Does using the ESC Island Model to train on different combinations of enemies contribute to improved generalization?

20-10-2023

María Ármann
Vrije Universiteit Amsterdam
2776514
m.armann@student.vu.nl

Louisa Crijnen
Vrije Universiteit Amsterdam
2653938
l.a.p.crijnen@student.vu.nl

Peter Derksen
Vrije Universiteit Amsterdam
2714039
p.a.derksen@student.vu.nl

Sara Lute
Vrije Universiteit Amsterdam
2785832
s.lute3@student.vu.nl

Sanne de Wilde
Vrije Universiteit Amsterdam
2650188
s.j.de.wilde@student.vu.nl

ACM Reference Format:
María Ármann, Louisa Crijnen, Peter Derksen, Sara Lute, and Sanne de Wilde. 2023. Does using the ESC Island Model to train on different combinations of enemies contribute to improved generalization?: 20-10-2023. In *Proceedings of (Evolutionary Computing).* , 5 pages. https://doi.org/none

Evolutionary Computing, Oct, 2023

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/none

ABSTRACT

This paper proposes a method for training a Genetic Algorithm on various combinations of enemy sets in the EvoMan video game framework to enhance algorithm generalization. Two distinct approaches, including a novel ESC Island Model, to find out whether training on different combinations of enemies contributes to improved generalization. The results show that the ESC Island Model in our genetic algorithm is an effective strategy for optimizing the training to play against a diverse set of enemies.

1 INTRODUCTION

Evolutionary computing, inspired by natural selection and genetics, tackles complex optimization and search problems. Genetic Algorithms (GAs), a subset of this field, stand as prominent examples of these bio-inspired algorithms. Finding a good trade-off between exploration (diversity) and exploitation (fitness) in a GA's play a significant role in leading the algorithm to convergence and influencing its capacity to explore solution spaces effectively. For the interior of the algorithms, this will lead as the main goal.

Like many optimization algorithms, assessing performance is essential. Video games provide a valuable testing ground for evaluating and benchmarking various methods due to their controlled and interactive nature. In our study, we evaluate and compare two algorithms using the video game platform EvoMan [6].

During the algorithm training process, our aim is to find a specific strategy that can defeat all enemies. Within the EvoMan framework, all individuals within each generation compete against all enemies in a predefined enemy set. However, by using the complete enemy set, the diversity among these enemies can pose challenges for the algorithms to generalize effectively. Thus, a more promising approach may involve training on a subset of enemies. This strategy introduces an element of uncertainty, as the algorithms will eventually face unseen enemies. Due to this, we aim to explore a hybrid approach that encompasses both methods. This approach entails having multiple groups of individuals, each trained on distinct enemy sets. During the training phase, we transfer some individuals between these sets with the hope that they can retain the knowledge gained from their previous encounters while learning to conquer the new enemy set.

This brings us to the central focus of this paper, where we seek to investigate whether training on different combinations of enemy sets, inspired by the island method, can yield superior generalization when compared to training on a thoughtfully selected subset of enemies.

1.1 Enemy Sets

The EvoMan video game framework will serve as the testing platform for performance evaluation. The framework contains 8 different enemies. The purpose of the game is that the player learns how to beat all of them by creating a strategy that uses the following actions: move left, move right, jump, release jump and shoot.

This paper aims to optimize algorithm generalization by selecting a specific group of training enemies. The goal is to maximize the number of adversaries the algorithm can outperform. Two sets of enemies will be compared. The first set, [1, 2, 3, 7, 8], was chosen through trial and error, as it showed superior generalization. The second set, [2, 3, 5, 6, 7, 8], initially included all eight enemies, but due to the negative impact of enemies 1 and 4, they were excluded. These adjustments make these two sets the most promising options.

2 METHODS

2.1 Set-up and Initialization

Both evolutionary algorithm controllers undergo training through a one-layer feedforward neural network with 265 hidden neurons and sigmoid activation functions. The inputs consist of the weights and biases of these neurons, while the outputs correspond to the player's actions. Initiating the process with random solutions has been observed to have a higher likelihood of achieving optimality compared to initializing with a heuristic [1]. Therefore, the initial weights are randomly generated from a Uniform distribution within the normalized range of [-1, 1].

