Investigating importance of the size of the training group in genetic algorithms in a generalist agent scenario

Evolutionary Computing, Group 41
Task 2: Generalist Agent

October 18, 2021

Brian Dmyszewicz
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
2714646@student.vu.nl

Venkat Mohit Sornapudi
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
2721697@student.vu.nl

Don Mani
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
2693434@student.vu.nl

Azhar Shaikh
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
2701842@student.vu.nl

1 INTRODUCTION

In the recent years, the field of Artificial Intelligence has seen a rapid expansion with various approaches and algorithms being utilized in many domains. Evolutionary Computing[2] is an important branch in this ever-growing discipline. Inspired by the phenomenon of evolution, it aims to replicate the real life processes such as genetic recombination and mutation on artificial populations using various optimization-focused Evolutionary Algorithms in order to find the best solution to a given problem. Evolutionary Algorithms (EAs) have shown promising advancements in beating video games[5].

A framework called EvoMan[1] can be used as a playground to test EAs. Utilizing it, we can run MegaMan II, a simple two-dimensional video game, and benchmark the EA's performance. The objective of the player in the game is to attack the enemy and defeat it by decreasing its health to 0 while avoiding enemy attacks to preserve the player's health. In the framework, the actions of the player can be controlled by an Artificial Neural Network[4], which can be trained using an EA. The goal of the EA is to produce the most optimal weights for the Neural Network, improved upon over several generations through evolutionary processes.

The two main evolutionary methods simulated by any EA are recombination (or more specifically chromosomal crossover) and mutation. Crossover occurs when 2 chromosomes (i.e. 2 strings of genes) split into multiple sections and recombine into 2 new chromosomes. Mutation, on the other hand is a stochastic process that changes the allele (variant) of the gene. In order for the EA to train the Neural Networks to defeat enemies in the EvoMan framework, different variations of those processes can be implemented. Crossover is generally perceived to have an exploratory function as it produces a greater variation, while mutation is usually exploitative and produces smaller changes.

The research question. that we investigate is whether a greater size of the training group (and coincidentally its higher overlap with the testing group) can produce better results in a generalist agent scenario. For this purpose, we selected two differently sized groups of enemies from the EvoMan framework. To reach more general conclusions, we conducted the training using two different evolutionary algorithms - one with and one without mutation. The structure of this report following the introduction consists of the methods, results and conclusions sections where we explain the used algorithm and our implementation of it, present and describe obtained results and conclude on our findings.

2 METHODS

2.1 EvoMan

The EvoMan framework[1] includes 8 static enemies to choose from, each one providing a different difficulty level. Various parameters can be changed throughout the framework such as the mode of the game which decides on player and enemy control methods. For the Individual Evolution mode, where the Neural Network-controlled player (agent) fights against a static enemy, one or more training objectives can be selected to train a specialist or generalist agent respectively. For the Neural Network to receive inputs, the agent has access to 20 sensors that measure its distance to the projectiles, the enemy and it's own orientation.

2.2 Algorithms

For the purpose of answering our research question, we use two Evolutionary Algorithms, one with mutation called EA1, and one without mutation called EA2. Below we explain the general steps involved in both algorithms.

(1) Initialization: generating the initial population consisting of individuals. Each individual is characterized by a genotype - weights of the neural network (parameters), and phenotype - the resulting neural network that outputs individual's actions. This step is only performed once, the following steps apply to every generation.
(2) Evaluation: evaluating individuals with a fitness function against a training group and assigning them a fitness score.
(3) Individual Selection: selecting only the best performing individuals to participate in the evolution.
(4) Recombination: pairing a number of individuals together and exchanging parts of their genotype.
(5) Mutation: randomly selecting different alleles for certain genes of selected individuals, this step is skipped for EA2.
(6) Repeat the evolutionary process (3-6) till last generation.
(7) Evaluate the last generation and get the best individual from all generations.

2.3 Implementation and Motivations

In order to construct the EAs, we used the tools provided by the DEAP[3] framework (Distributed Evolutionary Algorithms in Python). The exact implementation of the aforementioned algorithms including selected parameters, exact evolution operators and the motivation behind design decisions are as follows:

Population: We generate the initial population consisting of 40 individuals, each of them is assigned random initial weights (parameters) using random uniform function. Those weights are used by a Neural Network to control the player. The population size of 40 is motivated by initial testing, where selecting a higher population size did not produce better results. 40 was therefore selected as a good compromise between performance and computation time.

