Philip Sloan

CS 420

Project 2

November 13th, 2016

**Solving n-Queens using Steepest Hill and Genetic Algorithm**

***Introduction***

The n-Queens problem is a well known problem in computer science. For this assignment , N-queens was solved using Steepest Hill climbing algorithm, which would not solve every problem as well as a Genetic Algorithm, for which the goal was to generate a solution from the randomly seeded population. Both algorithms proved to be interesting insights into how hill-climbing algorithms operate. The genetic algorithm especially provided numerous challenges, as it was highly sensitive to changes in the parameters.

***Hill Climbing***

To represent the board in the n-queens problem, several approaches are possible. For this program, an Object-Oriented approach was used. It would be possible to simply represent the board as an array of length n, with each element representing a column, and the value of the element representing the position of the queen in that column. Ultimately, I created a board class so that the code would be more readable, and that each board could keep track of variables such as the hill climbing cost, and later, the fitness value for the genetic algorithm.

The algorithm would check all of the neighbor board states and move the queen to the spot corresponding to the lowest cost board state. In the case where multiple moves had the same cost, a move would be selected at random from the possible moves. If the current state of the board was the lowest cost state, the algorithm would break, even if the state it was in was not a solution.

As you can see in figure one, the algorithm performed as expected, solving roughly 14% of the problems at n=8. As the board becomes large the percentage drops off significantly and is only 2% by n=19. Average time increased as n increased, from .031 ms at n=8 to .64 ms at n=19. The average number of moves only increased slightly from 3 at n=8 to 7 at n=19.

Figure 1: Percent solved vs Size of board.

Overall, the steepest hill climbing algorithm may be useful for some problems, but it is not especially useful for nQueens, as it can get stuck in unsolvable states rather quickly. If a form of backtracking was implemented, then the performance would likely be increased. Also, calculating the costs is computationally expensive, which would be true regardless of the algorithm used. If there were added constraints to how the board was initialized (i.e only one queen per row and column) , then the cost calculations could be simplified, since only diagonals would have to be checked.

***Genetic Algorithm***

The Genetic Algorithm aims to simulate evolution in nature by creating a population of states in a given problem, mating different states together using crossover, and mutating the offspring at some small probability rate. Over successive generations, the overall fitness of the population improves until, hopefully, a solution is reached.

Genetic algorithm takes significantly more time to arrive on the solution then does hill climbing , but it is able to reliably find a solution given a randomly generated population. There are several parameters that effect the performance of the algorithm. One of the biggest hurdles encountered was that the initial function used for choosing parents for the next generation was leading to exorbitantly long computations, where for n=19 the program would not converge to a solution. This was the function that determined probability of selection by : . This created a distribution which actually ended up favoring selection of finesses more close to the average then to the solution . Instead, a ranked based selection method was used where the selection probability was : . For the best rank in population N, rank = N. The worst rank is 1. Using this, the performance significantly improved. For a 19x19 board, the solution was found in 146.0 ms. 3535 generations were needed to reach the solution, and the population size was 38, initially. The mating rate was set at .7 (or 70%), and the mutation rate was set at 2%.

In order to determine the best mating rate and mutation rate, I relied on generally accepted values. Research showed a rate of around .6 to .8 to be ideal. The idea being that if the number is too high, then high fitness individuals will be modified too often and the algorithm will struggle to converge to a solution. The same goes for the mutation rate. Because of the high variability, many tests would need to be run to find the best parameters for any given board size. In testing, it was observed that a population size roughly twice the board size resulted in rather quick convergence for n<75. Above that, the time increased at an exponential rate.

***Conclusion***

Both algorithms are capable of finding solutions to the N-queens problem. The steepest hill algorithm, however suffers from the problem that it can only solve a small percentage of boards. The genetic algorithm is useful for converging onto a solution to a particular problem space, but it would not be good for say generating a series of moves to reach a solved state from an initial starting point. Genetic algorithm is an extremely interesting algorithm , and studying the problem further with it would be fascinating.