Evolutionary Computing

52848EVC6Y

Task 1: Specialist Agent

October 4, 2019

Group 44

Maarten van den Ende
maarten.w.j@gmail.com
2529117

Parva Alavian
parva.alavian@gmail.com
2686884

Laurens Stuurman
laurensstuurman678@hotmail.com
2654410

Lotte Heek
lotteheek@gmail.com
2600106

Comparing Two Diversity Mechanisms in Individual Evolution Algorithms for a Predator-Prey Game

ABSTRACT

This paper describes two evolutionary algorithms with different measures for maintaining diversity implemented for a predator-prey video game framework (EvoMan). Mutation parameters were tuned through a simple grid search. Randomly introducing new genes into the population as a diversity mechanism in an evolutionary algorithm produced significantly better results than deterministic crowding.

1 INTRODUCTION

This paper describes two evolutionary algorithms for the EvoMan framework of predator-prey games, referred to as enemies and agents respectively [1]. The goal for the algorithms is to improve the agents' performance as measured by a fitness function.

1.1 Evoman

Evoman is a simple Pygame game in which a character fights with an enemy in a two dimensional non-scrolling platform setting. The enemy has different abilities which the agent needs to dodge while also shooting and hitting its opponent. Both characters have a certain amount of hitpoints and as soon as these run out on either one the game is over and the next round starts.

In this research only a single round is used, with the inputs given to the controller being the values of 20 sensors: 16 for the distance between player and projectiles, two for the distance between player and enemy and finally two for the directions the player and enemy are facing.

Evoman has the possibilities to play against eight different enemies on multiple maps. In this research only a single enemy on a single map is used.

1.2 Evolutionary Algorithm

Evolutionary algorithms are based on the principle of natural selection, which refers to the survival of the fittest individuals in a population within an environment of limited resources [2]. It is not necessarily the case that only the fittest individuals progress to the next generation, as a matter of fact maintaining diversity in the population strongly influences the algorithm performance [3]. In this paper two different methods of maintaining diversity in an evolutionary algorithm will be compared. We expect that an algorithm maintaining diversity by means of deterministic crowding will perform better than an algorithm which does so by random means.

2 METHODS

2.1 Controller

The controller is a method in the Evo-Man framework that determines the action of the player sprite given the input of the sensors. We chose to implement the controller as a deterministic finite state machine. The states of the machine consist of a list of five binary numbers, each coding for a action that the sprite does or does not take. The state of the machine, from now on called agent, changes based on input from the sensors from the framework. These sensors are binarised and then the next state of the agent is determined by looking up the sensor input in the rule book of the agent. The rule-book of an agent constitutes of a dictionary with keys for all possible values of the sensors, and a list of five binary random numbers.

2.2 General outline of algorithm

2.3 Crossover

The crossover function in our algorithm combines genotypes of two agents at a random point to create either one[algorithm A1] or two[algorithm A2] children. This results in a proportion of 0.2 children being created of the population size for algorithm A1, and a proportion of 0.4 for algorithm A2.

2.4 Mutation

In our approach we only mutate the offspring of every generation. Mutation paremeters were tuned through a grid search (figure 1 2). The legend shows a string of parameter tomutate and nflip which represent the proportion of offspring to mutate and the amount of mutations to perform respectively. We found that a combination of values 0.9 1000 for these parameters worked best for algorithm A1, and second best for algorithm A2. For matters of comparability we chose to use these values in both algorithms. A mutation consists of replacing the output at a random index in the rulebook by a random output.

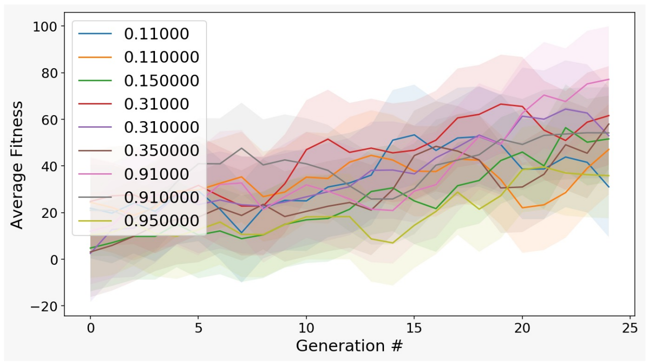

Figure 1: Plots of average fitness and standard deviation through generations for grid search for algorithm A1

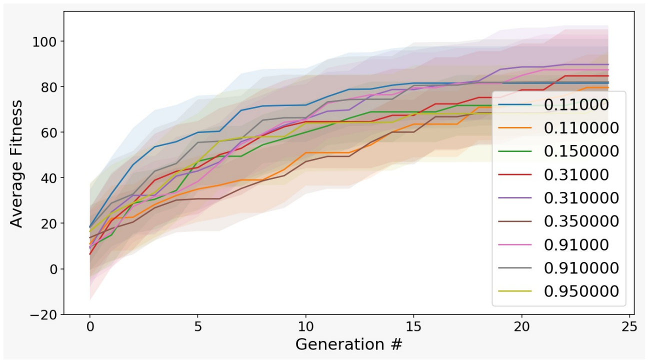

Figure 2: Plots of average fitness and standard deviation through generations for grid search for algorithm A2

