Assignment 1 CS170: Introduction to Artificial Intelligence, Dr. Eamonn Keogh

Cruz Ramirez
SID 862175850
Email crami119@ucr.edu
Date Nov-1-22

In completing this assignment I consulted:

- https://stackoverflow.com/questions/1380463/sorting-a-vector-of-custom-objects
- https://cplusplus.com/reference/vector/vector/
- https://cplusplus.com/reference/queue/queue/
- https://en.wikipedia.org/wiki/New_and_delete_(C%2B%2B)
- https://www.dropbox.com/sh/rltooq0t3khobuj/AAA3MYkZc8gb1RLa3tNSnsrga?dl=0 Dr. Eamonn Keogh Lecture Materials
- My colleague and close friend Karan Bhogal

All important code is original. Unimportant sorting and helper functions are

- C standard library vector containers as well as functions
- C standard library queue containers as well as functions
- C standard library sort function

Outline of this report:

CS 170: Assignment 1: The eight-puzzle

Cruz Ramirez, SID 862175850 Nov-1-2022

Introduction:

This report details the application and results found of search algorithms used in solving the eight-puzzle during Dr. Eamonn Keogh's Introduction to AI class at the University of California Riverside during the fall quarter of 2022. The eight-puzzle is a simple sliding puzzle that has room for nine identically sized tiles on a board, but only eight tiles are used so that there is room for one of the tiles to slide in a cardinal direction at a time. Typically, the puzzle pieces have a kind of picture or number represented on their face so that there is an overall image or order that can be seen when all tiles have been moved to the right place on the board, in this specific case the numbers should be ordered from one to eight with the blank spot in the ninth position. Additionally it is important to note that we view solving the puzzle by focusing on the blank, in one move we can either move the blank space up, down, left, or right, (depending on where the blank is physically on the board) until we reach the desired goal configuration of the tiles, with the configuration of the tiles after each movement being referred to as a node. In order to solve this puzzle, the three search algorithms of Uniform Cost Search, A* with a Misplaced tile heuristic, and A* with a Manhattan Distance heuristic will be used and compared with each other.

Uniform Cost Search:

Uniform Cost search is a search algorithm that utilizes the cost or distance of a node in a search tree to its root node (cost/distance traditionally represented by g(n)) in order to make decisions on which nodes to expand when adding their children to the search tree. In our specific case for the eight-puzzle, the cost from one node to one of its child nodes is exactly one, which causes our Uniform Cost Search to act identical to the Breadth First Search blind algorithm, which expands all nodes on the same depth before exploring the child nodes. This algorithm is a brute force approach which will check each depth node by node from the leftmost node to the right in order to find the goal state of the puzzle.

A* with Misplaced Tile Heuristic:

The A* algorithm utilizes the cost to get to a node or g(n) as well as some other value known as a heuristic (traditionally represented by h(n)), which is a piece of information that can be used to gauge how close the current node is to the goal state. The A* algorithm adds the cost to get to a node (g(n)) with a heuristic value (h(n)) in order to get a value f(n), and the node with the smallest f(n) value is considered closer to the goal state and will be expanded by the algorithm in order to reach the goal state more optimally. For the eight-puzzle, the h(n) value is calculated by counting the number of tiles that are not in the correct goal position, so that the lower the h(n), the closer the tile configuration is to the goal state.

A* with Manhattan Distance Heuristic:

This version of the A* algorithm is very similar to the last one, but instead of counting the amount of misplaced tiles, it calculates all the number of spaces in the cardinal directions that the tiles are away from their intended locations and adds them altogether to get a larger and more precise gauge for the closeness a given node is to the goal configuration. Since this algorithm utilizes more information, its heuristic is stronger and allows the puzzle to be solved more optimally than with the misplaced tile heuristic.

Comparison of Algorithms in Solving eight-puzzles:

Through running each of the algorithms and collecting data on all three, displayed in figures 1,2, and 3 below, it is very apparent that if an algorithm is provided better quality heuristics on how close a node is to the goal state, it has a direct correlation to the optimality of that algorithm, and the differences between each algorithm exponentially increases as the depth of the solution increases. In comparing data from figure 1, Uniform search takes approximately 552 seconds to run to completion, where as Misplaced Tile A* only takes 10 seconds, and Manhattan A* takes even less at a miniscule 2.3 seconds. Additionally, the trend of the lines shown in figure 4 highlight how the runtime exponentially increases as the depth of the puzzle increases. Figures 5 and 6 also point out that search algorithms with better heuristics also use less resources than their counterparts, with a smaller amount of nodes expanded (which uses less cpu cycles) and smaller maximum queue sizes (uses less memory).

