Project 4

Ezekiel Sung: 219507107
Hung Truong: 301035748
CSC 180-01
Due:4/23/21

Problem Statement

Our goal for this project is to have hands-on practice with genetic algorithms by using Distributed Evolutionary Algorithms in Python (DEAP). We will use DEAP to solve the 8 queens puzzle where it is the problem of placing eight queens on an 8 x 8 chess board so that no two queens attack each other. Queens can either attack each other vertically, horizontally or diagonally.

Methodology

We will be using Google colab and DEAP to create 8 x 8 chess boards that satisfy the 8 queen rules. The first task we will do is using a position-index-based board representation where an 8 x 8 board will have an index from 0-63. We will start by creating a FitnessMin with a weight of -1 because we want boards with the lowest fitness. Next we create a list of individuals with the creator and FitnessMin to hold many solutions. Finally we use the register method to define our functions so DEAP knows how to instantiate each individual and the first population.

Then we need to define the fitness evaluation for each individual. We will create a function called evaFitness and count how many pairs of queens can attack each other. The function will take an individual that is position-index-based and the function will scan for possible attacks vertically , horizontally and diagonally across the board. Each attack will be added together and that will be the fitness value for that individual. The final fitness would have to take into account if multiple queens are at the same index, which is why we will create a function called checkDuplicate that will calculate the number of queen pairs in the same position for a given individual. With the amount of duplicates, we will multiply that count with a fixed penalty of 50, that way the increased fitness for duplicates will discourage future generations to do the same. The final fitness sum of the individual will then be the checkDuplicate fitness value and the possible attack count.

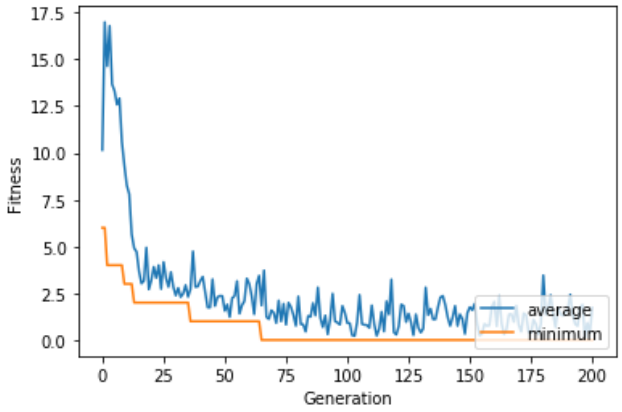
After defining the EvalFitness function, we will define the genetic operators and register the evaluate function in our toolbox. After having our algorithm all set up, we can let our algorithm run and study the results of each generation. To help us with the analysis of the results, we will use tools such as Statistics and a Hall of Fame. Statistics will help us display the results in graphs and the Hall of Fame will keep track of the best individuals that appeared during the evolution. We will compare different results of the simple evolutionary algorithm by tuning the different arguments: population, probability of crossover, probability of mutation and the number of generations.

The other task is very similar but each individual is a row-index-based board representation. All the steps are the same except for calculating the fitness value of each individual. In the evaFitness function, we only will need to calculate the possible attacks vertically and diagonally because in a row-index-based board, it is impossible to have a horizontal attack. We also will not be required to check for duplicates because it is also impossible in a row-index-based board. After defining the EvaFitness function, we will follow the same steps and also compare the results with the previous board representation.