2.5 Diversity

To reduce the chance of finding a local optimum instead of a global optimum we implemented two different ways of maintaining diversity in our populations. The first strategy(A1) for ensuring diversity, consists of randomly replacing a fraction of the population by completely new agents at the end of the evolution of a generation. The fraction that was replaced in simulations was 0.2 of the total population.

The second strategy(A2) is referred to as deterministic crowding[3]. Randomly paired parents produce offspring which undergo mutation before being evaluated. Offspring then competes with the most similar parent for survival to the next generation. This maximizes fitness in sub-populations or niches rather than in the whole population thereby maintaining diversity, aiming to prevent getting stuck in a local optimum[3][2].

2.6 Parameters

A population size of 20 was used evolving over 30 generations. This simulation was ran 10 times independently. Exploratory plots of the algorithms showed that fitness stabilized after 15 generations for algorithm 2, therefore it was decided to let the algorithm run for no longer than the double of this. Population size was chosen based on running time of the algorithms. The default fitness function was used, and the algorithms were run on level 2 for opponent 2.

$$fitness = \gamma * (100 - e_e) + \alpha * e_p - logt$$

3 RESULTS

In this section a statistical analysis of the different methods will be performed and discussed. The two evolutionary algorithms are compared to determine the best working one and their behaviour.

First of all the algorithms are compared by the amount of life (hitpoints) algorithms' characters have left over at the end of the game. In Figure 3 and 4 this is plotted over different runs for each generation. Even after more generations not a single population manages to have its members be close to having a perfect run. The maximum hitpoints is 88, while the maximum mean of any population achieved is 64.6. In addition to this the standard deviation is significantly high, with even in the final run many agents dying and loosing the game. Figure 5 plots a comparison of both algorithms based on life remaining at the end of the game.

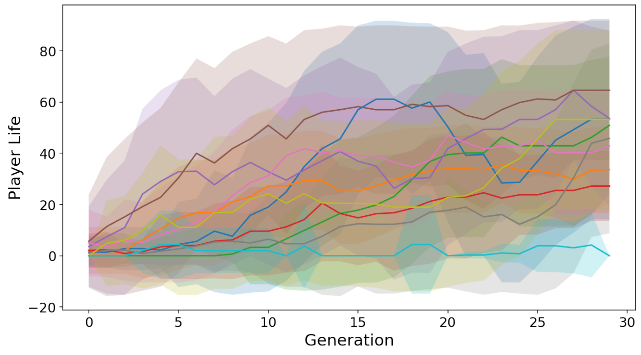

Figure 3: Mean and standard deviation of character life at the end of the game for 10 independent runs for algorithm A1

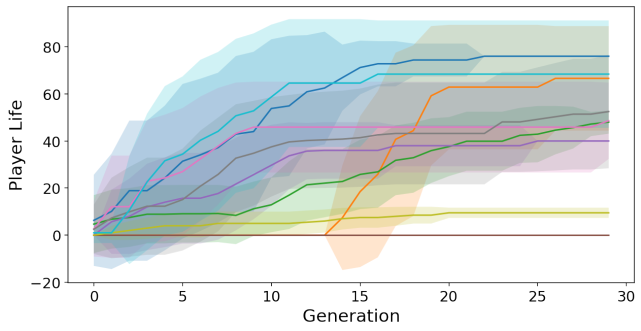

Figure 4: Mean and standard deviation of character life at the end of the game for 10 independent runs for algorithm A2

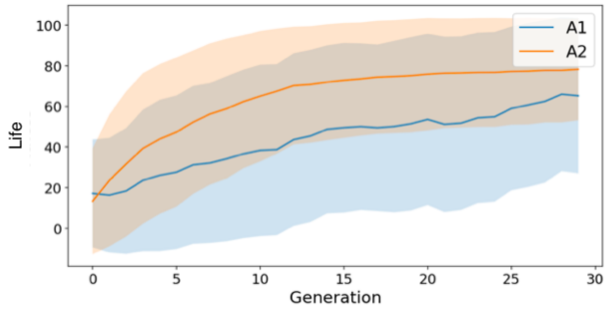

Figure 5: The average of the ten independent runs of the populations per generation for both algorithms.

In Figure 6 the average fitnesses of the ten runs and their standard deviations are plotted. It can be clearly seen that while A1 keeps having a high standard deviation, A2 manages to converge to a winning strategy, with its standard deviation reducing quickly. Its mean is very high, we expect that the algorithms have found the optimum strategy and defeat the enemy in as little time as possible.

