DATA SCIENCE AND ADVANCED ANALYTICS MASTER

SOLVING 2048

IMPLEMENTING A GENETIC ALGORITHM

CIFO PROJECT

Diogo Rasteiro • r20170826 @novaims.unl.pt
Mariana Camarneiro • r20170744 @novaims.unl.pt
Matilde Pires • r20170783@novaims.unl.pt
Ricardo Peixoto • r20170756 @novaims.unl.pt

Task definition

This project's primary goal is to implement a Genetic Algorithm that is able to solve the game 2048. This game consists of a 4 by 4 grid and its goal is to continuously join tiles with the same values to ultimately reach a score of 2048. The original code used in this project was taken from Pygame.

With this in mind, we chose for the representation a list of 1620 real values that is then converted to an array, that serves as weight in a Neural Network. This NN is composed of three layers, each with a weights array and a bias array, so overall these values compose a list with 6 arrays. Initially, the numbers were created using np.rand, being uniformly distributed between -1 and 1. However, the numbers might be out of this range, due to mutations and crossover operations performed over them.

To accommodate our model, several changes had to be made to the original game. First, the game was designed to be played by a human, so we made two big changes to the structure of the game: we added a Keras model to the file that would load the weights, and, in each turn, it would accept the game's board as input and return an int between 1 and 4 as output, which would then be converted to a move (up, down, left or right). Then we integrated this into our GA code in such a way that Pygame itself would be loaded as the program began. When we created an individual, the fitness function would load the weights into the Keras model, play the game, clean and reset the board at the end and simply return the score. This meant that we would only have to load the board and close it once per run of the GA.

Second, early models tended to get "stuck", as they would continuously output a move that would not change the board, causing an infinite loop. To combat this, we implemented a rule where if a model had the same score for a certain number of turns it would automatically "lose" the game.

At a first stage, our primary goal was to maximize the score obtained in the game, so the fitness function was defined correspondingly, by the sum of the numbers being joined at each play. At a later stage of this project, we decided to turn this into a multi-objective optimization problem, by adding two other objectives: to minimize the number of plays made to reach a score, and to obtain the maximum valued tile possible. Since the fitness function had to reflect this option, several attempts to adjust it were made to contain these multiple purposes and analyse the impact of prioritizing one of those objectives over the others. In order to clearly assess the evolution of the values in all three aspects (maximum tile, number of moves and score), we returned them as output, in all runs.

Approach

To reach the best possible combination of operators for our problem, we first had to decide which ones to try. So, we chose, for each:

- **Selection:** Fitness Proportionate, Tournament and Rank approaches.
- **Crossover:** Considering our individuals are composed by weights with real values, it is extremely unlikely that two individuals contain the exact same values. For this reason, it does not make sense to implement either the Cycle crossover or the Partially Matched

crossover. Therefore, we used only Single Point crossover (Standard approach), the Geometric crossover and the Uniform crossover.

- **Mutation:** Since we do not have binary individuals, it does not make sense to use Binary mutation. Instead, we resorted to Inversion mutation, Geometric mutation and a customized version of the Geometric mutation, detailed later.
- **Elitism:** All the tests are performed with elitism since it guarantees that the fitness of the population does not decrease throughout generations. Considering the limited time, we did not consider that for our problem it was crucial to run without elitism implemented, since we truly believe it is better to have it. For our case, we decided to adjust the elitism so that the best individual's fitness is recalculated on the next generation, to make sure that its performance is adapted to the environment it is given at each generation This way we make sure that the best individual is truly good in each generation and did not simply happen to achieve a good score once, in a better context.

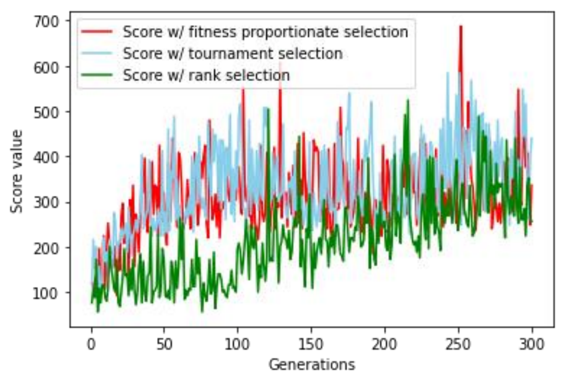
With the problem defined and our goal set, it is left to define the method/process to follow. So first, we will run the combinations of the three types of selection, crossover and mutation, to understand which combination performs better. Once the best combination is picked, we will dive deeper in that solution and fine-tune it, by altering some details in order to improve its performance even further.

