NOVA IMS

COMPUTATIONAL INTELLIGENCE FOR OPTIMIZATION

Playing snake with genetic algorithms

Nuno Bourbon Carvalho Melo 20210681

Ivan Jure Parać 20210689

Stuart Gallina Ottersen 20210703

https://github.com/stuartgo/cifo

Introduction

In this project we aimed to implement a genetic algorithm (GA) able to optimize the set of weights of an artificial neural network so that it was able to successfully play a snake game. After implementing the game, we tested multiple combinations of genetic operators, selected the best ones, and carried out further experimentation by changing specific parameters, applying elitism, and modifying the fitness function (FF).

Initial exploration

Initially, we decided to implement the snake game, the state representation, and the FF with custom functions built from scratch. However, after extensive experimentation we realized the GAs were not converging, even after several generations. Therefore, we ended up retrieving the code for the snake game and part of our testing structure from similar projects to have a better baseline to experiment with. This resulted in fewer game bugs and successful convergence.

Once we had a model that seemed to improve over time, we implemented the genetic operators. Specifically, we implemented standard mutation, creep mutation, and geometric semantic mutation. Creep mutation is very similar to standard mutation, but the random value is less arbitrary as it is limited to the range between the highest and lowest values present in the weights vector [1]. Regarding crossover, we implemented standard, uniform, and geometric semantic crossover. Uniform crossover, specifically, selects a gene from either parent with 50% probability [1]. As for the selection methods, we tested fitness proportionate, ranking, and tournament selection.

During our first attempts we quickly discovered that applying genetic algorithms, at least to this specific problem, is computationally demanding. This forced us to move away from the brute force strategy of testing every possible combination of genetic operators with each version of elitism, state representation and ff. We wanted to do a preliminary test with each combination of the genetic operators, and decided to do this initial testing with elitism as it was implemented in many similar projects, we will get back to how elitism affects performance later.

Problem representation

To tackle this optimization task we built an artificial neural network with an input layer with 12 neurons (number of input parameters), a single hidden layer with 32 neurons (chosen rather arbitrarily), and an output layer with 4 neurons (the four possible directions the snake can move in). This network allows us to represent each snake, *i.e.*, solution, as a set of 512 weights $(12 \times 32 + 32 \times 4)$. The input layer is later changed to attempt another representation of the state of the game. The genetic operators are all written to use maximization as an optimisation technique, but equivalent functions for minimisation have been written where necessary.

In more detail, the input vector consists of 4 values that detect “danger” around the head of the snake, 4 more values representing the direction in which the snake is moving, and 4 other values representing where the apple is relative to the snake. After several failed attempts at implementing our own snake game and choosing our own input, the final decision was made based on existing projects. Using these 12 features as input expedited convergence, thus decreasing the number of generations needed to achieve convergence.

For each generation we have considered a population of 50 snakes, where each snake has made 5000 moves, meaning a snake can die multiple times within those 5000 steps, and it will be reset. For each generation, 12 individuals are chosen based on the selection operators. These are then used to breed the next generation in addition to being replicated into the next generation themselves (when elitism is enabled). This differs slightly from the approach mentioned in the booklet and the practical classes, where a new set of parents is selected for each crossover event. Here, the selection is made from a breeding pool, which adds an additional degree of randomness. Concretely, randomness is introduced by the selection functions themselves when building the *breeding_pool*, and again when selecting the individuals from the breeding pool.

For our initial testing we have adapted the FF from an existing project as we wanted to make sure that we had a working function before running 27 different models [4].

$$f =- 150 * deaths + 5000 * record - 1000 * \ penalty - 100 * steps\,per\,food$$

The result of the FF can be interpreted as the snake's score, which increases whenever a snake attains many points and decreases for one of three reasons. First, the score decreases if the snake dies. Second, a penalty is applied to prevent infinite loops, *i.e.*, situations where a snake keeps "going around the block", which is actually quite common in early generations. Specifically, the score decreases if a snake goes 200 steps without eating anything, which also results in the snake being reset. Finally, more efficient snakes are indirectly rewarded by punishing snakes that have a higher average number of steps per food eaten.