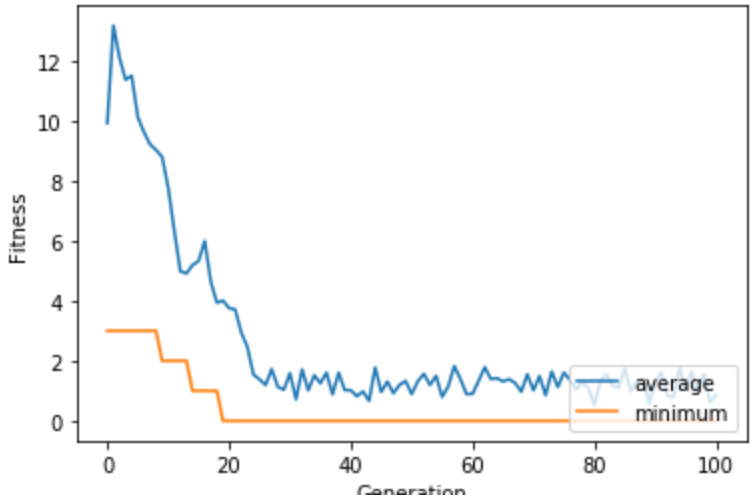
Experimental Results and Analysis

Position-index-based

We varied the different arguments to see how it affects the average and minimum fitness value for each generation. One thing that was very clear is that increasing the generation did not help steadily decrease the fluctuating average fitness per generation. We found that the sweet spot to be around 15 generations where the lowest minimum is usually reached and any more generations will not decrease the average of fitness.

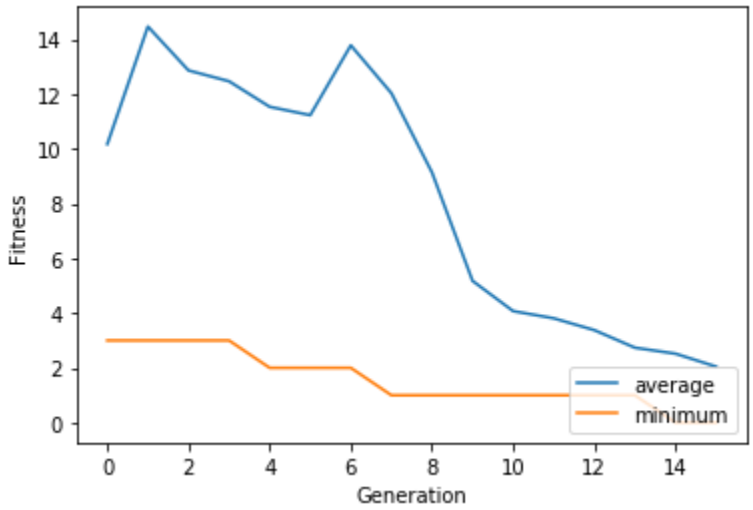


We found that increasing the population from 100 to 200 helped to actually reach a minimum fitness of 0 and helped to decrease the average fitness in the later generations. With the population at 100, the average fitness of the last few generations ranges from 2-3, whereas the average fitness of the last few generations with population 200 ranges from 0.8-1.8.



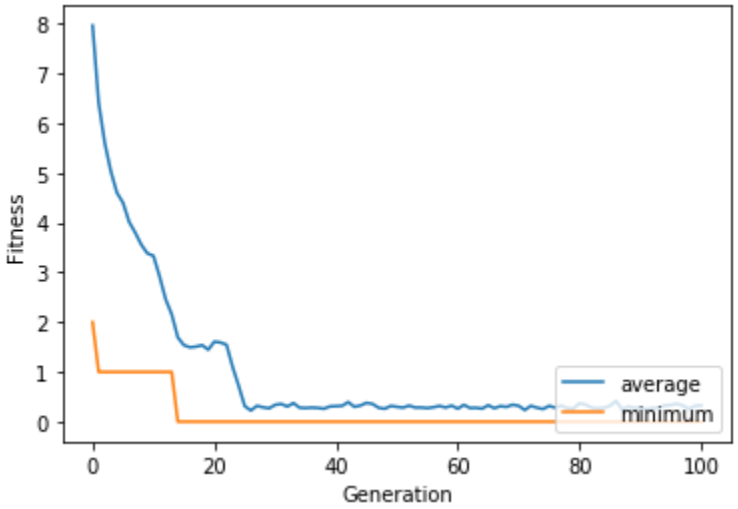
For crossover and mutation probability, we could not exactly pinpoint an ideal range that will give us the best results. What we did find was that the higher both these probabilities were, the more unstable the average and minimum fitness were. If the probabilities were too low, it will also prevent the algorithm from reaching a 0 fitness value minimum.

