Ian Brown
COMP 5600
Homework 1

Advanced Approaches

Score function:

- I defined the score function for this problem as the number of conflicting queens.
- The goal is to minimize this function f(s)
- I calculate the score by summing each diagonal, row and column. If the sum of any particular diagonal/row/column is greater than 1 then there must be at least two queens there.
- My score function does not include the current queen in its score total. For instance, if there are two queens on a horizontal, it will return one because only one queen is causing a conflict. Likewise, if there is only one queen on a horizontal it will return zero because there are no other queens conflicting with it.

Neighbors:

- My queens are restricted to move vertically (up and down in their column). Therefore, each queen's neighborhood consists of all the valid spaces it can move to in its column.

Best state:

- The best state my algorithm was able to find was score = 5. Meaning 5 queens were conflicting. This is down from 25 conflicting queens in the start state.

```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]]
5
```

- My approach was hill climbing with random restarts.

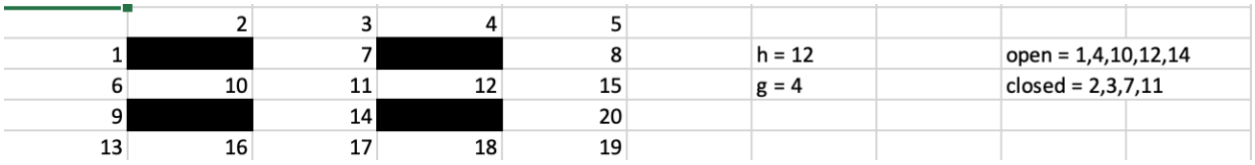
The first time I iterate through the matrix, I move each queen to a random position in its neighborhood. As stated earlier, my queens are restricted to moving vertically and the neighborhood is defined as all the possible locations it can move to in its column. I do this because the starting state can be considered a local optimum. This prevents the algorithm from staying still in the start state.

From there, for each queen that is not safe from being attacked horizontally or diagonally, I iterate through its neighbors to find an open position which is safe and move to it. After moving the queen, I compare the new state to the previous state. If the score of the new state is better, I keep it and start expanding from there. Otherwise, I throw it away. I run this algorithm "max_iteration" number of times. The higher the number of iterations, the longer it will take to halt, but the closer the result will be to zero. I keep the highest scoring board out of all the iterations.

My program prints the original board state and its score followed by the best state found and its score.

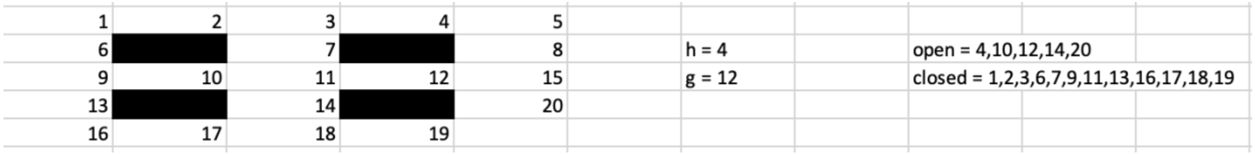
A* Algorithm:

- A* took 17 steps to reach the goal state
- I expanded 17 states
- The 5th State



	2	3	4	5					
1	■	7	■	8		h = 12		open = 1,4,10,12,14	
6	10	11	12	15		g = 4		closed = 2,3,7,11	
9	■	14	■	20					
13	16	17	18	19					

- The 5th last state



1	2	3	4	5					
6	■	7	■	8		h = 4		open = 4,10,12,14,20	
9	10	11	12	15		g = 12		closed = 1,2,3,6,7,9,11,13,16,17,18,19	
13	■	14	■	20					
16	17	18	19						

Please see my full trace attached to this submission as "A* Trace".

A* Proof (contradiction):

Assume that A* is not optimal given the heuristic function h(s) is admissible. If A* is not optimal, path p will not have optimal substructure. The definition of admissible is h() will produce estimations *h(s) ≤ h*(s)* where s is the given state and h*(s) is the actual cost of the move. Due to the optimistic nature of an admissible heuristic, A* will never skip over a shorter path while expanding nodes. Thus, path *p* must have optimal substructure. If the path *p* has optimal substructure, then the entire path must be optimal. Therefore, A* is optimal.

Dijkstra's Vs. UCS:
Dijkstra's algorithm is a variant of Uniform Cost Search. In Dijkstra's, there is no explicit goal state and execution will continue until all nodes have been removed from the priority queue. It will return the shortest path to all nodes in the graph. Not just the shortest path to the goal node.

Open question:
After trying different heuristics, I cannot find one that performs more optimally than Manhattan distance for this particular problem. I tried using the total number of misplaced states as my heuristic instead and ended up with the same goal state. I expanded more states along the way because this approach is unable to tell that moving 11 is the best starting move. I believe the Manhattan distance is the best heuristic for this problem.