2.2 Fitness function

The fitness function for individual i is as follows:

$$f_i = (\sum_{j=1}^{8}(0.9(100 - e_l^j) + 0.1p_l^i - log(t))) * 1.5^k$$

Where e_l^j represents the remaining life of enemy j and p_l^i represents the remaining life of player i. The variable k is the total number of enemies defeated, where the fitness value increases significantly as more enemies are eliminated. Here, t is the elapsed time, creating a penalty for a game that takes too long to finish. In this fitness function, the emphasis is on beating as many enemies as possible, which is the main goal of the game.

2.3 Algorithms

2.3.1 Parent Selection. The parent selection operator is one of the most important in guiding the GA to converge to the optimal solution [4]. In our previous research paper, we investigated the impact of introducing diversity into the parent selection process. This was achieved by modifying the linear ranking method, such that it chooses a set of parents where the first parent is chosen based on the selection probability, $P_{lin-rank}(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$, where s falls within the range $1 < s \leq 2$. The second parent is chosen based on the same formula with inverse ranking. Meaning that the parent with the lowest fitness value ranks the highest and gets the highest selection probability. As this method could lead to less chance of reaching an optimal solution, elitism is included as well, with 10% of the fittest individuals having a guaranteed place in the next generation. This combination of modified parent selection and elitism proved to be significantly more effective than the original linear ranking method. It exhibited faster convergence for all types of enemies. Therefore, we decided to adopt this parent selection mechanism for both approaches.

2.3.2 Fitness Sharing. Fitness sharing, which aims to preserve diversity by lowering the fitness of similar individuals, encourages exploration of various solution areas, potentially leading to better solutions and escaping local optima [5]. In this study, we integrated fitness sharing into parental selection, measuring distances with Euclidean metrics. Surprisingly, including fitness sharing yielded

inferior results. This outcome may be attributed to the natural diversity present in the initialization phase, which persisted throughout learning. This raised concerns about determining an optimal sharing radius, risking over-penalization with a small radius or insufficient penalization with a larger one. Consequently, this aspect was eventually removed from the algorithms.

2.3.3 Offspring Creation. To generate offspring, several methods are chosen with a certain probability. The parameter p_m represents the likelihood of a mutation occurring, while a probability of $1 - p_m$ results in new offspring through recombination. In cases of recombination, the offspring is created with equal probability of n-point crossover and arithmetic crossover. This approach strikes a balance between exploitation and exploration, enabling the model to perform both.

Mutation is an exploitative tactic where it makes small changes in the DNA, thus staying close to the parent. We included the probability of a mutation occurring as a parameter that could be tuned, thus enabling us to investigate whether it is most effective to have crossover or mutation. We initially included it in parameter control, however, we noticed that this did not give promising results and the value of the parameter was random and did not evolve into either favoring mutation or crossover. Thus we chose to tune this parameter instead.

Our N-point crossover takes two parents' genetic sequences, splits them into 53 segments (so, segments of 5 genes), and, for each segment, randomly chooses genes from one of the parents based on a probability threshold (0.5 in this case). This process repeats for all segments, creating an offspring with a mix of genes from the two parents.

For arithmetic recombination, the weights of the offspring are created as follows: $z_i = \alpha x_i + (1 - \alpha) y_i$, in which z_i is the offspring, α is defined in parameter control, which will be discussed in section 2.5.2, and x_i is one parent, and y_i is the other parent.

2.3.4 Survivor Selection. For the purpose of selecting survivors, we opted for the fitness-based Round-Robin (RR) tournament method. This selection is motivated by its potential to incorporate less-fit solutions if they have a lucky draw of opponents [3]. The Round-Robin tournament combines the current population with the newly generated offspring to create a new total population. Each individual is scored in a tournament against a random selection of q other individuals. Within the tournaments, the individual is compared to each of the q others and is given 1 point if the score is better, otherwise 0. The individuals are then ranked based on their victory counts, and the highest-scoring ones are selected as survivors. The elite parents identified in the parents-choosing algorithm are kept safe during the survivor selection, creating 10% of the new population. The Round-Robin tournament is then run on the rest, creating the remaining 90% of the new population.