Our best parameter settings were a population of 500, crossover probability of .5, mutation probability of .2, and 15 generations. With this algorithm we were able to reach a 0 fitness value minimum very quickly and steadily but the averages of each generation is not as long as it could be. If we increased the generations a bit more it would also decrease the average fitness value.

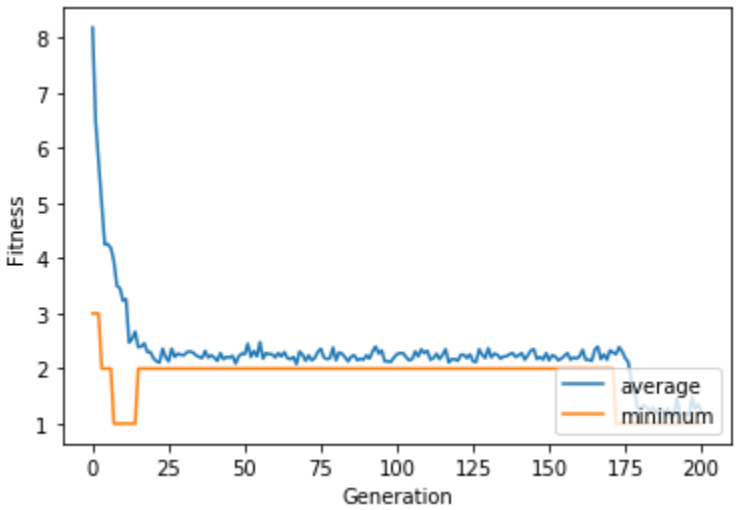


Row-index-based

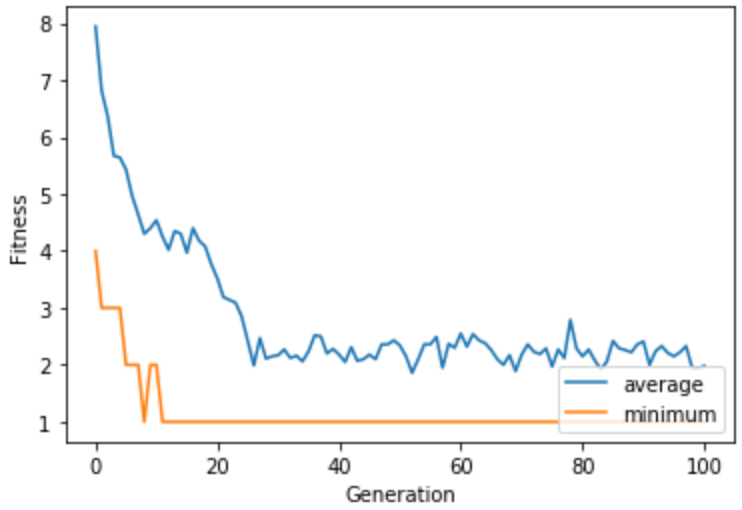
Just from the initial trials we can tell that Row-index-based representation is far better than position. From initial trials we can see that the average fitness is lower and more consistent than the position-index-based representation. Similar to the findings of the position-index-based algorithms, we also noticed that increasing the population to 500 also yielded us better results. The average fitness value was about .3 towards the later generations and it was more consistent.



