CS 165A MP1 Report Sarah Darwiche – 6031843

1) Explanation of code architecture: My code is organized into a search class and a move class. The search class contains various variables for a unique N-queen problem such as initial state, best state, and queen count. It also contains functions for that unique problem to operate on such as count violations, hill climb search, read from standard input, and write to standard output. I also have a satisfies\_upper\_bound(), which makes sure the solution agrees with the passed in value of k. Main controls timing and initializing and starting the search. The program is set to time out after thirty minutes in hill\_climb\_with\_restart().

Additionally, I have some debugging functions, which I do not call in my main, but I have provided as an extra tool. These functions include print\_grid(), which prints the visual representation of the grid at the current state. Generate\_random\_problem(int queen\_count) creates a random problem of size queen\_count that can be used to test the algorithm.

2) Heuristic I chose and why: I chose random restart hill-climb search because I believed from the textbook that it would be the most efficient in solving the N-queen problem. (The example in the text for hill-climb illustrated the N-queen problem.) I chose this based n the fact that in the text it says that it is trivially complete with probability approaching one, because it will eventually generate the goal state as the initial state. It even stated that for three million queens the approach could find solutions in under a minute!

3) Challenges I faced and how I solved them: My first attempt at hill climb resulted in an infinite loop because when the algorithm reached two states with the same number of violations it kept changing from one to the other and never finishing. It was getting stuck during side moves. Another technique I employed to raise my chances of finding a solution is I implemented random restart hill climb so that if a solution wasn’t found or k was not satisfied, the problem would restart with a different initial state with the same number of queens.

I also faced a timing complication at one point because I was placing my check at the wrong point in hill\_climb\_with\_restart() instead of putting the check in hill\_climb() – where most of the computation happens. The problem was that it would continue to evaluate even after the timeout was up. After changing the location, this was no longer an issue.

I also realized that my algorithm was executing with quadratic time because of a double loop in count\_violations() so I changed that to only execute the entire double loop the first time and minimize the computations every time a queen was moved afterwards. This helped change the fact that the time the algorithm was taking was growing quadratically. Although I couldn’t change it at every instance, it still helped dramatically changing a few instances where it was called. After the changes my algorithm performs in linear time.

4) Weaknesses in my methods: For the n-queen of size 2 and three there is no solution. Other weakness include a slow down in timing as a result in reading from standard output and writing to standard input. Another issue arises when k is small. It will restart hill climb many times trying to find a solution that satisfies the k upper bound and may time out before doing so.