2.4 ESC Island Model

With over 200 potential combinations of enemies to train the model on, deciding which combination is essential. The ESC island model suggested in this paper divides the population into isolated subpopulations (islands), with each island focusing on a specific combination of enemies.

Combinations are made for all enemy sets of size 3 to 8 and for each set size it takes a maximum of 50 combinations to limit computational time. The model runs two times five generations for each enemy set, each enemy set has a population of size 20 allowing for independent evolution. After one epoch, ten generations, the island model destructs 20% of the islands with the worst fitness scores (based on the fitness function described in Section 2.2).

Once there are less than 50 islands left, the model starts to periodically allow migration and share genetic information between islands to encourage diversity and adaptation. Implemented as the "island-hopping" function, 5% of the individuals of each island's population are randomly moved to another island's population for each consequent epoch. This random selection mitigates biases that could be introduced by deterministic or fitness-based migration strategies and ensures that a wide variety of individuals get the opportunity to move between islands. It is limited to 5% to warrant a balance between exploration and exploitation for a smaller population size.

Thus, this implementation adds diversity on the island level by swapping individuals, not on the population level by swapping weights.

2.5 Hyperparameters

To ensure a lower budget, the traditional hyperparameter tuning was combined with parameter control.

2.5.1 Tuning. Tuning was done on the population size, the number of parent sets, the variable q, the number of generations and the mutation probability. This resulted in the following optimal parameters:

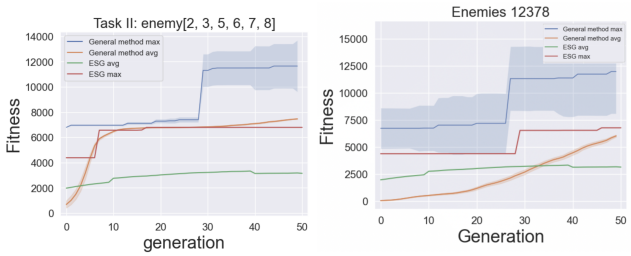
population size: 100, parent sets: 40, q: 10, generations: 50, mutation probability: 0.9.

The most interesting parameter to discuss in this situation is the mutation probability. This value is quite high showing that there is a strong favor towards mutation. This can possibly be explained by the fact that in the case of mutation, the offspring are very similar to the parents. We chose the best-performing parents, thus with a small mutation the offspring will stay close to the well-performing parents.

2.5.2 Parameter Control. Instead of tuning parameters beforehand, we decided to tune some parameters during the run of our evolutionary algorithm. The rationale behind our decision for parameter control is that different values of parameters might be optimal at different stages of the evolutionary process [3]. Our approach is inspired by self-adaptive mutation as described by Eiben and Smith (2015). This means that for the chosen parameters, which are α in arithmetic recombination, and s in parent selection, they are included in the genome of an individual. This means that to the weights of an individual, α and s are added as such: $< w_1, w_2, ..., w_{265}, \alpha, s >$. In every step of the genetic algorithm, the parameters of the individuals evolve similarly to the evolution of the weights of the individuals. The only difference is that we make sure that they do not exceed their predefined interval. α is initialized for every individual uniformly random between [0, 1]. During the run, α seems to oscillate between 0.4 and 0.8. The parameter s is initialized for every individual uniformly between (1,

2]. During the run, s seems to start quite high and decrease until half of the number of generations, and then increase again until the end.

3 RESULTS AND DISCUSSION



(a) Fitness for ECS Island Model and set [2,3,5,6,7,8] **(b) Fitness for ECS Island Model and set [1,2,3,7,8]**

The best individual comes from the ESC Island Model.

Enemy	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Enemy Life	0	18	0	0	59.8	3.4	21.4	5.8
Player Life	80	0	0	20	0	0	0	0

Table 1: The player- and enemy health of the best individual against each enemy

	Enemy 1	Enemy 2	Enemy 3	Enemy 4
P-value	0.0123	0.0158	0.0098	0.0234

Table 2: P-value of the Wilcoxon signed-rank test for the gain between the methods per enemy

	Enemy 5	Enemy 6	Enemy 7	Enemy 8
P-value	0.0020	0.0345	0.0076	0.0412

Table 3: P-value of the Wilcoxon signed-rank test for the gain between the methods per enemy