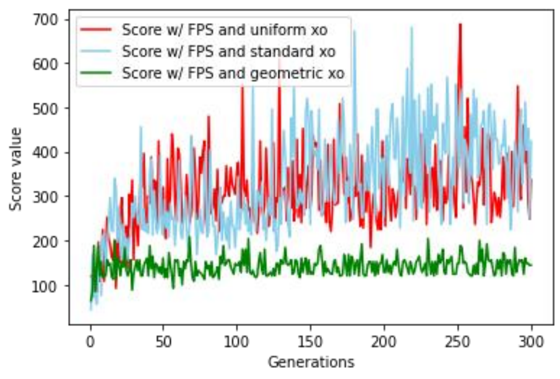
A challenge that emerged is that this game is highly influenced by a random factor (the tiles appearing at each move in random positions). Although this randomness would not be a critical issue if the GA were to run a high number of times, it is not computationally feasible to do so in our case.

An important detail to mention is that in order to achieve good results with a neural network it is necessary to run a high number of generations (other similar works have done it with around 700), and a high number of individuals per population to ensure diversity. This is computationally demanding and considering the limited time and computational resources of our group, it was not possible to run the GA sufficient times to reach the desired results. Still, the algorithm was run with 30 individuals for 300 generations, to ensure that we were able to see enough improvements to draw conclusions.

Results

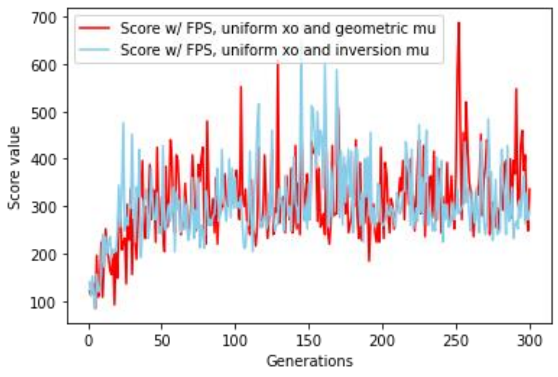
As for the selection, Fitness Proportionate, Tournament and Rank were run with the Uniform crossover and Geometric mutation as a baseline for comparison. As it is seen below, Rank selection is the one that reveals worse performance, while the other two show a similar behaviour throughout the generations, resulting in relatively similar scores. However, FPS still performed slightly better, reaching a fitness higher than 600, after more than 250 generations. For this reason, out of the three approaches, for our problem FPS is the one that proved to yield better results.





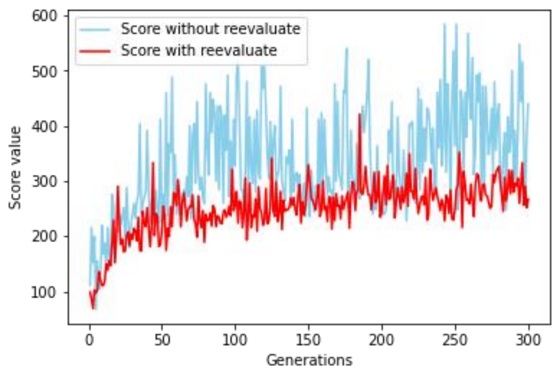
Concerning crossovers, geometric, uniform and standard were compared against each other, using FPS and Geometric mutation. Based on the graph below, again it is clear that one performs worse than the others, the Geometric crossover. Between the other two, they are quite similar, however Uniform was the one that achieved the highest score. Since this was the first decision point that was necessary to be taken, we chose to move on with this crossover method since we believe it has potential to replicate these high results.

Regarding mutation, we compared geometric and inversion at first, with FPS and geometric mutation as a baseline. From this we understood that, although they behave similarly, by the last generations, Geometric yielded slightly higher scores and there was also a peak after the 250th generation, with a score close to 700.



AVERAGE FITNESS EVALUATION

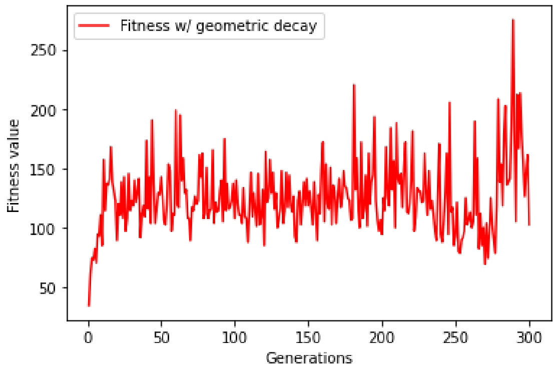
Due to the random nature of the game, there is no guarantee that a particular individual has a high fitness since they might have simply gotten a “luckier” seed. To combat this, we implemented an option where each individual was evaluated several times and their score was the average. This way, the final fitness would better represent an individual’s skill instead of their seed. Due to computational limitations, we simply made 3 repetitions.



As it can be seen on the graph above, the fitness of the population in this approach is generally lower (because the effect of lucky seeds was lessened). However, the variance of the fitness is much lower, being a slow and steady climb without large oscillations. Due to this, we conclude that this method of evaluation would be more precise and therefore preferred.

