Brandon Palomino
CS 420

N-Queen Project Analysis

<u>Approach</u>

The N-Queen problem solution is simply about repeatedly checking whether all the queens on the board are in conflict with each other. I chose to represent the queen's board using a list of integers, where the index represented the column number and the value represented the row number. This also serves to minimize representing the whole two-dimensional board itself. After setting up this representation of each state, I approached solving the problem with two different algorithms which both pertain to local search topic I've been studying.

The first is the steepest ascent hill climbing algorithm which basically starts with an initial state, maps out all the heuristic (conflict) values of the board, then chooses one of the best possible states to be in next, and repeats until it cannot go any further. The problem with this algorithm is it will stuck in local maxima most of the time and plateau there, preventing it from reaching the goal state of 0 conflicts. My implementation is the straight forward method as described above. Using functions, lists, tuples, and a dictionary to help generate my appropriate successor state at each step in the algorithm.

The continue the second algorithm I tested and used is called Simulated Annealing. This one is different but still straight forward with its implementation. Modeled after the cooling of glass, it uses a temperature and annealing values to measure the gradual "cooling" of conflicts and approach to the solution. It starts off with the initial board state, represented as a list as described earlier, then makes a random move in the board moving one of the queens to a new location, followed by checking if the move should accepted. By that I mean if the heuristic was reduced or if not assess by using the probability of acceptance formula described by the algorithm to determine if it is risky or not. This process repeats itself until a heuristic is accepted, then the temperature is reduced by the annealing rate and the process repeats until the heuristic equals 0.

<u>Analysis</u>

For the Hill Climbing algorithm it is noted prior that 14% of problems for n=8 queens. Through my testing for 100 cases at n=22 queens I successfully solved 0% of the 100 random problems generated with average elapsed time of 0.213207979202 seconds. For Simulated Annealing for 100 test cases at n = 22 queens I solved 100% of the problems with an average elapsed time of 0.447971119881 seconds. These values can be observed from the output.txt file.

<u>Findings</u>

I found that although the run time for both smaller and at n=22 queen cases was longer for simulated annealing. I could observe and predict that at larger sizes simulated annealing will win out overall in terms of elapsed time and being able to solve all the problems thrown at it. While I was also surprised that the hill climbing algorithm had an amazing speed to approach a very close solution most of the time and would win out if it came to small queen sizes.