42512.42515>
- [2] Fantinato D, Miras K, Eiben A, Vargas P, De França, F. 2019. EvoMan: Game-playing Competition.
- [3] Miras K, Fabricio, D. 2016. Evolving a generalized strategy for an action-platformer video game framework.
- [4] Miras K, Fantinato, F. 2016. An electronic-game framework for evaluating coevolutionary algorithms.
- [5] John H. Holland. 1992. Genetic Algorithms. *Scientific American* 267, 1 (1992), 66–72. <https://doi.org/10.1038/scientificamerican0792-66>
- [6] Natalio Krasnogor and Jim Smith. 2000. A Memetic Algorithm With Self-Adaptive Local Search: TSP as a case study. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)* (05 2000).
- [7] K. Miras. 2019. EvoMan - Framework 1.0.
- [8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
- [9] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80. <https://doi.org/10.2307/3001968>
- [10] Ágoston E. Eiben and James E. Smith. 2015. *Introduction to evolutionary computing* (2nd. ed.). Springer-Verlag, Berlin.