Assessing the Performance of Two Evolutionary Strategies on Various Enemy Groups

Task 2: Generalist Agent

Group 44

Caroline Hallmann
Student number: 2640914

Ockert Malan
Student number: 2787978

Oromia Sero
Student number: 2720857

Rishikesh Narayanan
Student number: 2773673

Vidhya Narayanasamy
Student number: 2733527

1 INTRODUCTION

One common problem domain found in the field of artificial intelligence (AI) is that of autonomous navigation and problem-solving, either in the real world for applications such as self-driving cars, or a virtual one with autonomous agents serving as opponents in video games [4]). One method for the development of these autonomous agents is evolutionary computing (EC), where the cross-over, mutation and selection processes that drive biological evolution are imitated to "evolve" AI agents that are capable of solving specialist or generalist tasks [6, 4].

The implementation of the EC approach, known as an evolutionary algorithm (EA), must be carefully designed for the specific problem implementation, not only in the critical components of it (namely representation, mutation, cross-over, fitness evaluation and selection), but also in the selection of problem-cases to use to drive the evolutionary process [6]. The aim of this paper will be to investigate the effect that this latter component can have on the robustness and overall performance of on an EA. This will be done by using the EvoMan video-game framework and comparing the performance of two generalist EA implementations using two different selections of enemy groups as training sets.

2 METHODS

2.1 EvoMan framework

The EvoMan framework was created to aid in the evolution of game-playing agents. It is a video game in which the player must fight an enemy who shoots, jumps, and moves. The framework includes eight different enemies, each with unique behaviour, and a way of winning that can be controlled by manual input or an autonomous agent. There are 20 sensors available that provide information about the environment. Furthermore, both the player and the enemy begin with 100 points of energy.

2.2 Algorithm Description

The EA used consisted of a randomly initialized population, an evaluation function, a parent-selection mechanism, variation operators, and finally a survivor selection mechanism. The evaluation function was set to the default fitness function from the Evoman Framework[5].

Two different approaches with different crossover, mutation, and parent selection methods were chosen to create two different genetic algorithm variations that could potentially learn to create a good generalist player. These distinct methods for the two strategies were chosen based on empirical studies that compared various techniques for optimization problems[3, 10]. The same survivor selection method was used for both strategies. Moreover, an array of Floating-Point(Real-Valued) representations was used for the implementation of the strategies.

2.3 Strategy 1: EA1

2.3.1 ***Blend crossover (BLX)***. BLX is a method for producing offspring in a region larger than the one originally spanned by the parents. The additional space varies by coordinate and is proportional to the distance between the parents[6]. Given two parents x1 and x2, where in terms of position $x1 < x2$, the blend crossover randomly selects a child in the range $[x1-a(x2-x1), x2+a(x2-x1)]$. It is suggested that a good choice of alpha(α) is 0.5, with each chosen value being equally likely to lie either inside or outside of the original rectangle spanned by the parents, creating a balance between exploration and exploitation.

2.3.2 ***Uniform Mutation***. The values of new gene values for uniform mutation are drawn at random (uniformly) from a given [Lower bound, Upper bound]. It's similar to bit-flipping (binary encodings) and random resetting (integer encodings)[6].

2.3.3 ***Stochastic Universal Sampling(SUS) Parent Selection***. Stochastic Universal Sampling (SUS) is a single-phase sampling algorithm with minimal spread and zero bias[1]. SUS employs N equally spaced pointers instead of a single selection pointer used in roulette wheel methods, where N is the number of selections required. A single random number pointer1 in the range [0, 1/N] is generated after the population is randomly shuffled. The N individuals are then chosen by generating the N pointers, starting with pointer 1 and spacing them by 1/N, and selecting the individuals whose fitness spans the positions of the pointers. Furthermore, because individuals are chosen solely based on their position in the population, SUS has no bias.

2.4 Strategy 2: EA2

2.4.1 ***2-point crossover***. 2-point crossover involves the increase of numbers of cut-points. This results in the exchange of more gene segments which then creates more distinct individuals. This method results in a greater diverse population which aids in searching the higher fitness value within the search space.

2.4.2 ***Gaussian Mutation***. Gaussian mutation includes the addition of a random Gaussian distribution value to a selected gene.

It is non-uniform and involves mutating genes by adding a small amount to their value [6]. The value generated through the random Gaussian distribution then causes the development of the new offspring. This form of mutation was chosen as the ability to specify σ allows for customization concerning the extent to which genes are mutated [2].

2.4.3 ***Tournament Parent Selection***. For tournament selection, individual pairs are sequentially chosen from a mating pool. This method compares the generated fitness values of grouped individuals and selects the highest fitness values. By ranking individuals' fitness per group rather than the entire population at once, it is more time-efficient nature in comparison to other selection mechanisms.

2.5 Survivor Selection Method

Both EAs used the μ,λ survivor selection method. The parent population consists of μ individuals and generates λ number of offspring. Trough the combination mechanism, the best μ individuals with highest fitness values are kept in the pool and form the next generation. Thus, parents from the previous generation are disregarded and selection mechanism continuous with a new population of individuals. By taking into account both age and fitness, it promises higher degree of selection pressure compared to other replacement mechanisms [6].