Figure 1: Chart displays running time of each algorithm at each puzzle depth in seconds, uniform cost time at depth 24 omitted as it takes too long to run to completion.

Depth	Uniform	Misplaced	Manhattan
2	0.1	0.1	0.1
4	0.3	0.2	0.1
8	1.5	0.3	0.2
12	7.1	1	0.6
16	57.2	3	1.8
20	552	10	2.3
24		106	17

Figure 2: Chart displays number of nodes expanded by each algorithm at each puzzle depth, uniform cost expansion number at depth 24 omitted as it takes too long to run to completion.

Depth	Uniform	Misplaced	Manhattan
2	6	2	2

4	32	4	6
8	310	19	18
12	2327	131	30
16	13709	683	280
20	54665	3285	482
24		18316	4777

Figure 3: Chart displays the maximum queue size recorded by each algorithm at each puzzle depth, uniform cost max queue size at depth 24 omitted as it takes too long to run to completion.

Depth	Uniform	Misplaced	Manhattan
2	8	3	3
4	28	6	7
8	190	16	15
12	1280	83	24
16	6584	400	185
20	17289	1813	304
24	0	8592	2668

Figure 4: Time vs Depth of Puzzle graph, blue line represents Uniform cost search, red line represents Misplaced tile A*, and yellow line represents Manhattan Distance A*. Uniform cost search depth 24 point displays time of zero but actually takes too long to run to completion.

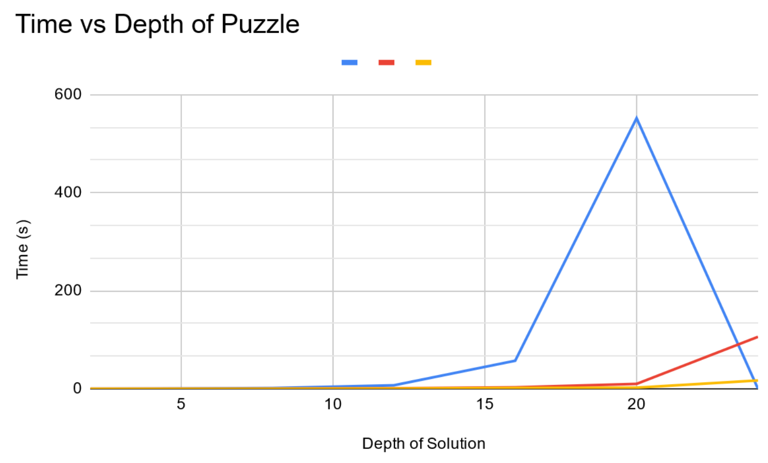


Figure 5: (next page) Node expansions vs depth of puzzle bar graph, blue bar represents Uniform cost search, red bar represents Misplaced tile A*, and yellow bar represents Manhattan Distance A*. Uniform cost search depth 24 bar omitted but actually takes too long to run to completion.

