Exam Number: Y3868453

EVAC: Evolutionary Algorithm

Methods

Genetic Programming

I decided to use Genetic programming to evolve both the snake and food agents for this coevolution task. For the snake's presets, I created several primitives for the snake, consisting of 'if' functions that detected if a wall/tail or food is nearby. I also made four terminal states which consist of the four directions that a snake can take. The idea is that a snake will evolve by detecting if an object is nearby by using its senses function and then using one of the four terminal directions to make a decision. I gave the snake more senses by adding an attribute for the left and right of the snake's head, allowing the snake to have more versatility in its environment. This would also detect if the food is left or right and be able to choose the desired direction. I lastly created four primitives called 'if moving' coupled with the direction; this helped the snake even more in its environment because it could have a better sense of awareness of its approach.

For the food presets, I created XSIZE - 1 x YSIZE - 1 (size of the grid) terminal nodes, each coordinate of the grid. I then had two primitive sets made, one to add two coordinates together and the other to subtract two coordinates together. The idea is when a food pellet is eaten, the snake's head coordinates are passed to the food presets, and the evolution algorithm evolves from learning the best possible coordinate to put the next food. I had to overcome a problem because the food placing method does not know where the snake exists on the board as it can not place a food pellet on top of the snake. Therefore, I implemented a while loop that calls the food placing function with random coordinates (other than the snake's head). This would only slightly affect the coevolution with randomness for a small percentage of the time.

Genetic Algorithm Toolbox

I had two separate genetic algorithm toolboxes for the snake and food populations. I will run these genetic programs concurrently, linking them to the evaluation function.

When creating my toolbox, I made my psets, so they start with at most a depth of 4 and with a least one node. This allows me to start with a variety of populations containing small enough trees to find incremental improvements in snakes' fitness.

For the snake genetic algorithm, I have included a selection algorithm with a tournament; I started with a double tournament but found there was a lack of diversity where all the individuals exhibited the same behaviour; when changed to a selection tournament, there was a growth in fitness because more diverse snakes began to evolve and were selected more easily. Also, the roulette selection algorithm doesn't work initially as some snake individuals had a probability of 0 fitness which cannot be computed with a roulette selection algorithm.

I used a one-point crossover on two individuals for mating the snakes, and for mutation, I used a Uniform mutation.

The preset tree will mate and mutate with a genHalfandHalf() algorithm that uses both genFull() and genGrow() of lengths between min and max of 0 to 3 depths, allowing a variety of depths to be added to the tree.

I used all the same selection, mating, and mutation algorithms for the Food genetic program. I also compiled the food in a different toolbox called 'toolbox2'. This is then linked with the food presets called 'pset2'.

In both toolboxes for the snakes and food population, I added a static limit so that the trees that represent the individuals that undergo mutation and crossover do not bloat to a large tree. This will prevent memory errors as it ensures no tree higher than the limit will ever be accepted into the population.

Evaluation Function

In my evaluation function, I started with just the score for the number of foods eaten by each snake to be passed as a fitness variable for the individuals. However, there was no incentive for the snake to move and has no idea about the environment it existed; I then implemented a fitness for the number of steps which gave the snake more incentive to move and explore. The score for the number of foods eaten is given to the fitness evaluation function, but with a higher weighting than the number of steps; this evolves snakes with the behaviour to seek out food.

I found that if the weighting for the score and steps are the same (1,1), the most common evolution is for a snake to take advantage of the step fitness, whereby the snake gets trapped in a cycle. This means the snake doesn't actively go out to hunt for food but seeks a higher step fitness. As I wanted the snake to be more incentivised by finding food, I then increased the fitness weighting for the score and steps to (2,1), respectively, while also calling the 'game_over' variable for that individual snake if it hasn't eaten any food in a certain number of steps.