3 EXPERIMENTAL SETUP

3.1 DEAP framework

The DEAP framework was used to implement the different functions of EA1 and EA2. The experimental set-up included the application of DEAP on python to run the EvoMan game framework. DEAP is a well-designed EC library that provides useful tools for a rapid development of unique EAs [7].The advantage of using DEAP is the explicit representation of EAs and its transparent data structure.

3.2 Parameter Setting

Given the time and computational power available, numeric parameters were adjusted based on examinations of various values previously studied for the parameters of the methods described in table 1. Based on the outcomes of these studies, parameter values that would best suit the problem at hand were chosen. In accordance with Eiben et al., parameters for the Gaussian mutation method were selected, and Alpha (α - for Blend Crossover) was set to 0.5. For crossover probability, mutation probability and population size selection these studies [8, 9] were referenced.

Additionally, two groups of enemies were created for the purposes of this project. Group one included enemies 3, 6, and 7, while Group two included enemies 2, 5, and 8. These enemy groups were designed to have a wide range of behaviours, abilities, and attack strategies that demand a variety of strategies from the player. This is done in an attempt to test the generalizability of the two EA approaches that have been implemented. All three enemies in each group move differently, and the projectiles they fire during an attack differ in method and amount.

Table 1: Numerical Parameters

Parameter	EA1	EA2
Population Size	100	100
Crossover probability	80	80
Alpha (for Blend Crossover)	0.5	-
Mutation probability	30	30
Standard deviations (for Gaussian mutation)	-	1
Mean (for Gaussian mutation)	-	0

4 RESULTS

Two experiments were conducted using the two different EA algorithms as mentioned in the previous section. The EA algorithms were experimented on two different independent groups where group 1 consisted of enemies 2,5,8 and group 2 consisted of enemies 3,6,7. The experiment was conducted by running the EA algorithms 10 times on the two enemy groups individually. The evaluation results of the EAs are represented using line plots and Box plots where the line plots represent the average of mean and maximum fitness of each generation and their standard deviations from each training runs and the box plots represent the individual gain of the best individual from each testing runs. The line plot (Figure 1) below shows the comparison of the fitness of EA1 and EA2 for group 1 (enemies 2,5,8) and group 2 (enemies 3,6,7) respectively. For group 1 (Figure 1, left plot), the average fitness of EA1 indicates a sharp increase from generation 0 to generation 1, following which the fitness level stabilizes. In contrast, the average fitness of EA2 steadily rises across generations. The best fitness value for EA1 began with a relatively high value, increased until generation 2, and then stayed the same until generation 4. Thereafter, the best fitness value shows a moderate increase and stabilizes after generation 6. On the other hand, EA2 exhibits a sharp increase in fitness value until generation 3. For generation four, the fitness value is at its maximum and steadily rises following a fall.

The average fitness of EA1 for group 2 (Figure 1, right plot) shows a significant increase from generation 0 to generation 1, after which stabilization of the fitness level is noticed. While EA2's average fitness increases consistently over generations, in contrast to group 1, its mean values are still substantially lower. The best fitness values for the both algorithms started at the same point. But throughout the generations EA1 outperformed EA2.

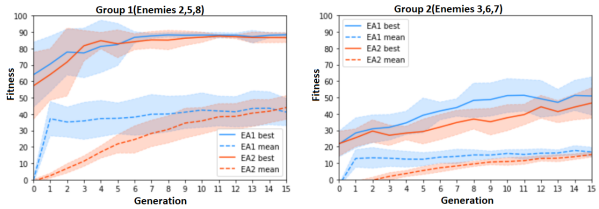

Figure 1: Comparison of fitness of EA1 and EA2 with enemies group 1 (Enemies 2,5,8) and group 2 (Enemies 3,6,7)

The box plot (Figure 2, left plot) shows that the performance of EA1 and EA2 on group 1 is very similar. EA1 achieved a slightly higher maximum gain than EA2, but EA2 had a higher median gain. In case of group 2 (Figure 2, right plot) EA1 achieved higher maximum and mean gain than EA2 with much more narrow distribution but many outliers were detected.

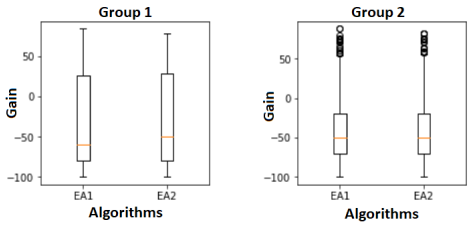

Figure 2: Comparison of gain of EA1 and EA2 with enemies group 1 (Enemies 2,5,8) and group 2 (Enemies 3,6,7)

The box plot (Figure 3) demonstrates that EA2 outperformed EA1 in terms of consistency of outcomes. Despite having poorer overall performance, both EAs trained on group 2 (enemies 3,6, 7) were far more effective at defeating "unseen" enemies than the EAs trained on group 1 (enemies 2, 5, 8). With the support of the plots, it is fair to say that EA2 outperforms EA1 for group2 compared to that of group1. A t-test was performed to support our assertions statistically, and the P-values of 0.563 for group 1 and 0.02375 for group 2 respectively which is in favor to our findings.