As for the parameters for the various operators we used a breeding pool of size 12, a creep mutation rate of 0.1, standard mutation rate of 0.05, ms for geometric semantic mutation of 0.1 and a tournament size of 15. Breeding pool size was kept from existing projects, while the mutation rates were set based on some preliminary tests with some guidance from literature[2]. The ms was set based on literature and tournament size was varied slightly initially based on values given in the booklet from class [3].

Initial evaluation

Naturally, the main evaluation used in GAs is the fitness function, but this value is not very informative for a human. As such, we kept track of statistics that are easier to interpret, such as the average number of deaths, maximum score, overall average score, and maximum average score (the snake with the highest average score in a generation). As a criterion to select the best models we used the mean average score of the last five generations.This created a nice separation where the top 6 models (out of 27) had a score that was at least twice as good as the model in the 7th position and below. The models selected are shown in *Table 1*, and an overview of their fitness values throughout the different generations is provided in *Figure 1*.

Table 1. Top 6 genetic algorithms for optimisation of the snake game, based on the average score of the last 5 generations. In total, 27 combinations of different mutation, crossover, and selection operators were tested.

Mutation	**Crossover**	**Selection**	**Avg of avg score in last 5 generations**	**Max score**
Creep	Standard	Tournament	1.95	36
Creep	Uniform	Tournament	1.32	11
Standard	Geometric	Tournament	1.32	16
Standard	Uniform	Tournament	1.22	10
Creep	Geometric	Tournament	1.11	20
Geometric	Geometric	Tournament	1.02	7

As seen in the previous table, one of the deciding factors seems to be the selection algorithm, given that tournament selection was used in every single one of the best models. This is likely related to the fact that, out of a randomly selected pool of individuals, those with the best fitness keep getting selected, and those individuals continue to perform well in the following generations.