3.0.1 General Method. Interestingly enough, you can see in the graph for the general method on enemies 2,3,5,6,7,8, that the average fitness of this method converges quite quickly to the optimal point, while during tuning the optimal value for the number of generations is 50. Furthermore, you see that the confidence interval for the average is very small for this method. This could possibly be due to the fact that fewer crossovers are used, thus the algorithm mainly takes small steps to improve the individuals. Thus resulting in quite similar results against the enemies. However, you see an interesting jump in the max fitness after 30 generations, and the CI is much larger in this case. This could possibly be due to the fact that some individuals escape a local optimum, however, this does not impact the average, thus it is only for a small number of individuals. This is similar for the second set of enemies as seen in figure (b).

3.0.2 ESC Island Model. The island model performs quite well in beating the number of enemies, however, in the graphs is performs worse. This is interesting.

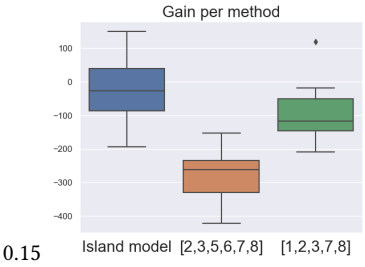


Figure 2: Boxplot of the gain per method

3.0.3 Comparison. Interestingly enough when you look at plot (a), you can see that the general model seems to be outperforming the island model. However, when you look at the number of enemies that the island model beats, it performs much better. The general model, when training on the the enemy set of 2,3,5,6,7,8 only beats 2 enemies, while the island model beats 5. When you look at the other set, it is similar. However, in practice the island model performs much better.

3.0.4 Baseline Model. When we compare these two algorithms to the ones presented in the baseline paper [2], it becomes evident that our General Method exhibits slightly inferior performance compared to the baseline. In contrast, the ECS Island Model significantly surpasses the baseline in terms of performance. This illustrates that having effective algorithms alone is insufficient; the training process of these algorithms plays a crucial role in achieving generalization.

3.0.5 Budget. In terms of budget, the algorithms run efficiently. The sections that took the most time were the tuning of the hyperparameters, due to the large number of parameters, and the training of the algorithm using the ECS Island Model, due to its many islands of combinations.

4 CONCLUSIONS

In conclusion, the ESC Island Model in our genetic algorithm has proven to be an effective strategy for optimizing the training to play against a diverse set of enemies. By combining isolated evolution with periodic random migration, we strike a balance between exploration and exploitation. This approach promotes diversity, reduces the risk of premature convergence, and ultimately enhances the algorithm's ability to defeat more enemies. Overall, the island model has demonstrated its value in improving the effectiveness of our evolutionary training process.

REFERENCES

[1] Chathurangi Shyalika 2010. Population Initialization in Genetic Algorithms. (2010). Retrieved Oct 1, 2023 from https://medium.datadriveninvestor.com/population-initialization-in-genetic-algorithms-ddb037da6773

[2] Karine da Silva Miras de Araujo and Olivetti de Franc¸a Fabrício. 2016. *Evolving a Generalized Strategy for an Action-Platformer Video Game Framework.* Ph.D. Dissertation. Federal University of ABC, Santo Andre, Brazil.

[3] J. Eiben, A. E. Smith. 2015. *Introduction to Evolutionary Computing.* Springer. https://doi.org/10.1007/978-3-662-44874-8

[4] Volker Nannen, S.K. Smit, and A.E. Eiben. 2008. Costs and Benefits of Tuning Parameters of Evolutionary Algorithms. *Parallel Problem Solving from Nature – PPSN X* 287 (2008), 528–538. https://doi.org/doi.org/10.1007/978-3-540-87700-4_53

[5] Pietro S. Oliveto, Dirk Sudholt, and Christine Zarges. 2019. On the benefits and risks of using fitness sharing for multimodal optimisation. *Theoretical Computer Science* 773 (2019), 53–70. https://doi.org/doi.org/10.1016/j.tcs.2018.07.007

[6] Jacob Schrum and Risto Miikkulainen. 2014. Evolving multimodal behavior with modular neural networks in ms. pac-man. *Proceedings of the 2014 conference on Genetic and evolutionary computation* (2014), 325– 332. https://doi.org/10.1145/2576768.2598234