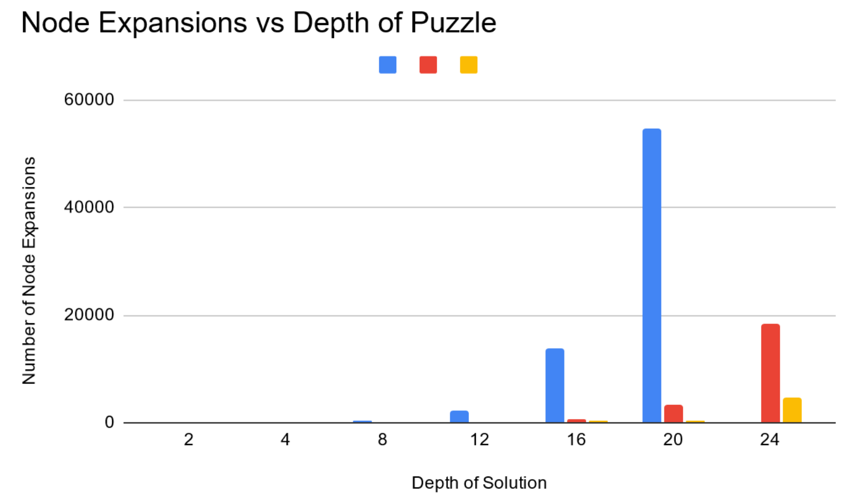
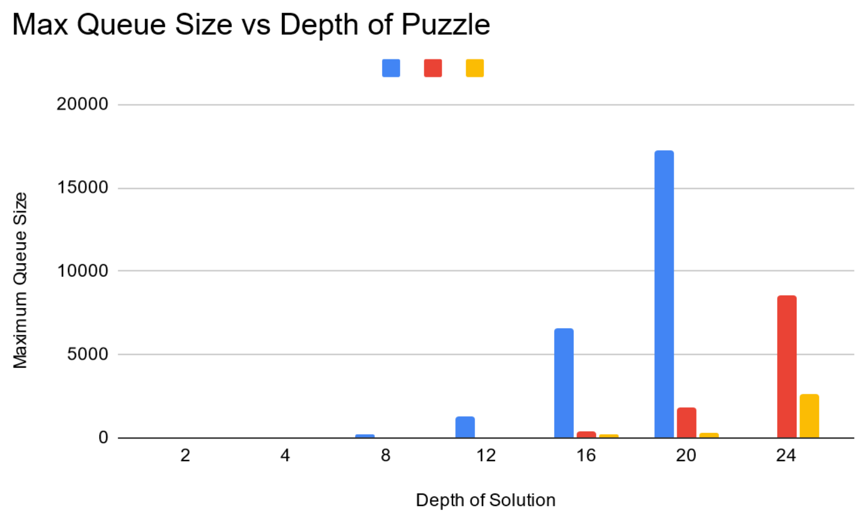


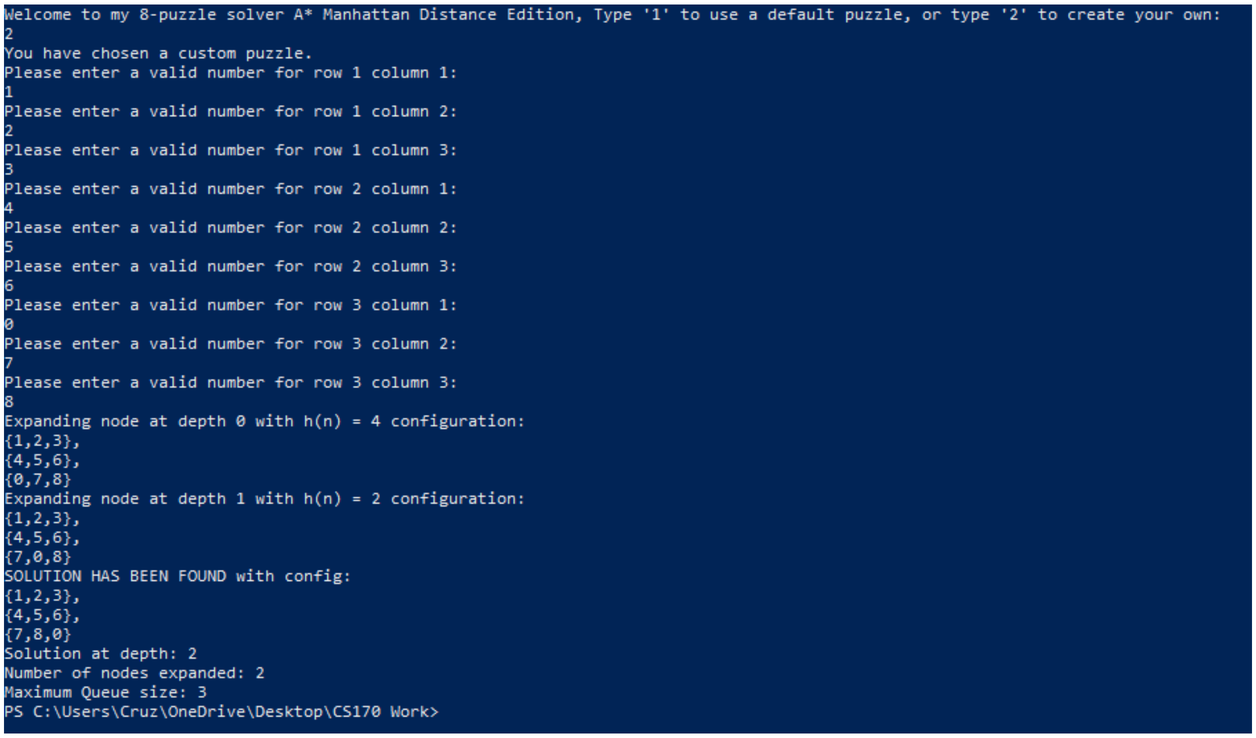
Figure 6: Maximum Queue size vs depth of puzzle bar graph, blue bar represents Uniform cost search, red bar represents Misplaced tile A*, and yellow bar represents Manhattan Distance A*. Uniform cost search depth 24 bar omitted but actually takes too long to run to completion.



