Sacramento State University

Project 4 - Report

CSC 180 Intelligent Systems

Due: April 23, 2021 @ 11:00am

Logan Hollmer (**ID:** 301559973)

Quinn Roemer (**ID:** 301323594)

Problem Statement:

In project 4 we were tasked with solving the N-Queens problem using a genetic algorithm through DEAP (Python library). The N-Queens puzzle defines the problem of placing N queens on a chessboard of size NxN in such a way so that no queen can attack another queen. Solutions exist for all N, except N=2 and N=3. The first step to solving this problem with a genetic algorithm is finding a way to represent the queens in code. With this in mind, we will test two different options, positioned-index-based and row-index-based. The first one defines the position of a queen on the board by directly correlating it with the location specified with the index of the board. The second method records the position of the queen on a specific row defined by the index. Once implemented, a method will have to be established to measure the fitness. This can be done by measuring the number of distinct pairs of queens that can attack each other. The lower the value returned (with 0, meaning the board is solved) the greater the fitness of the individual. Once implemented, we will use built-in algorithms for evolving our population (varied) over a given number of generations (varied).

Methodology:

As usual, in tackling this project, each team member worked as an individual and fully finished the project on their own. Each person produced their own evaluation function and trained their own populations. Although different parameters were often changed, thus producing varying results.

While creating the evaluation functions our strategies differed. For example, Quinn created a lengthy function that focused on evaluating the board as efficiently as possible. This was done by considering each queen separately, checking for possible routes of attack through all possible means. In contrast, Logan produced a function that checked every queen against every other queen. Verifying that no routes of attack existed.

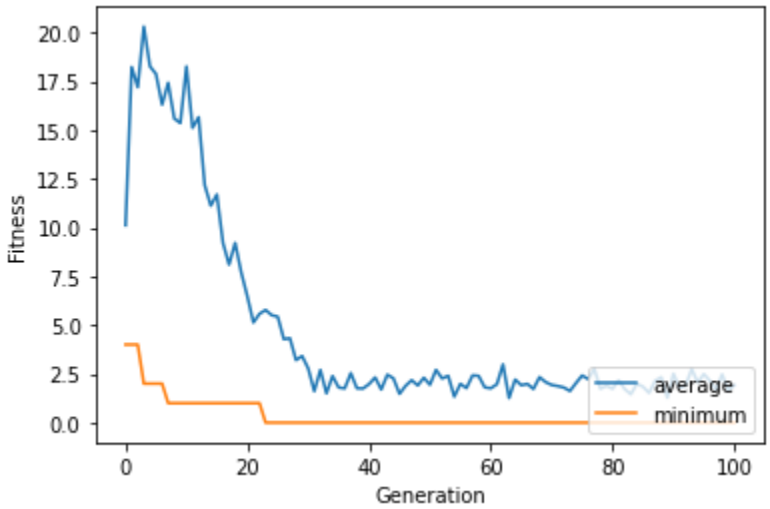
Once the evaluation functions had been created and registered in DEAP, we both proceeded to evolve our populations over a specific number of generations using the *eaSimple()* algorithm provided by DEAP. Quinn trained with population sizes of 64, 128, 256, and 512 with both 256 and 512 generations for each board representation type. This totaled 16 individual groups trained. Most found a solution to the 8x8 queen's problem, though some did not, with those having a higher population size finding the solution sooner on average. Logan trained with a number of variations as well, with his most successful group having a population size of 1000, finding a solution within 10 generations.

Lastly, both team members performed some extra work in attempting to solve boards of a higher order. Quinn attempted and solved boards of size 16x16, and 32x32. However, he failed to find a solution for the 64x64 board. In doing this extra work, he modified the mutation strategy used and increased the mutation chance so as to introduce new genes into the population at a higher rate. The results were generated with a population size of 100, with an unspecified number of generations (the program halted when the solution was found). Likewise, Logan pursued this same route and found solutions for all the above, plus the 64x64 and 124x124 boards. He accomplished this by changing the mutation strategy and increasing the mutation chance (similar to what Quinn did). Logan also increased the size of the tournament in the selection strategy. Lastly, he switched out the *eaSimpe()* algorithm for the *eaMuPlusLambda()* algorithm. The results were generated with a population size of 10000, trained for 500 generations.

