report

CSc-180 Project 4

Alec Resha

CSc 180-02, MWF 11:00-11:50

Problem Statement

The problem being addressed by this project is to create a program that will solve the N Queens problem. This problem is finding out the best way to place N Queens, 8 in this case, on a chess board so that no two Queens attack each other. There is no one set algorithm for this problem since the starting position is random, so machine learning is used to repeatedly mutate a chess board towards a solution that works.

Methodology

The DEAP library is used for creating our genetic algorithm. We first make a toolbox and fill it with the necessary functions. These functioons are: fitnessMin, Individual, population, evaluate, mutate, select, avg, min, and mate. The toolbox is then used to create our population and the genetic algorithm. In each case, the population is made entirely random. For position index based, the numbers represent the location from 0 to 64 possible locations. For row index based, each position is a new row, with a value from 0 to 7 representing the column.

Experimental Results and Analysis

For position index based, I found that the min of 0 was found faster with a larger population, but was found more consistently as the number of generations increased. The most consistent result that I found was a population of size 300 with a number of generations of 400. As both of these numbers increase, the likelihood of getting 0 conflicts increase, but so did the run time. These numbers are a good balance of speed and accuracy.

For row index based, it was more accurate with fewer generations or a smaller population than the position index based solutions. I settled on a population of 100 and a number of generations of 300 for consistently finding a valid solution. It was able to achieve it in less than 1/4 the time of the position index that I found (2.2 seconds vs 0.5s) As population and number of generations increased, the accuracy increased but so did run time, same as position index based.

Task Division and Reflection

This project was done alone so all work was done by myself. The largest problem that I encountered was the format of the evaFitness output. When making the evaFitness function for the index-based portion, I thought the comma in the return was a typo and kept getting an error when using eaSimple saying an integer had no method len. Besides this, the largest issue was determining what test algorithms worked best since each run was different.

This project helped me make sense how the fitness functions worked most of all, but also made the different functions for DEAP a lot more clear on their purpose.

Row-based was the easier implementation. The code was identical except for the fitness function. In the row basded implementation, we did not need to worry about them being in the same row which simplified the algorithm, which was the main difference.

I think this could easily be adapted to a larger N Queens problem, it would just be a matter of changing some of the setup functions and increasing the number of generations and/or population.