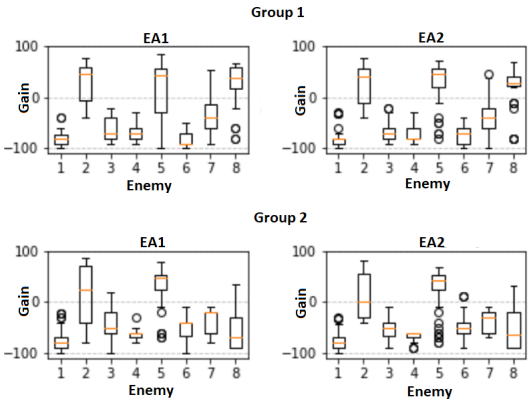

Figure 3: Comparison of gain of EA1 and EA2 with enemies group 1 (Enemies 2,5,8) and group 2 (Enemies 3,6,7)

5 DISCUSSION

According to the experimental findings, both the algorithms perform similarly in the case of group 1 but in the case of group 2 EA2 performs significantly better than EA2. The results obtained show that training set selection may be more important for algorithm robustness than algorithm design. It is also fair to say that EA2 shows slightly more consistent performance than EA1, and supports that it may be more robust. This is due to the different crossover methods used in the algorithms.

The selection methods used in EA1 and EA2 are Stochastic universal sampling (SUS) and tournament selection respectively. SUS as explained in the methods section is unbiased and uses uniform selection hence it might be possible the weakest candidate is selected a lot of times which may result in inconsistency in mean fitness while in tournament selection the individual with higher fitness is selected which may lead to better consistency and better results (fitness) after each generation. In addition to SUS, the blend crossover strategy makes the search area wider which is not in the case of 2 point crossover. The current setup of EA1 will help solve design problems while EA2 will be better at solving repetitive problems.

It is possible that healthy individuals, or those with high fitness, may be eliminated during recombination and lost to the genetic pool as a result of the natural selection process. Therefore, it could be interesting to investigate in other studies how the existence of survival selection and a different crossover technique affects the comparison of EAs utilizing self-adaptive mutation.

6 CONCLUSION

This study compared the fitness and performance gains of two EA strategies using distinct recombination and parent selection procedures. All things considered, we may say that individual gain and fitness were impacted by the chosen selection methods and the crossover operators. The selection of enemy groups also had a significant impact on the EA's performances, with the selection of widespread enemies like enemies 2,5,6 the EAs were able to defeat enemies on which they were trained on. When the EAs were trained with enemies 3,6,7 they were able to defeat a lot of unseen enemies. This show that the selection of enemies plays a key role in the EA's performance. Additionally, EA1 was fast in finding a solution however was not consistent. On the contrary, EA2 was slow in finding the solution but was consistent. Having said that, EA1 will be appropriate for solving design problems and EA2 for solving repetitive problems due to its consistency.

REFERENCES

[1] James E Baker et al. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the second international conference on genetic algorithms*, volume 206, pages 14–21, 1987.
[2] Okezue Bell. Applications of gaussian mutation for self adaptation in evolutionary genetic algorithms. *CoRR*, abs/2201.00285, 2022.
[3] Adriana Cervantes-Castillo, Efrén Mezura-Montes, and Carlos A Coello Coello. An empirical comparison of two crossover operators in real-coded genetic algorithms for constrained numerical optimization problems. In *2014 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*, pages 1–5. IEEE, 2014.
[4] Karine Da Silva Miras De Araujo and Fabrício Olivetti De Franca. Evolving a generalized strategy for an action-platformer video game framework. *2016 IEEE Congress on Evolutionary Computation, CEC 2016*, pages 1303–1310, nov 2016.
[5] Karine da Silva Miras de Araújo and Fabrício Olivetti de França. Evolving a generalized strategy for an action-platformer video game framework. In *CEC*, 2016.
[6] A E Eiben and J E Smith. *Introduction to Evolutionary Computing*. Springer, Heidelberg, 2nd edition, 2015.
[7] Félix-Antoine Fortin, François-Michel De Rainville, M.A. Gardner, Marc Parizeau, and Christian Gagné. Deap: Evolutionary algorithms made easy. *Journal of Machine Learning Research, Machine Learning Open Source Software*, 13:2171–2175, 07 2012.
[8] Mohammad Hamdan. A heterogeneous framework for the global parallelisation of genetic algorithms. *Int. Arab J. Inf. Technol.*, 5(2):192–199, 2008.

[9] Ahmad Hassanat, Khalid Almohammadi, Esra'a Alkafaween, Eman Abunawas, Awni Hammouri, and VB Surya Prasath. Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach. *Information*, 10(12):390, 2019.

[10] Matthew P Thompson, Jeff D Hamann, and John Sessions. Selection and penalty strategies for genetic algorithms designed to solve spatial forest planning problems. *International Journal of Forestry Research*, 2009, 2009.