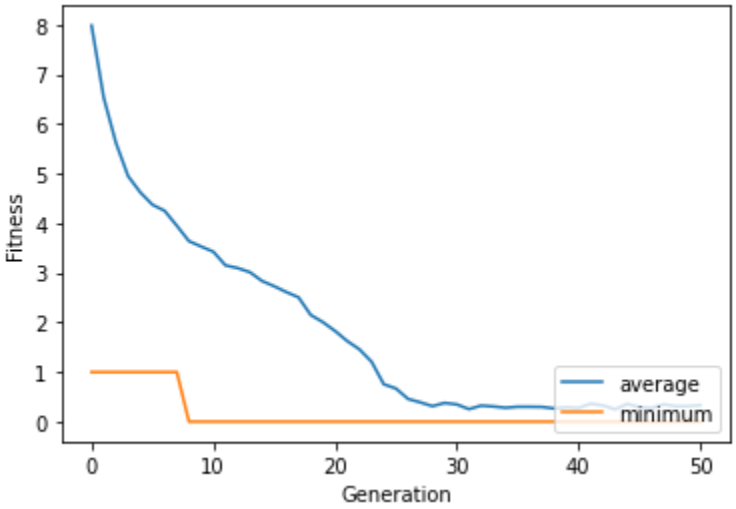
Experimental Results and Analysis:

As stated earlier both Logan and Quinn solved the 8 Queens problem using both the row and index-based representations. Once implemented both strategies quickly found solutions by varying the population size and number of generations. Below are the graphs of two successful solutions generated for the 8 Queens problem each for differing representations.

Graph of Index-Based 8 Queen problem using Genetic Algorithm



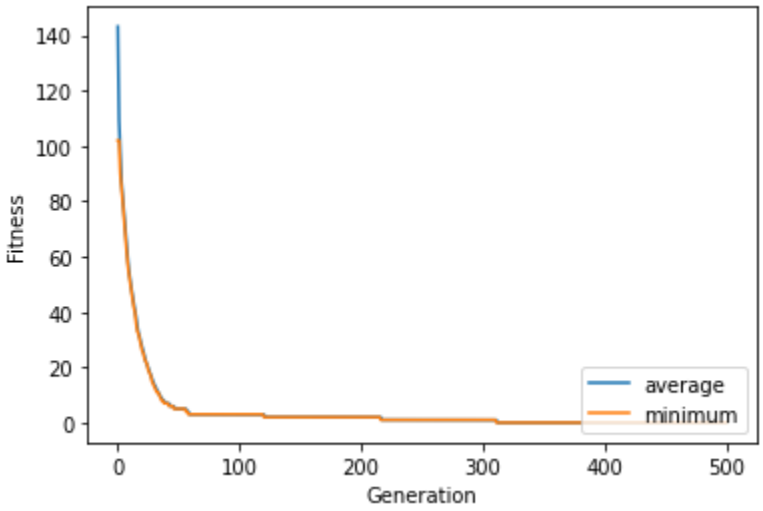
Graph of Row-Based 8 Queen problem using Genetic Algorithm



Both Quinn and Logan found that their row-based implementations performed better than their index-based implementations. We believe this to be due to the fact that the row-based implementation strategy guaranteed that only one queen can exist per row, with no duplicate positions possible. Although, there is still the possibility of collisions on the columns and along the diagonal. We also found that having larger populations seemed to help more than increasing the number of generations we trained over.

Lastly, we also created a function to solve the N-Queen problem for arbitrary N values, and successfully solved up to the 124 queen problem. When solving the 124 Queen problem we again found that using a higher population size helped more as opposed to increasing the number of generations. In the end, it took slightly over 300 generations with a population of 10k to solve the 124 queen problem. We also found that using a different evolutionary algorithm such as *eaMuPlusLambda* was helpful in achieving results faster. Finally, we also increased the tournament size to 1k. This led to much less diversity but also seemed to help our population come to a solution faster. Below you can find the graph for the 124 queen problem.

Graph of Row-Based 124 Queen problem using Genetic Algorithm



Task division and project reflection:

For task division, both members did the entire project themselves. Quinn wrote the P*roblem Statement* and *Methodology* sections of the report with Logan writing the remaining sections. Quinn also did the presentation and compiled the notebook with both members' code.

Reflecting back on this project, we feel that the row-based implementation strategy was vastly superior to the index-based strategy. This was due to the way the queens were represented, removing a significant portion of collision conditions, resulting in faster training. When looking at the N-Queen problem in general, we feel as though with enough time and resources any N-Queen problem could be solved with a genetic algorithm. However, an ever increasing amount of resources and computation time would be required, making such a strategy infeasible.