The food coevolution is connected with the snake's fitness score with the same parameters; this is the score for the number of foods eaten and a total number of steps. However, the food agents' fitness has a higher weight for the number of steps taken. This gives the food agent the behaviour to find difficult places for the snake to move into. This intern creates a robust solution for both the snake and food agent. They both try to maximise their fitness, fighting over the best possible strategic individuals against each other.

As two fitnesses variables evaluate the score and steps for an individual, each time the evaluation function is called, I create a list with a 2-tuple. Each tuple represents an individual's fitness score passed onto a list that is the whole population's fitness score. The tuples (individuals) are also shuffled, so that snake and food populations play against random opponents other than competing the fittest individuals with each other. The 'run_game' procedure has been altered with additional parameters of the individuals, which allow the compiler to be called from when the game is running.

Genetic Algorithm Evolution

I created a population for each snake and food agent in the evolution loop. I separated each generation by calling the snake evolution first with selection, crossover, then mutation. Then I continued with the food evolution, copying with selection, crossover, then mutation. Lastly, after evaluating the individuals' fitness, I recorded both populations in their separated logbooks to compare how the fitness of each agent has adapted after each generation.

I will change the headless parameter towards the end of the evolutionary algorithm to see how each individual has evolved. This would help to visualise what the diversity of the individuals have become. If a snake or food population has become disengaged, I will use evolutionary strategies such as changing tournaments size for my selection algorithm to maintain diversity. I will also pay attention to the standard deviation, which is also a good indicator if a population has become disengaged.

Results

Because of the stochastic nature of the evolution algorithm, I need to repeat my algorithm a few times to test which parameters affect the population the most. I will implement a loop around the main evolution code that calculates the statistics of ten individual models that are saved onto a list. This ensures that after ten runs of an algorithm, I will be able to find how robust an evolution algorithm is under various initial conditions. I will need to keep some factors constant such as the number of generations, to keep consistency throughout the experiment. I will then calculate the mean and standard deviation of all the ten models into one list containing the average mean and average standard deviation so that I can compare different parameters.

For my experiment, I started with 40 generations for each algorithm. I choose to experiment with three variables. This will be crossover probability, mutation probability and the total population size. My first algorithm will be investigating a high crossover probability.

Total population size = 50, Crossover Probability = 0.8, Mutation Probability = 0.5

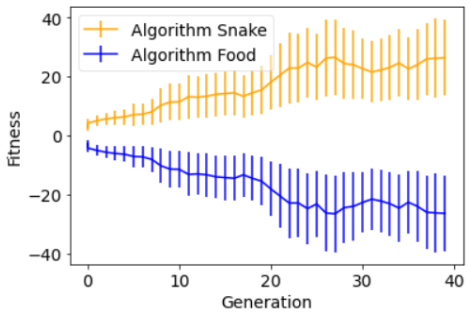

Figure 1

The first thing you will notice about my results is that both snake and food algorithms contain the same magnitude. This is because the evaluation function returns the same variables. However, the food algorithm has a negative fitness score to create a coevolution environment.

From *figure 1,* we can see the change of crossover probability has created an average fitness score for both models of 30 by the end of its generation cycle. The standard deviation also shows an apparent deviation from the mean, which helps predicts where to find the average fitness score. We can also see that the average fitness score varies around the 25th generation point, showing the food-placement agents starting to pick up new evolution to fight against the snake agents.

The next test I chose to conduct was to increase mutation probability and see how this affects the score.