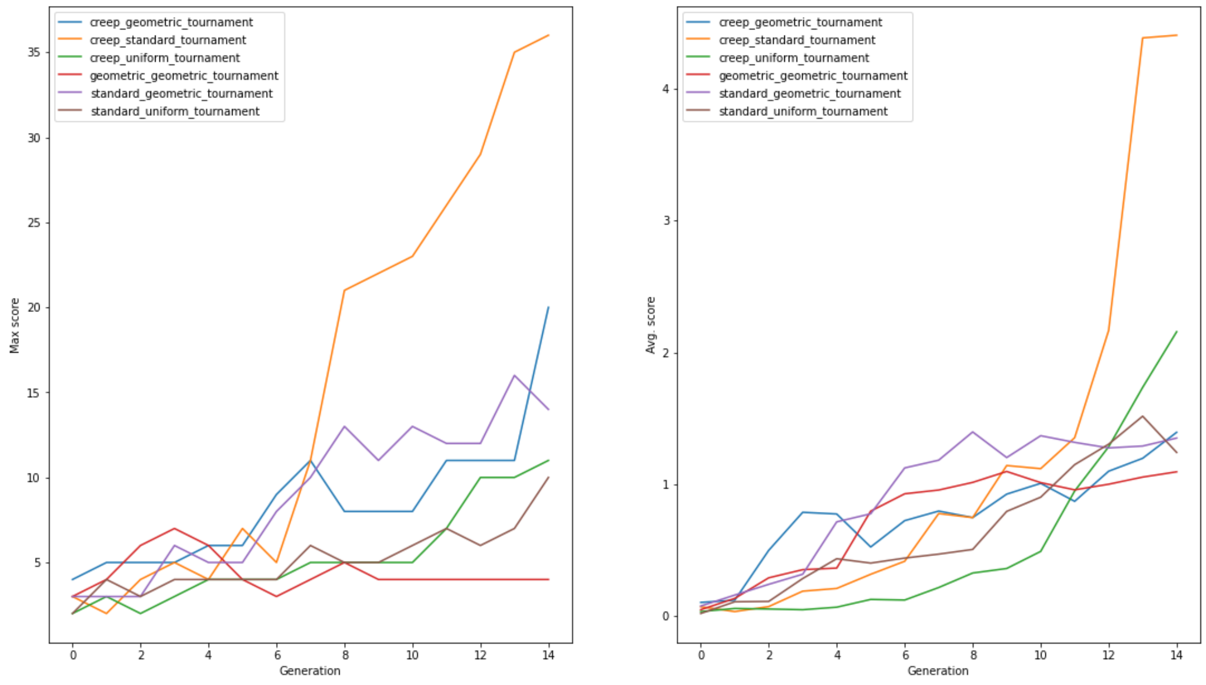

Fig 1. Maximum game score for each of the 6 best performing genetic algorithms throughout the generations. The algorithm leading to the highest maximum score used creep mutation, standard crossover, and tournament selection, and was also the algorithm leading to the highest average score in the last 5 generations.

It is important to note, however, that the maximum score achieved by each algorithm does not necessarily correlate with the average score. This is evidenced in *Fig. 2*, where the best model was run for 50 generations. Despite the ability of a few snakes to achieve a fairly high score, most snakes still have a weak performance and score less than 5 points.

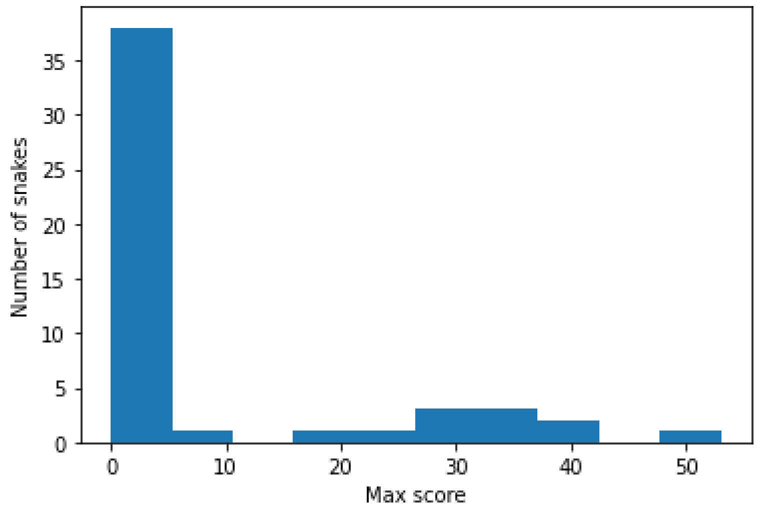

Fig 2. Distribution of snakes based on their maximum score for the best performing algorithm. The best performing algorithm used creep mutation, standard crossover, and tournament selection.

We acknowledge that selecting models in this manner and then changing only their parameters limits the scope of the experimentation, as non-selected models could have performed better with the changed parameters. However, this was one of the compromises we needed to make due to limited computational power.

State representation

The initial state representation we chose intended to capture the information that is essential for the snake to make decisions. We later went on to test a more complex state representation with a matrix representing the board with 1s where the snake is, 2 for the food, and 0s otherwise. This was run for 15 generations with little sign of improvement, and the maximum score for the best 6 models was always 3. This is probably due to the complexity of the state, capturing the mapping with the now much larger input layer to the output layer might have converged with more generations, but it's hard to say.

Testing fitness functions

As seen above the FF has a tendency to create a few very good individuals while the vast majority of the snakes have very low fitness and produce low scores. To improve this we experimented with a new FF:

$$f =- 100 * deaths + 8000 * record - 2000 * \ penalty$$

For one, the penalty of each death was reduced in an attempt to increase diversity in later generations. Furthermore, we increased the reward for achieving a record. The penalty was also increased so that an individual is punished any time it returns to an already visited state, to avoid looping snakes. Finally, we also tried removing the penalty for snakes that followed an inefficient path between current position and the position of the food, to promote exploration of the map. However, this FF led to results that were considerably worse than the previous FF. In fact, the best snake obtained using this modified FF had an average score that was about one-fourth of the score of the worst snake obtained using the previous FF. This was true even after training the model for a longer time.

