**Heavy NQueens**

**Question 1:**

The astar algorithm can solve a 4 x 4 board in under one second, but a 5 x 5 board can take a couple of minutes to solve. In both cases, the depth of the solution matters, and for astar, the cost of the solution matters- higher cost solutions will take longer to find if there are many lower cost solutions to explore. In other words, board complexity depends on a number of factors, including size, depth to solution and cost of the solution. Thus, we decided to do further analysis on the 4x4 and 5x5 board for astar.

Greedy search can solve a more complex board compared to astar and we noticed more of a difference between h1 and h2 in terms of the algorithms ability to successfully come up with a correct solution. Heuristic h1 could solve boards up to 8x8, and h2 could solve boards of 9x9 and beyond- we stopped at 9x9 since h1 could not solve this board greater than 50% of 10 attempts. While greedy search can solve more complex boards, in a shorter period of time, the solutions that greedy search presents are usually far more costly compared to the astar solution- astar finds the solution with the lowest cost.

**Question 2:**

We chose to use both 4x4 and 5x5 boards for the analysis of the two algorithms. Further, we randomly generated each board analyzed and ran each algorithm-heuristic combination on that board and corresponding queens. The decision to randomly create boards was made from the observation that board complexity is related to the 3 factors of size, cost to solution and depth of solution, and it is unclear exactly how these three factors influence board complexity when considered in tandem.

The branching factor of this version of the heavy\_n\_queens problem is (n \* (n-1)). The maximum number of states to explore for astar is equal to:

N = b\* + (b\*)2 + ... + (b\*)d

Where b is the effective branching factor, N is the number of nodes expanded and d is the depth of the solution. Solving for b is computationally complex, but according to Dr. Gabriel Ferrer, the effective branching factor can be approximately estimated with a provable error bound using the following equation:

Effective branching factor = N(1/d)

We used this formula to determine the effective branching factor for each individual run of an algorithm-heuristic combo. We then took all the effective branching factor values for each algorithm-heuristic combination and computed the mean across all values for that combo group.

Again the equation used to determine the branching factor was as follows:

Branching factor = (n -1) \* n

Where n is the number of queens. For a 4x4 board, the branching factor is 12, and for a 5x5 board, the branching factor is 20. Both algorithms with either heuristic will generate these number of successors for the respective board.

Next, we are going to present data from the analysis we ran on each of the algorithm-heuristic combinations, and at the same time we will answer questions 3 and 4. We analyzed 20 boards for each combination- 14 of which we kept after filtering (Removing already correct boards), the raw data is available in the submitted package. Please note that we selected the number of sideway moves for hill climbing based the average number of times the algorithm could complete a puzzle within 10 seconds. Our experiment was as follows: start at some random number of sideways moves (5) and a board size of 5 (best on astar analysis). Next, increase the board size until the number of sideways moves can no longer solve the board within 10 seconds. I the algorithm can’t solve the board, increase the number of sideways moves. There was a 1-to-1 pattern between increasing board complexity and increasing sideways moves until about a board size of nine- at this point it took up to 15 sideways moves to consistently solve the puzzle. Thus, we decided to select 9 sideways moves, as this would guarantee solving most puzzles of size 8x8 and under, and larger boards would benefit from other algorithm optimizations. The raw data for this analysis is contained in a file called sideways.txt.

**Questions 3 and 4:**

Consider the astar algorithm using heuristic 1 and heuristic 2.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Number of queens | Runtime | Cost | Heuristic\_score | Depth | Effective\_Branching | Nodes\_Processed |
| count | 14 | 14 | 14 | 14 | 14 | 14 | 14 |
| mean | 4.357142857 | 75.38918575 | 52 | 0 | 2.214285714 | 26.90506624 | 8233.214286 |
| std | 0.4972451581 | 279.1411736 | 60.73523879 | 0 | 1.577659973 | 88.90349031 | 30068.4904 |
| min | 4 | 0.005715370178 | 1 | 0 | 1 | 1.414213562 | 2 |
| 25% | 4 | 0.02429062128 | 8.5 | 0 | 1 | 2.227002062 | 5.75 |
| 50% | 4 | 0.08050227165 | 23.5 | 0 | 1.5 | 2.805913143 | 20.5 |
| 75% | 5 | 0.9489191175 | 81 | 0 | 3.5 | 3.165745804 | 318 |
| max | 5 | 1045.224869 | 217 | 0 | 5 | 335.702249 | 112696 |

Table 1. Astar algorithm with heuristic 1.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Number of queens | Runtime | Cost | Heuristic\_score | Depth | Effective\_Branching | Nodes\_Processed |
| count | 14 | 14 | 14 | 14 | 14 | 14 | 14 |
| mean | 4.357142857 | 77.39721414 | 52 | 0 | 2.214285714 | 26.90506624 | 8233.214286 |
| std | 0.4972451581 | 286.6114427 | 60.73523879 | 0 | 1.577659973 | 88.90349031 | 30068.4904 |
| min | 4 | 0.006340742111 | 1 | 0 | 1 | 1.414213562 | 2 |
| 25% | 4 | 0.02500534058 | 8.5 | 0 | 1 | 2.227002062 | 5.75 |
| 50% | 4 | 0.08604693413 | 23.5 | 0 | 1.5 | 2.805913143 | 20.5 |
| 75% | 5 | 0.9607008696 | 81 | 0 | 3.5 | 3.165745804 | 318 |
| max | 5 | 1073.187645 | 217 | 0 | 5 | 335.702249 | 112696 |

Table 2. Astar using heuristic 2.