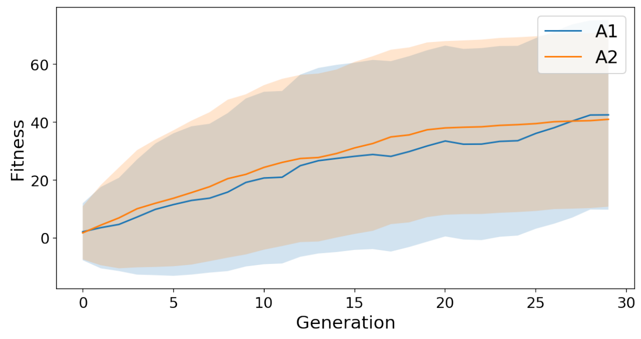

Figure 6: The average fitness and standard deviation averaged over the ten independent runs of the populations per generation for both algorithms.

In the Figures 7 and 8 the fitness functions of the different algorithms are shown for ten runs.

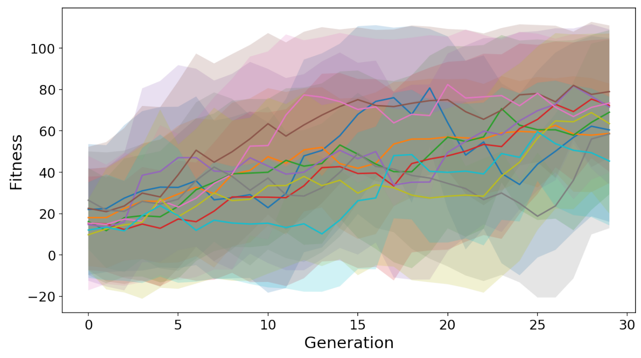

Figure 7: Average fitness and standard deviation throughout populations of each of the ten runs for algorithm A1.

4 CONCLUSION

Independent samples Welch's t-test for unequal variances (t=1.95, p=.08) was non-significant for alpha=0.05. This means the average fitness in the last generation (averaged over 10 runs) from the algorithms are significantly different, from which we can conclude that algorithm A1 performs better than algorithm A1.

5 DISCUSSION

The hypothesis that an algorithm maintaining diversity by means of deterministic crowding will perform better than an algorithm

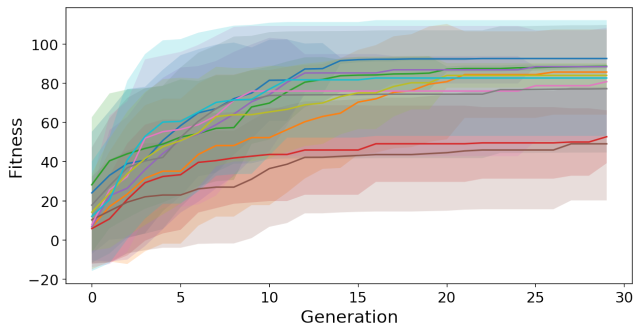

Figure 8: Average fitness and standard deviation throughout populations of each of the ten runs for algorithm A2.

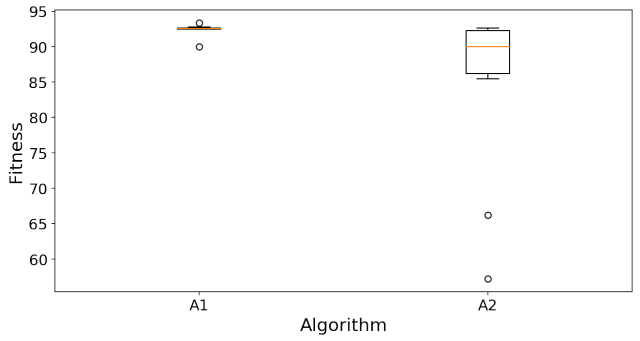

Figure 9: Average fitness of the average final population over ten independent runs for both algorithms.

which does so by random means was rejected. This can be due to a limited population size and a limited amount of generations. Deciding to let the algorithm run for 30 generations based on some exploratory plots was preliminary. Lastly, the used implementation of the deterministic crowding could be non-optimal thereby negatively influencing its performance.

6 REFERENCES

[1] de Araujo, K. D. S. M., de Franca, F. O. (2016, July). Evolving a generalized strategy for an action-platformer video game framework. In 2016 IEEE Congress on Evolutionary Computation (CEC) (pp. 1303-1310). IEEE.
[2] Eiben, A. E., Smith, J. E. (2003). Introduction to evolutionary computing (Vol. 53, p. 18). Berlin: springer.
[3] Mahfoud, S. W. (1992, September). Crowding and preselection revisited. In PPSN (Vol. 2, pp. 27-36).