Evaluation: To train a generalist agent we run the EvoMan multiple mode simulation on the existing population in order to evaluate each individual against each of the two selected training groups. The simulation uses Individual Evolution mode, where the player is controller by an Artificial Neural Network (with 10 neural nodes) trained by the EA and the static enemy follows behaviour rules defined by the game. In order to reach more meaningful results for a generalist agent scenario, random enemy initialization in EvoMan has been enabled. The fitness function used to evaluate individuals against each enemy is as follows: $f = 0.9 * (100 - e) + 0.1 * p - log$, t representing the duration of the fight, p and e being the respective values of player's and enemy's remaining hit points. In multiple mode, every individual is assigned a fitness score based on this formula: $f = \bar{f} - \sigma$, where $\bar{f}$ is the mean of individual fitness values against each enemy and σ is their standard deviation.

Selection: For exploitation, we utilize DEAPs selTournament function that implements tournament selection operation, the parameters passed are: population, number of individuals to select, number of participating individuals and fitness score of individuals.

The number of individuals to select is linked to the initial number of individuals to preserve the population size, it is therefore set to 40. Number of participating individuals in tournament was set to 3 based on initial testing and it seemed appropriate for our population size. With those parameters, the function randomly picks 3 individuals and saves the one with the highest fitness score, this is repeated 40 times. We tried to use different methods for this purpose, such as DEAPs selNSGA2 function, but they produced inferior results.

Crossover: For this step we pair odd and even-numbered individuals together and use DEAPs Uniform Crossover function. The function switches weights within the pairs with a defined probability per each weight. In our implementations, that probability has been set to 30% as a result of initial testing. For the purpose of fine tuning, we also decided to only apply this function to 90% of the population. We decided to use uniform crossover as opposed to other crossover methods because of its simplicity and under the assumption that it would represent performance of other methods well enough for us to investigate the research question.

Mutation: *(for EA1).* We use DEAPs Gaussian Mutation function to randomize 30% of an individual's properties. We only apply this process to 20% of the population in order to avoid overly drastic changes in fitness values which could potentially slow down the process of evolution. The standard deviation has been set to 1 and the mean to 0 in order to avoid generating too many extreme values which we speculated could again be disadvantageous. We used the Gaussian mutation method due to it's simplicity. Furthermore, since the genotypes are real-valued floating point numbers, it made more sense to use Gaussian Mutation. For EA2 this step is omitted entirely by changing the mutation rate from 20% to 0%.

Table 1: Parameter values

Parameter	Values
Representation	Real-valued Vectors
Population Size	40
Number of Generations	50
Selection	Tournament (tournsize=3)
Crossover	Uniform (indpb=0.3)
Crossover Probability	0.9
Mutation	Gaussian ($\sigma = 1$, $\mu = 0$, indpb=0.3)
Mutation Probability	0.2 for EA1, 0 for EA2

2.4 Experiment Design

In order to test our research question, for the purpose of training we selected 2 differently sized groups of enemies provided by the EvoMan framework. The first group consists of enemies 4, 7 and 8, the second group includes enemies 2, 4, 5, 6, 7 and 8. We used both of the described EAs - with and without mutation and ran each 10 times against both enemy groups while using multiple mode. We obtained the means of each generation's maximum fitness scores, average fitness scores and standard deviations over the 10 completed runs for both EAs and both enemy groups. The fitness scores were assigned as described in the Implementation section. The purpose of this training was to produce 10 individuals from each EA and each training group, best suited to face all 8 provided enemies. In order to test their performance in this generalist scenario, the 10 individuals played the game 5 times against each of the 8 enemies. Their 5 run average individual gain (obtained by subtracting the remaining enemy life from remaining player life) against each enemy was then saved in order to find the very best individual across both EAs and both training groups and portray his result against each enemy. Their average individual gain over all enemies was also saved in order to compare overall performance of both groups. For the purpose of testing, single mode was used instead of the multiple mode and iterated over all enemies, this was a preferable solution as it allowed us to save all necessary data.