However, one major problem of this is the running time necessary. A normal run(without multiple evaluations) takes an average of 3 to 4 hours. However, when evaluating fitness for an individual 3 times, a single run took 22 hours. Given the time limit to deliver this project, we thought it might be infeasible to make all our runs with this parameter.

As it was possible to observe, geometric crossover was leading to worse results, stagnating at some point and showing little development. Since we were disappointed with these results, we decided to adapt the combination of geometric crossover and mutation. We believe that such results could be due, not only to the randomness explained previously, but also to the amplitude of the mutation change being constant, meaning that always having such a huge amount of change as we were getting closer to the desired results might be the reason for not converging. Finally, as the scores were only being evaluated once, a good result might be due to occasional luck in that specific run. As a solution, we implemented geometric mutation with decay to enhance a lower mutation change amplitude, decreasing from generation to generation, shown further below.



Once again, although there is a slight improvement compared to the one seen on the Crossover section (in green), it is not significant enough to justify the computationally demanding operations we implemented.

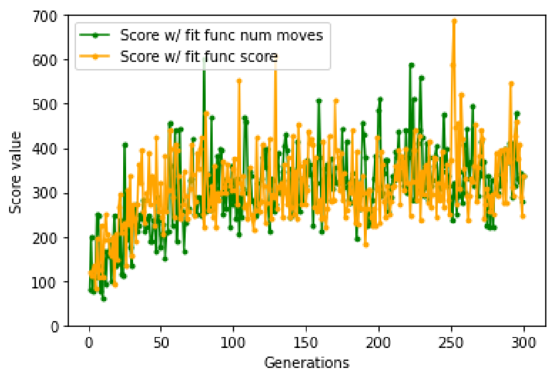
DIFFERENT FITNESS FUNCTIONS

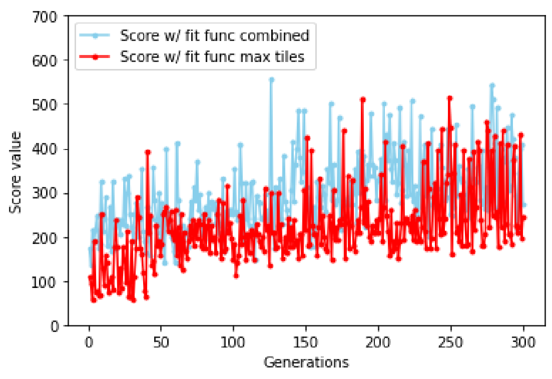
As stated above, our baseline fitness was simply the score achieved during the game. However, we were interested to see the impact of different fitness functions in the performance of the individual, in terms of max tile achieved, number of moves performed and ultimately the score.

Additionally, we discussed the implementation of a multi-parameter fitness function and the analysis of the results obtained.

$$\text{combined} = \log_2(\text{max\_tile}) \times \text{score}^2 \times \frac{\text{score}}{\text{num\_moves} + 1}$$

The rationale behind this fitness function is, to reward higher tile values, since they result of merging multiple tiles, reward higher scores since they are the most primitive assessment of success and lastly, the fraction to untie and penalize scores that were achieved with a higher number of moves.





Bellow we have a table with the register of the best Individual achieved with each fitness function and the impact this fitness function had on the other metrics.

Fit Func used	Generation	Max Tile	#Moves	Score	Combined
Max Tile	249	64	125	516	6 542 290
Nr Moves	80	64	154	600	8 361 290
Score	252	64	174	688	11 165 510
Combined	126	64	140	556	7 314 026

Conclusion and Further Work

After analysing the performance of the different experimented combinations, the chosen one ended up being a genetic algorithm with Fitness Proportionate Selection as selection criteria, Uniform Crossover and Geometric Mutation, the best for providing diversity to the population. As fitness function we opted for the score since it was the one the achieved better results more consistently.

Overall, we were satisfied with the results obtained, considering the constraints in time and computational resources explained before.

Nonetheless , the fine tuning of parameters was not achieved totally, since exploring all the combinations was not feasible. Exploring the impact of different number of individuals per population would be interesting, as well as increasing the number of generations to fulfil the game objective of achieving a tile of 2048. Also, trying different neural network structures would be interesting to see if the observed behaviours would change.

Furthermore, we would also invest more attention in exploring different designs of the fitness function since we understand we might have given too much weight to the score by using the exponential, for instance. Therefore, it would be interesting to observe more carefully the impact of slight adjustments to the fitness.

Additionally, upon searching for similar works we saw different types of mutation, like the Gaussian mutation, and different types of crossovers like the Shortest Path Based Crossover that have been successfully implemented for similar purposes and we would have liked to analyse and implement.

Lastly, we also discussed the implementation and impact assessment of implementing fitness sharing but, due to time restrictions we ended up discarding this hypothesis, nonetheless, would be of our best interest to register the results of such decision.