Elitism

We tested the models both with and without elitism (*Fig. 3*). In the case of the former, the entire parent pool was copied to the next generation. In general, elitism greatly improved the results obtained with the algorithms applied. This was particularly noticeable with the best performing *creep_standard_tournament* algorithm. This makes sense, since elitism ensures that the best performing snakes of a previous generation are not lost when creating a new generation. Based on the results obtained, it is likely that these snakes continue to perform well as generations progress.

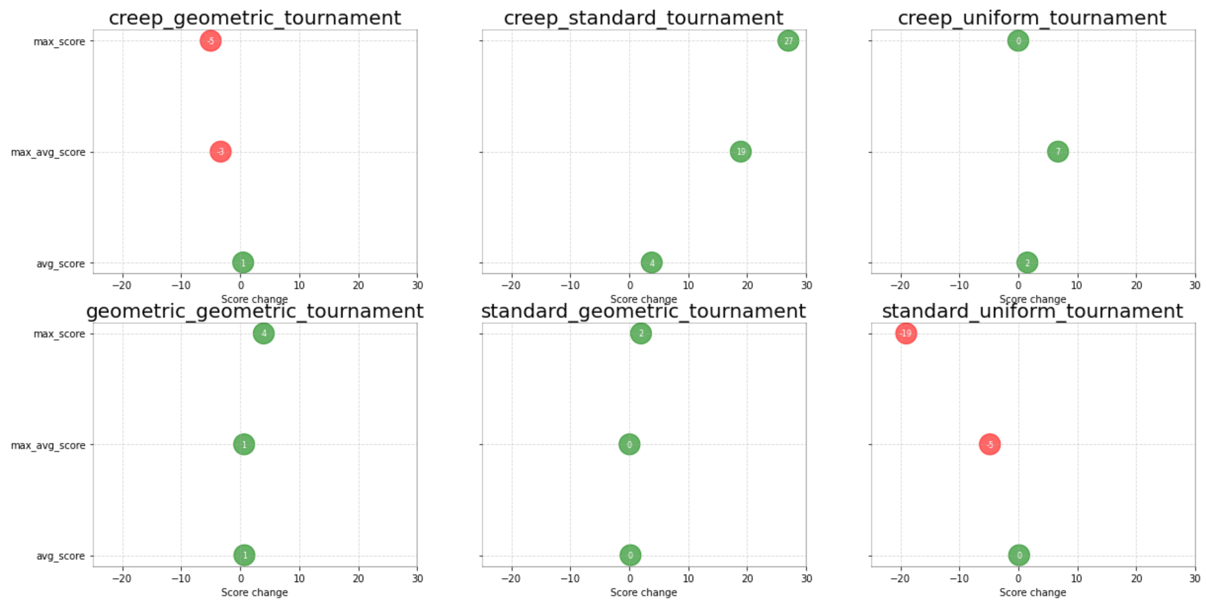

Figure 3. Changes in maximum score, maximum average score, and overall average score when elitism is applied.

Conclusion and further work

Overall, we consider that the main goal of this project, *i.e.*, building a model able to autonomously play snake, was achieved. When using the most promising GA to optimize the weights of the neural network the maximum score was 58 after only 50 generations, which we consider to be satisfactory. However, it is important to note that for every combination of genetic operators, the average snake got a fairly low score. Ideally, instead of producing only a few very good snakes, the model would lead to an overall improvement of the population over time. This can probably be achieved with further experimentation and fine-tuning of the fitness function and of the parameters used for the genetic operators. Changing the structure of the neural network might also improve the results.

Work distribution

Work was split evenly, with everyone contributing to both the code and the report. In fact, a good understanding of the code was required so that it could be run simultaneously in multiple computers.

[1] “Study of Various Mutation Operators in Genetic Algorithms”, 2014, Soni, Kumar
[2] “Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach “,2019, Hassanat, Almohammadi
[3] Geometric Semantic Genetic Programming Algorithm and Slump Prediction,2017, Xu, Shen
[4] https://github.com/davide97l/Snake-Battle-Royale