Adam Mischke
Dr. Phillips
CSCI 4350 Open Lab 1
Tue. September 25, 2017

Solving Problems by Search

Problem:

The 8-puzzle problem (often called the N-puzzle problem), is a reputable and relatively easy problem that most people have seen before. The puzzle consists of a 3x3 game board where each tile is enumerated in some (mostly) random order with numbers starting from 1 and ending in 8. There is one tile in which no number (or 0) lies and that is the only piece on the board that is allowed to swap with any other adjacent piece.

Ex. 1:

0	1	2
3	4	5
6	7	8

move right

1	0	2
3	4	5
6	7	8

move down

3	1	2
0	4	5
6	7	8

Goal:

The goal of the puzzle is to move the 0 position such that the all of the numbers are in some correct position. In our example, the first position that we used will be the goal state.

Solution:

There are many ways to go about solving this puzzle, for this project we used the A* search algorithm that consisted of these basic steps:

1. Create a list of states that have already been visited.
2. Create a priority queue, where states that are closer to the goal are put before others and put the initial state in there
3. While the queue is not empty:
    a. Pop the best cost off of the priority queue
    b. Add the state to the visited list
    c. If that state is not the goal:
        i. If we haven't visited that state yet:
            1. add the possible moves (move left, move right, ..) to the queue in terms of ascending cost
    d. If that state is the goal:
        i. GOALL!!! (finish)

h(2) = Manhattan Distance			
Nodes	**Visited**	**Depth**	**Branching factor**
Minimum: 7 Median: 612.5 Mean: 1211.39 Maximum: 9668 Standard Deviation: 1660.622876483399	Minimum: 3 Median: 373.0 Mean: 762.46 Maximum: 6238 Standard Deviation: 1063.610825631255	Minimum: 2 Median: 18.0 Mean: 18.26 Maximum: 26 Standard Deviation: 4.7552497305609505	Minimum: 1.26124 Median: 1.39742 Mean: 1.4104358 Maximum: 2.64575 Standard Deviation: 0.13398869389004434
h(1) = Tiles Displaced			
Nodes	**Visited**	**Depth**	**Branching factor**

Minimum: 7 Median: 3585.0 Mean: 12441.0 Maximum: 92928 Standard Deviation: 19905.46886662055	Minimum: 3 Median: 2259.0 Mean: 8209.33 Maximum: 64344 Standard Deviation: 13564.377390838848	Minimum: 2 Median: 18.0 Mean: 18.26 Maximum: 26 Standard Deviation: 4.7552497305609505	Minimum: 1.46281 Median: 1.54893 Mean: 1.5596145000000001 Maximum: 2.64575 Standard Deviation: 0.11126527146756084
h(0) = 0			
Nodes	**Visited**	**Depth**	**Branching factor**
Minimum: 15 Median: 59928.5 Mean: 114951.2 Maximum: 460464 Standard Deviation: 123975.77236299032	Minimum: 8 Median: 40087.5 Mean: 86798.01 Maximum: 405545 Standard Deviation: 102520.91453557124	Minimum: 2 Median: 18.0 Mean: 18.26 Maximum: 26 Standard Deviation: 4.7552497305609505	Minimum: 1.64486 Median: 1.80523 Mean: 1.8352256999999994 Maximum: 3.87298 Standard Deviation: 0.23624108031100344
h(3) = My own, well, actually this is Euclids. Euclidean Distance			
Nodes	**Visited**	**Depth**	**Branching factor**
Minimum: 7 Median: 1349.0 Mean: 3821.16 Maximum: 29381 Standard Deviation: 5820.17015510715	Minimum: 3 Median: 846.0 Mean: 2445.7 Maximum: 19297 Standard Deviation: 3792.408866406681	Minimum: 2 Median: 18.0 Mean: 18.26 Maximum: 26 Standard Deviation: 4.7552497305609505	Minimum: 1.36811 Median: 1.47089 Mean: 1.4821485999999995 Maximum: 2.64575 Standard Deviation: 0.122981190944144

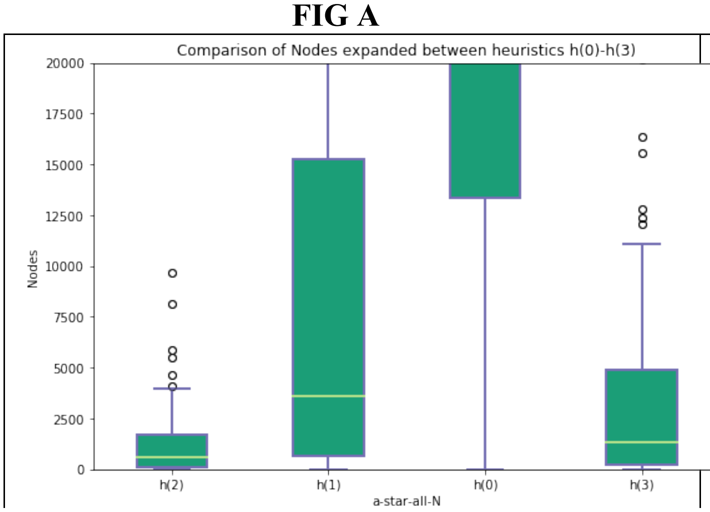
Heuristics:

Coming up with heuristics was difficult, but since we used the A* search algorithm to quickly search, we wanted solution path that exhibited four basic properties: : **complete**, meaning that we always find the goal, **consistent**, meaning that when we chose paths, we always chose the ones that were the closest to the goal, **admissible**, meaning that we never overestimated our guesses to states, and **optimal**, meaning that when we found to entire series of correct moves to get from the begin to end state, there were no quicker or shorter ways to get there. After using a machine to run each heuristic 100 times each with randomly seeded values, we come up some interesting data points.