Both table 1 and 2 look very familiar, and in fact they have the exact same statistics for the number of nodes processed (expanded) and the effective branching strategy. This makes sense considering the explanation provided in question in which we described the relationship between the heuristic functions and the cost function in finding a solution with astar. The runtimes for these two algorithms are in fact different, although this is not believed to be for any significantly interesting reason. Instead, we will focus on the over inflated mean values for the effective branching factor, the number of nodes processed and the runtime, and why we can see such low values for the average depth.

Astar combines properties of breadth first and depth first search. Astar becomes more like breadth first search with increasing depth of the solution. As the depth to the solution increased, the number of nodes expanded increases exponentially. Thus the average depth may be 2.2, which has a solution space of about 400 nodes, but given a board of say depth 4, the solution space jumps to over 168,420 nodes ()- hence the inflated average values.

Due to the exponential nature of the astar algorithm, we do think our method for computing an effective branching factor is closer to an actual value than if we had computed on a set of boards that were constrained in complexity.

In conclusion, the astar algorithm performed similarly with each of the heuristics. The analysis of the algorithm clearly demonstrates the exponential complexity of the depth first component of this algorithm.

Now we consider Greedy Hill climbing with reastarts.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Number of queens | Runtime | Cost | Heuristic\_score | Depth | Effective\_Branching | Nodes\_Processed |
| count | 14 | 14 | 14 | 14 | 14 | 14 | 14 |
| mean | 4.357142857 | 0.01676234177 | 152.8571429 | 0 | 3.642857143 | 1.165522915 | 4.285714286 |
| std | 0.4972451581 | 0.01770822648 | 197.7877095 | 0 | 3.152706843 | 0.1498388299 | 4.952521835 |
| min | 4 | 0.002733707428 | 12 | 0 | 1 | 1 | 1 |
| 25% | 4 | 0.003785192966 | 36 | 0 | 1 | 1 | 1 |
| 50% | 4 | 0.008401036263 | 81 | 0 | 2.5 | 1.25992105 | 2.5 |
| 75% | 5 | 0.02104461193 | 195 | 0 | 6.5 | 1.299036263 | 6.5 |
| max | 5 | 0.05458641052 | 744 | 0 | 10 | 1.316074013 | 19 |

Table 3. Greedy Hill Climbing with restarts using heuristic 1.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Number of queens | Runtime | Cost | Heuristic\_score | Depth | Effective\_Branching | Nodes\_Processed |
| count | 14 | 14 | 14 | 14 | 14 | 14 | 14 |
| mean | 4.357142857 | 0.01032815661 | 75.5 | 0 | 2.214285714 | 1.194414936 | 2.5 |
| std | 0.4972451581 | 0.007493778788 | 57.58972932 | 0 | 1.311403912 | 0.1951339533 | 1.829249531 |
| min | 4 | 0.002765893936 | 7 | 0 | 1 | 1 | 1 |
| 25% | 4 | 0.003875613213 | 40.25 | 0 | 1 | 1 | 1 |
| 50% | 4 | 0.007577896118 | 64.5 | 0 | 2 | 1.25992105 | 2 |
| 75% | 5 | 0.01407772303 | 95.25 | 0 | 3 | 1.316074013 | 3 |
| max | 5 | 0.02470684052 | 217 | 0 | 5 | 1.626576562 | 7 |

Table 4. Greedy Hill Climbing with restarts using heuristic 2.

The greedy hill climb with restarts algorithm has very different characteristics compared to the astar algorithm. First, we compare heuristic one to heuristic two. There is a surprising difference between these two heuristics for this algorithm, and we saw this in the initial complexity analysis of the algorithms. Heuristic two appears to generate lower cost, solution depth and nodes processed compared to heuristic 1. It is difficult to say exactly why this is this case, but it is worth noting at this point that H2 may not be admissible. If we were to collect more data on heuristic 2 we may see that these statistics are not guaranteed and could even result in a greater chance of no solution being found. However, at this scale, this is not the case- heuristic 2 appears to perform better than heuristic one.

When comparing astar to hill climbing, the runtime is the biggest difference. Hill climbing takes under one second on average while astar takes over 1 minute. Heuristic one with hill climbing finds a solution of greater depth on average compared to the other algorithm-heuristic combinations. This highlights the ability of hill climb to quickly search for deep solutions, but it does this at the expense of considering the cost of the solution. Thus, the average cost of the hill climbing algorithm is greater than the astar algorithm. Both algorithms have trade-offs. Again, we point out that heuristic two with hill climbing, at the present board complexity, seems to offer similar solutions compared to astar, but at a much faster speed. We are not exactly sure why this is the case.

In conclusion, in general and on average, hill climbing finds a solution at a faster speed but higher cost compared to astar.

**Question 5:**

Counter Example to H2:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Node 1 |  |  |  |  |  |  |
|  |  |  | 1 |  | 2 | H2 = 4 |
|  |  |  |  |  |  | attacking pairs = 3 |
|  |  |  |  |  |  |  |
|  |  |  |  | 9 |  |  |
|  | 1 | 1 |  |  |  |  |
| Node 2 |  |  |  |  |  |  |
|  |  |  | 1 |  |  | H2 = 5 |
|  |  |  |  |  |  | attacking pairs = 2 |
|  |  |  |  |  |  |  |
|  |  |  |  | 9 | 2 |  |
|  | 1 | 1 |  |  |  |  |

Moving from node 1 to node 2 would result in fewer pairs of attacking queens but the heuristic value is greater for node 2 than node 1. Thus, heuristic 2 is not admissible.