<u>Conclusion:</u>

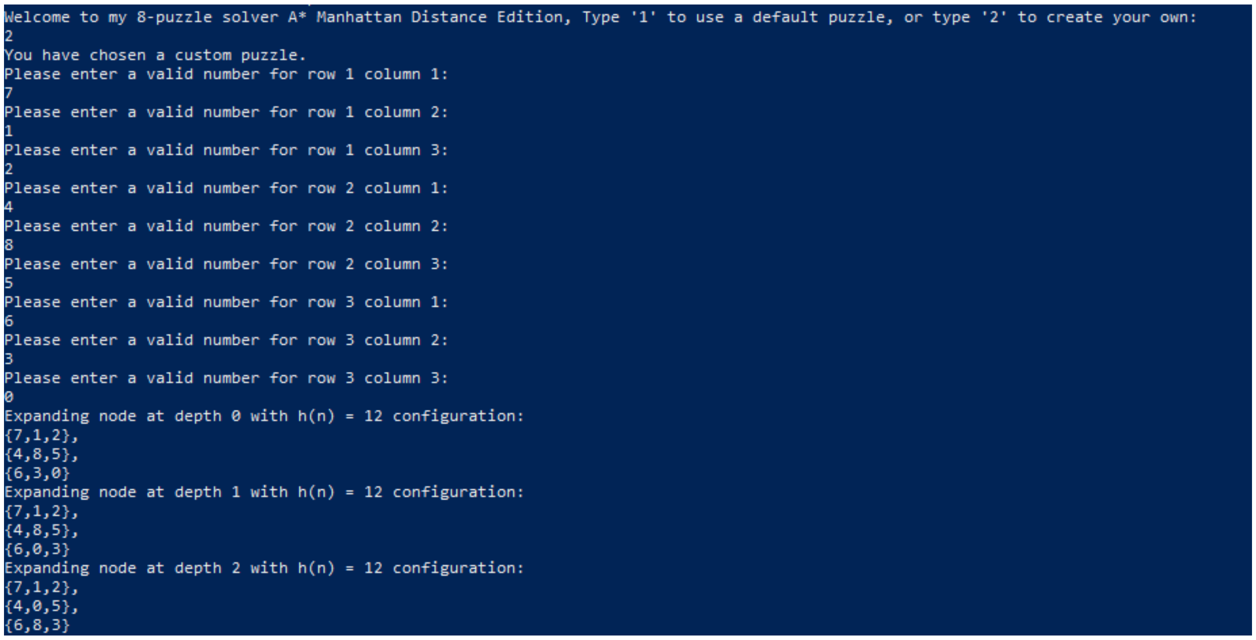
In conclusion it can be deduced from the data found in this project that search algorithms with better heuristics can find optimal solutions to problems or puzzles much quicker and with less resources than those with little or no heuristics. As shown in figures 1,2, and 3 above, Uniform Cost search performed the worst as it is not given a heuristic value, A* with Misplaced tile Heuristic performed much better with its weak heuristic, and A* with Manhattan distance heuristic performed the best as it had the most thorough heuristic which took more information into account than A* with Misplaced tile. This is precisely the reason as to why researching heuristics is the future to search algorithms as A* is already the most optimal search algorithm and can only be improved further with usage of better heuristics.