3 RESULTS AND DISCUSSION

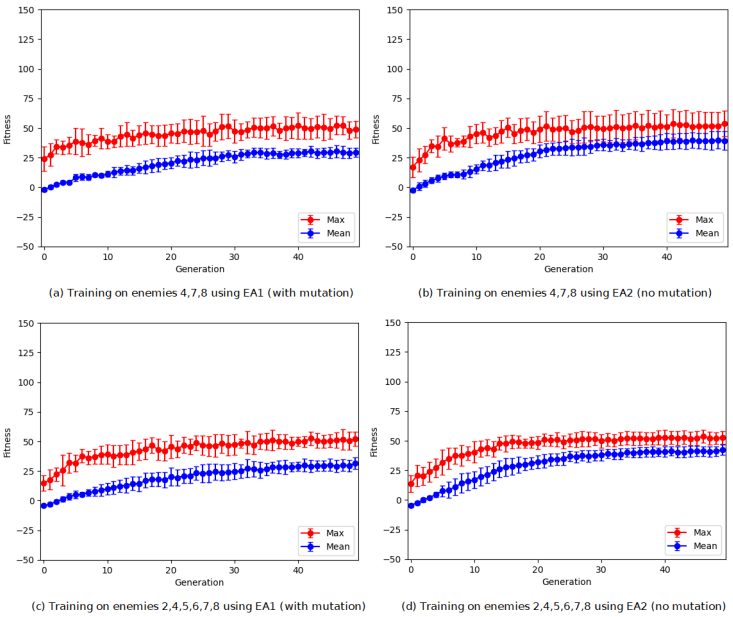

Figure 1: Line plots

Each point on the line plots (Figure 1) portrays the 10 run average of maximum and mean fitness values of a population against all enemies in a training group for each generation, as well as their standard deviation for every generation. The line plots (a) and (b) portray results for the first training group, the line plots (c) and (d) show the results for the larger group. In all four line plots, we observe that, for EA2 (no mutation), the mean fitness of the population gets closer to the maximum fitness quickly and eventually plateaus. In contrast, in case of EA1, due to the randomness associated with mutation, some individuals remain different from rest of the population across all generations and therefore, the mean fitness remains lower. Both evolutionary algorithms seem to perform very similarly in their search for an individual with the highest fitness score. We can also observe that the maximum fitness does not differ greatly between the two training groups.

Individual gain was obtained by subtracting the remaining enemy life from remaining player life. The best individual from each

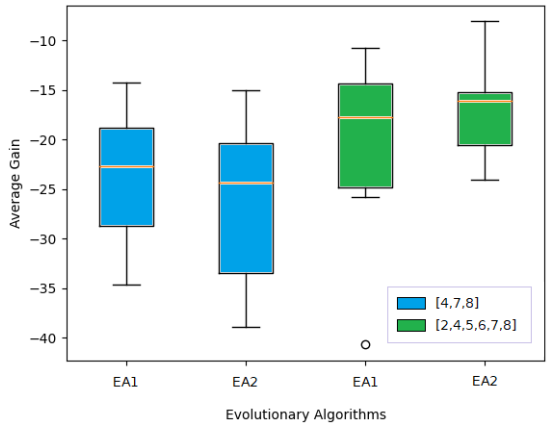

Figure 2: Box plots

of the 10 runs, fights 5 times against all 8 enemies to get the average individual gain. The distribution of these gains across 10 runs is portrayed in the box plots (Figure 2). The line in the middle of each box plot represents the mean, the whiskers represent standard deviation while the body represents the values that fall between 25 and 75% of the distribution. Upon observation of the box plots, we find that, on average, both EAs trained on both groups did not perform very well while tested against all 8 enemies. From additional analysis we found that agents from both groups are able to defeat enemies 2, 5 and 8, while failing to defeat the rest. This results in a negative average gain, as portrayed by the box plots. Their seemingly similar performance can be attributed to the difficulty of enemies, through additional testing we found that enemies 7 and 8 prove to be crucial for training. Enemies 2 and 5 on the other hand are the easiest to defeat and seem the least relevant for the purpose of training a generalist agent. Notwithstanding, despite both groups only managing to win over 3 enemies, a difference in their gain can easily be observed. The bigger training group clearly results in a higher gain and in case of EA1, lower standard deviation. This could be an indication of the agent being more generalist but could also result from a greater overlap between the training and testing groups. While we can observe an impact of mutation in case of both training groups, this effect is inconsistent. Mutations seems to benefit individuals trained on a smaller group but the effect is opposite for those from a bigger group. Possibly mutation allows a smaller training group to develop better generalist agents, they are still inferior, however, to agents produced by the greater group.