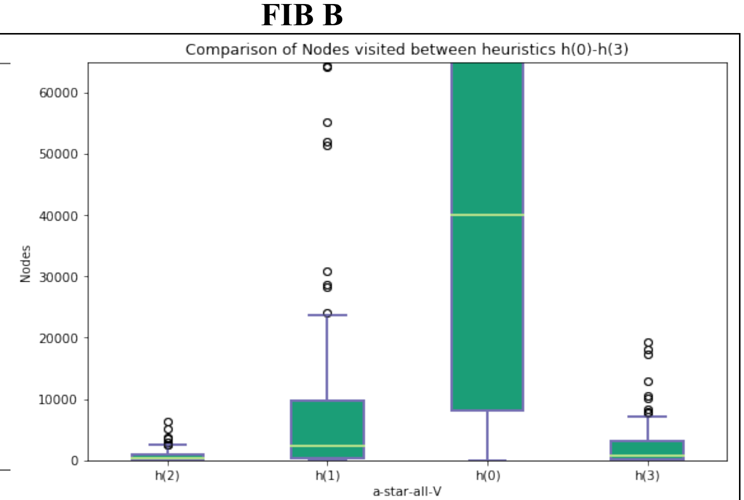
Using Jupyter to get statistics, we can see that the heuristics are sorted by dominance from top (best) to down (worst) **minus h(3) which is bottom one**. Determined by the Nodes expanded, Visited Nodes, and Depth averages, the heuristics that dominated had both expanded and visited less nodes and went less deep.

sAnalysis:

FIG A



FIB B



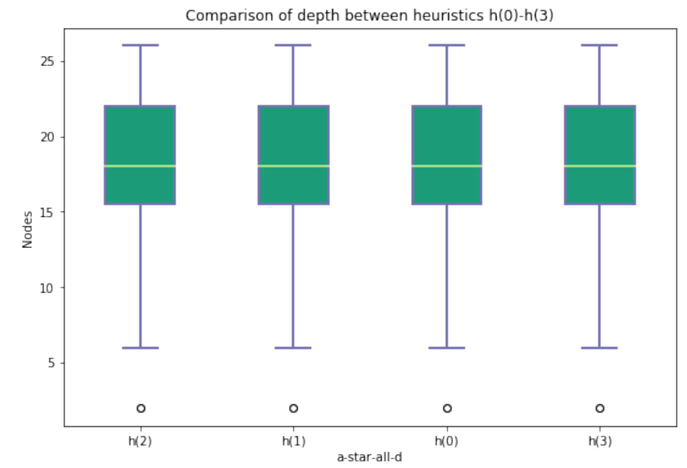


FIG C

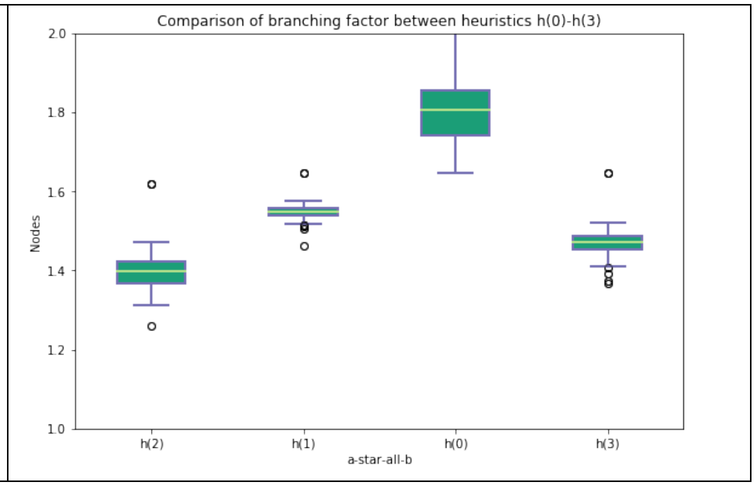


FIG D

FIG A & B-

Shows the massive difference between the heuristics. h(2) uses some factor of 90% less nodes than h(1) which uses 90% less than h(0). h(3) actually faired somewhere between h(2) and h(1).

FIG C-

This is exactly what we wanted. All of our heuristics are optimal, meaning they each all solved goals at the earliest depth. Another neat observation is that the maximum depth for this problem is **26**, while the minimum is **2**.

FIB D-

The miniscule difference of the branching factor is shown brilliantly, the more dominate the heuristic will result in a smaller branching factor.

Conclusion:

All the heuristics, h(0) = 0, h(1) = Tiles displaced, h(2) = Manhattan Distance, h(3) = Euclidean Distance exhibit the 3 properties of being **complete**, **consistent**, **admissible**, and **optimal**. In terms of domination, the winners in descending order: h(2), h(3), h(1), h(0). I think h(3), would have fared better had the board also been able to move in diagonal (straight-line) moves.

Implementation details: I used a priority queue as a frontier, a map for parent nodes, and a set for the visited/closed list. I believe I could speed my implementation if I used a single pointer to an object State, but I would be sacrificing speed for memory. This project really allowed me to visually see and mentally understand the constraints of the problem and scope (especially when picking data structures and memory management). Because of that, I have a better process on how I think about managing memory vs. speed. Another very important take-away is the understanding of what a clever heuristic can achieve to a NP-Hard Problem.

Citations:

https://heuristicswiki.wikispaces.com/N+-+Puzzle , for awesome information and ideas