The following is a traceback of an easy puzzle of depth 2:



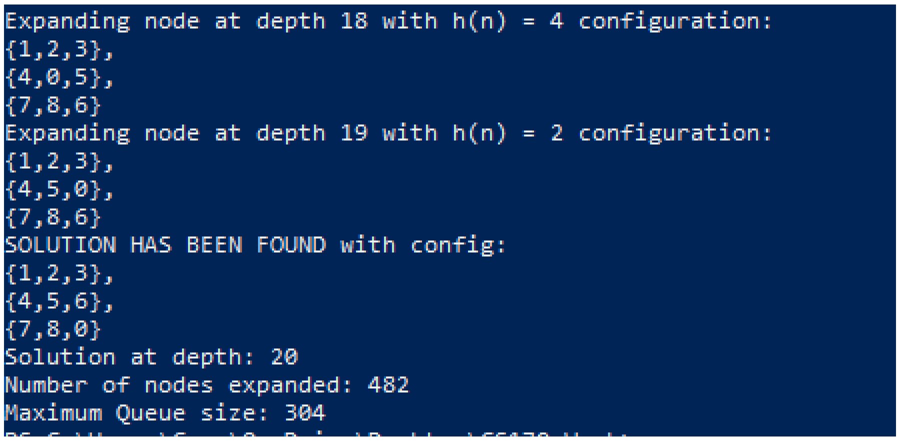
```
Welcome to my 8-puzzle solver A* Manhattan Distance Edition, Type '1' to use a default puzzle, or type '2' to create your own:
2
You have chosen a custom puzzle.
Please enter a valid number for row 1 column 1:
1
Please enter a valid number for row 1 column 2:
2
Please enter a valid number for row 1 column 3:
3
Please enter a valid number for row 2 column 1:
4
Please enter a valid number for row 2 column 2:
5
Please enter a valid number for row 2 column 3:
6
Please enter a valid number for row 3 column 1:
0
Please enter a valid number for row 3 column 2:
7
Please enter a valid number for row 3 column 3:
8
Expanding node at depth 0 with h(n) = 4 configuration:
{1,2,3},
{4,5,6},
{0,7,8}
Expanding node at depth 1 with h(n) = 2 configuration:
{1,2,3},
{4,5,6},
{7,0,8}
SOLUTION HAS BEEN FOUND with config:
{1,2,3},
{4,5,6},
{7,8,0}
Solution at depth: 2
Number of nodes expanded: 2
Maximum Queue size: 3
PS C:\Users\Cruz\OneDrive\Desktop\CS170 Work>
```

The following is a traceback of a hard puzzle of depth 20:



```
Welcome to my 8-puzzle solver A* Manhattan Distance Edition, Type '1' to use a default puzzle, or type '2' to create your own:
2
You have chosen a custom puzzle.
Please enter a valid number for row 1 column 1:
7
Please enter a valid number for row 1 column 2:
1
Please enter a valid number for row 1 column 3:
2
Please enter a valid number for row 2 column 1:
4
Please enter a valid number for row 2 column 2:
8
Please enter a valid number for row 2 column 3:
5
Please enter a valid number for row 3 column 1:
6
Please enter a valid number for row 3 column 2:
3
Please enter a valid number for row 3 column 3:
0
Expanding node at depth 0 with h(n) = 12 configuration:
{7,1,2},
{4,8,5},
{6,3,0}
Expanding node at depth 1 with h(n) = 12 configuration:
{7,1,2},
{4,8,5},
{6,0,3}
Expanding node at depth 2 with h(n) = 12 configuration:
{7,1,2},
{4,0,5},
{6,8,3}
```

Middle pages of traceback omitted for brevity



```
Expanding node at depth 18 with h(n) = 4 configuration:
{1,2,3},
{4,0,5},
{7,8,6}
Expanding node at depth 19 with h(n) = 2 configuration:
{1,2,3},
{4,5,0},
{7,8,6}
SOLUTION HAS BEEN FOUND with config:
{1,2,3},
{4,5,6},
{7,8,0}
Solution at depth: 20
Number of nodes expanded: 482
Maximum Queue size: 304
```

##URL to GitHub with project code: https://github.com/Qrooz/CS170Project1

- Version history is not representative of actual, real version history saved through uploads to personal google drive