Total population size = 50, Crossover Probability = 0.3, Mutation Probability = 0.9

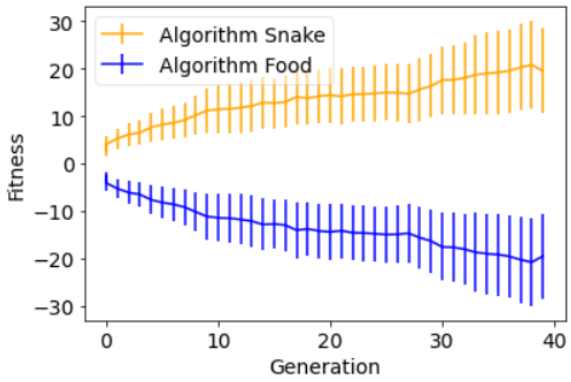

Figure 2

Figure 2 demonstrates a more gradual increase in fitness per generation. After these results, we can not fully construct a definite fitness score as the fitness score may still be increased passed generation 40. However, we can see that looking at the mean fitness and confidence intervals demonstrated by the standard deviation; we have a higher fitness score than the previous experiment suggesting a higher mutation rate is key.

My Last experiment conduts an increased population size

Total population size = 100, Crossover Probability = 0.3, Mutation Probability = 0.5

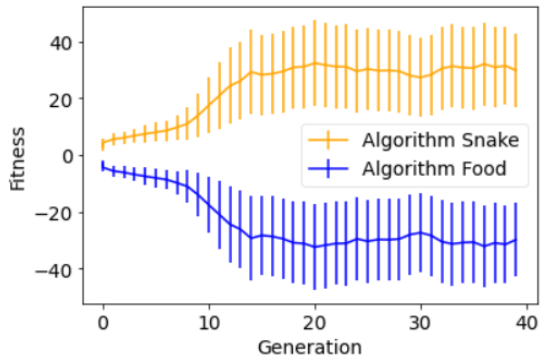

Figure 3

From *figure 3,* we can see when the population size doubles, there is more of a dramatic increase from the 10th generation to the 15th generation point. This will be

due to the format of the selection algorithm, as it is a tournament the best individuals will go through to the subsequent offspring. With a higher population size, more individuals exploit the best features earlier on, which has a dramatic increase in the generation cycle.

I will use a boxplot for these three experiments to compare how the mean fitness differ shown in *Figure 4*.

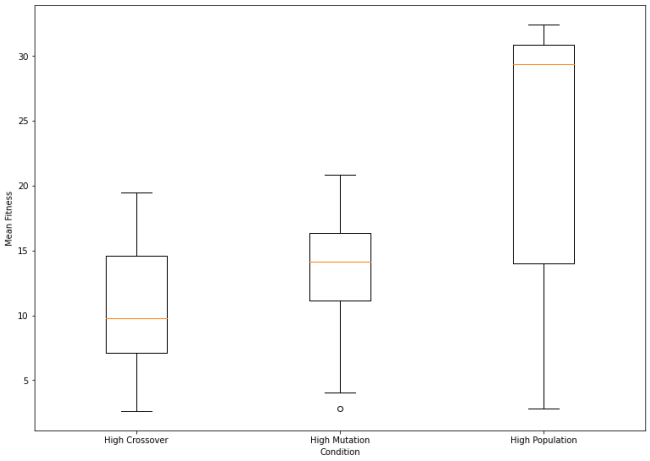

Figure 4

A high population has an apparent increase in mean fitness. Still, all three experiments have overlapping confidence intervals, so a U-test with a null hypothesis that the mean fitness is the exact needs to be used. After conducting a U-test between experiment 1 (high crossover) and experiment 2 (high mutation), I discovered a p-value of 0.031 with statistics = 305, which is enough evidence to reject the null hypothesis that the means are the same at a 95% confidence level.

Conclusion

The investigation shows that more robust solutions have evolved from both the snake and food populations when a high population rate is equipped with a high mutation rate.

However, in hindsight, as the food placement agents had a limited primitive set and a large terminal set, I would in the future either add ephemeral constants for the terminal nodes or add a neural network that could show more robust solutions and flexibility due to its 'black box' feature.

Several times I came across overspecialisation of the snake agents, which gave the food agents no chance to adapt as there was no incentive for the snake to go for the food. Therefore, an algorithm with fitness sharing could help minimise overspecialisation for more diversity.

End of Paper