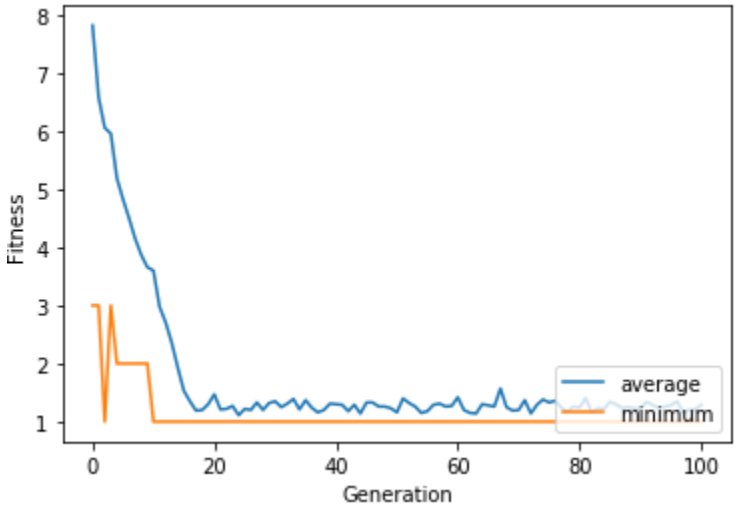
Increasing the amount of generations also gave us better average fitness but at a certain generation the difference was not that significant. If we disregarded having the best average fitness and focused on only reaching the 0 fitness value minimum, we found the sweet spot to be also around 15 generations.



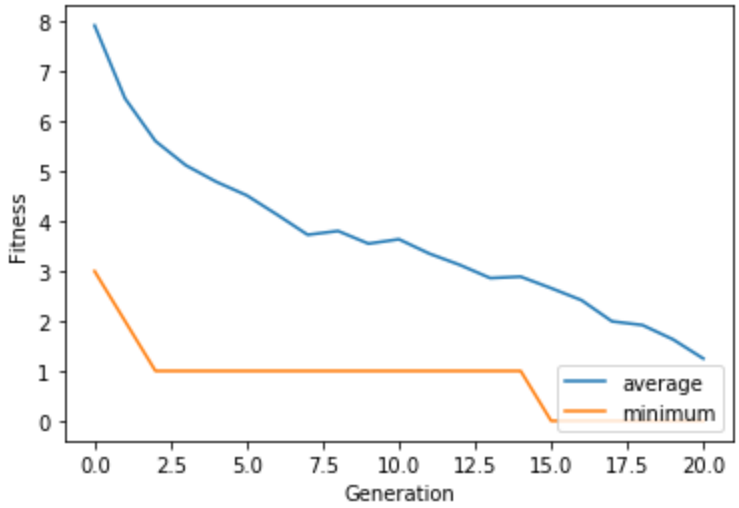
We noticed that increasing the mutation probability gave us a higher average fitness which is why we decided for our best algorithm we decided to keep it in the middle.



For the crossover probability, we found that having a slightly higher probability resulted in a bit more inconsistent average fitness value.



After comparing the results, the best results given to us was from the algorithm with the following arguments: population of 500, crossover probability of .5, mutation probability of .2 and a generation of 20. This was to quickly and consistently reach a minimum value of 0, if we aimed to have lower average fitness we can simply increase the amount of generations.



Task division

_____Hung was in charge of the position-index-based board representation and Ezekiel was in charge of the row-index-based board representation. We both worked together on figuring out the evaFitness for both representations. We also were in charge of building the algorithm, tuning the different results and comparing the results and findings of our respective board representation. Finally Hung was in charge of making the video and Ezekiel was in charge of making the report.

Reflection

In terms of ease of coding and final solution, we found that the row-index-based board to be significantly better than the position. It was easier to code because we did not have to detect attacks horizontally because it is impossible. In addition it was easier to detect attacks vertically and diagonally because we only had to account for the row index instead of modding the index with row index for position-index-based. In terms of solution, it was much better because it did not have to account for duplicates and horizontal attacks, which made the average fitness lower for row-index-based.

We both thought that it was very interesting and fun to attack an AI problem with a completely different method we have learned before. It is very cool to use DEAP to solve a problem that is not exactly an evolution problem but be able to manipulate our

problem to use DEAP. It is very different from our usual modeling and learning from a dataset but instead to treat it as evolution and evolution is a very cool approach for this problem.