Joshua Bicera

Professor Tang

CS 4200

07/30/2023

Project 2

Approach:

1. Data Structure: In this program for both algorithms I utilized a class called Chromosome() that would take in a size of the board as a parameter. The Chromosome object stores both its size and its genes that represent the board. Rather than a 8x8 matrix the board is stored a single size 8 array of integers where each cell represents the columns of the board and its value represents the row position of the queen on each column. With this approach I have a way to generate random solutions to the board that I can use for both algorithms.
2. Steepest Ascent Hill Climbing: With this algorithm it accepts one randomly generated Chromosome object and attempts to solve that instance of the board. The fitness function for this algorithm aims to maximize the fitness values by evaluating all the attack pairs of queens on every neighboring state of the board. This means it only performs "greedy" moves where it chooses the best neighboring board at the time. Additionally, this algorithm does not allow for "sideways moves" as it only chooses neighbors with greater fitness values or always moves uphill. After looping and finding the best fitness value of all the neighbors then it returns the solution before the max iterations are reached.

3. Genetic Algorithm: For this algorithm the idea is to have a given population of solutions and to evolve that population over time until one solution gains the most fitness value. This is done through a probability function and a mutation function that modifies the population where probability is calculated using the the total fitness value of the entire population. This makes it so higher fitness valued chromosomes are more likely to be chosen. Additionally, each offspring has a small percent chance to mutate and modify a gene of the chromosome to a completely new value. As it iterates through the population it creates a completely new population by crossing over two chromosomes and potentially mutating it. It also checks each new chromosome whether or not the crossover and mutation results in the max possible fitness and if it does break out of the loop. If the generation of chromosomes doesn't come up with a list then the new population replaces the old one and continues to mutate and crossover once again.

Analysis:

In this project I wrote the required steepest-ascent hill climbing algorithm and chose to write the genetic algorithm among the choices. As shown in the screenshots of my code running only 12-15% of the 100 instances of an 8-queen "solution" were able to be solved. In my code I made it so that there is a max amount of iterations the algorithm could run before it is deemed unsolved. This meant that for roughly 85-88% of all solutions it would have taken the algorithm more than 1,000 iterations and possibly much more to solve. This inefficiency is most likely caused from both its limited ability to have no sideways moves or explore same fitness valued chromosomes causing it to not escape local optima and its greedy nature causing it to overlook more valuable exploration in other neighbors.