Upon comparing the two EA's on groups [4,7,8] and [2,4,5,6,7,8], using Welch's t-test, we obtained the p-values of 0.500 and 0.3655 respectively. The high p-values can be contributed to the randomness involved in the experiments.

Table 2 presents the final player and enemy life of the very best individual found across both EAs and both training groups. The individual was trained on the bigger group including enemies 2, 4, 5, 6, 7 and 8, using EA2 - the algorithm without mutation. We can see that it managed to defeat 3 enemies: 2, 5 and 8, was quite close to defeating enemy 7 and 3 and did reasonably well against enemy 4 and 6. The average gain of our best individual was -8.085.

Table 2: Final player and enemy life of Best individual. Individual trained on group [2,4,5,6,7,8] with EA2

Enemy id	1	2	3	4	5	6	7	8
Player Life	0.0	46.0	6.0	0.0	49.12	0.0	0.0	26.2
Enemy Life	80.0	0.0	24.0	32.0	0.0	36.0	20.0	0.0

Comparison to Miras (2016): From table 2, we observe that our best individual defeats the same enemies (2, 5 and 8) as the best Genetic Algorithm (GA10) of [1] (in the generalization test) does. While Miras[1] ran across 100 generations for population size of 100, we ran across 50 generations for population size of 40. We chose these particular parameters as, after running multiple experiments, we did not see much fitness or gain improvement by increasing them.

4 CONCLUSIONS

Based on the experiment's results we see that the effects of mutation on the fitness value of the best individual across all generations are not very significant and produce contradicting results - positive for a smaller group, negative for the bigger one. We also see that implementing mutation reduces the mean fitness values of a population. This is not relevant in search of the best possible individual and can be corrected using individual selection. By training two Evolutionary Algorithms using two differently sized groups containing enemies of varying difficulty, we can conclude that enemies 2 and 5 provide little challenge while enemies 7 and 8 seem important for the purpose of training. We also conclude that the increased size of a training group, and the resulting overlap between the training group and the testing group proved quite significant. Including an enemy in the training group makes the agent perform better against the same enemy during testing, while in case of more versatile enemies such as 7 and 8, it also leads to improvement in performance against other enemies. We can also say that a bigger group produces more consistent results, as indicated by lower standard deviation. Given all our findings, we are able to answer the research question by concluding that a greater size of the training group (and coincidentally its higher overlap with the testing group) can produce better results in a generalist agent scenario. Whether this is due to the bigger group producing a better, more generalist agent or whether this is simply a result of greater overlap between the training and testing groups was the main limitation of our study. We speculate that the overlap itself might play a bigger role in the improved results but this remains to be investigated in future research. Our final conclusions hold true for all the previously described methods and parameters. To further generalize them, we would need to perform tests on a broader spectrum of problems and models.

5 CONTRIBUTIONS

Azhar: experiment code development, refactoring of code for experiments, data analysis, editorial work. **Brian:** initial algorithm design, data analysis, documentation, report creation, editorial work. **Don:** experiment code for data visualization, running experiments, data analysis, editorial work. **Mohit:** experiment code development, plotting of data, data analysis, editorial work.

REFERENCES

[1] Karine da Silva Miras de Araújo and F. O. D. França. 2016. Evolving a generalized strategy for an action-platformer video game framework. In *CEC*.

[2] Agoston E Eiben, James E Smith, et al. 2003. *Introduction to evolutionary computing*. Vol. 53. Springer.

[3] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research* 13 (jul 2012), 2171–2175.

[4] Frank Rosenblatt. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* 65, 6 (1958), 386.

[5] Julian Togelius, Sergey Karakovskiy, Jan Koutník, and Jurgen Schmidhuber. 2009. Super mario evolution. In *2009 ieee symposium on computational intelligence and games*. IEEE, 156–161.