CS205 Artificial Intelligence Project 1
The Eight Puzzle

Name : Niveditha Anand
SID : 862392888
UCR ID : nanan010@ucr.edu

"*A sliding puzzle, sliding block puzzle, or sliding tile puzzle is a combination puzzle that challenges a player to slide (frequently flat) pieces along certain routes (usually on a board) to establish a certain end-configuration.*"[1]

The eight puzzle is a sliding puzzle with 3 x 3 with 1 to 8 numbers and a Blank. The puzzle is shuffled and the blank is moved in different directions to achieve a goal state. Figure 1 is the puzzle with a shuffled state. The ' ' is the blank[' ', 0, B]. The ' ' is the moved in the direction 'up', 'down', 'left', 'right' to achieve the goal state in Figure 2.

1	3	6
5		2
4	7	8

Figure 1 Initial State

1	2	3
4	5	6
7	8	

Figure 2 Goal State

The sliding puzzle can be of any size of n [n= 3,4,5,…] which contains numbers from 1 to (n*n)-1[1 to 8, 1 to 15, 1 to 24,….].

How to Solve the puzzle?

The puzzle can be solved using the following algorithms:
1. Uniform Cost Search
2. A* with Misplaced Tile as the heuristic
3. A* with Manhattan Distance as the heuristic

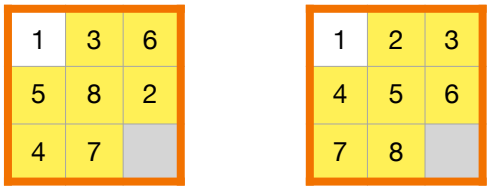
1. Uniform Cost Search

The Uniform cost search is an algorithm used to solve when exploring the nodes based on have the cumulative cost. The cheapest node is expanded first. The Data structure used is a Queue. The start node is enqueued and dequeued and expanded. The nodes' children are generated and enqueue. The child with the least path is selected for the next exploration. If the cost to explore the paths are all same(say cost is all 1), the uniform cost search is treated like a Breath First Search problem. The enqueue and child generation process is continued till the goal state is achieve at some depth. The algorithm breaks if the queue is empty before reaching the goal state. The cost of expanding a node is calculated using g(n).

2. A* with Misplaced Tile as heuristic

The A* algorithm uses f(n) to explore nodes. The f(n) is the sum of g(n) and h(n). g(n) is the cost to expand a node. h(n) is the heuristics used to reach the goal state.

h(n) can be computed be different metrics. The method used here is misplaced tile which is computed by checking for the difference in states(tiles) from the current state(tile) to the goal state. Figure 3 represents the h(n) computed using misplaced tile heuristics. The f(n) = g(n) + h(n). The g(n) is increases at the each child (depth).

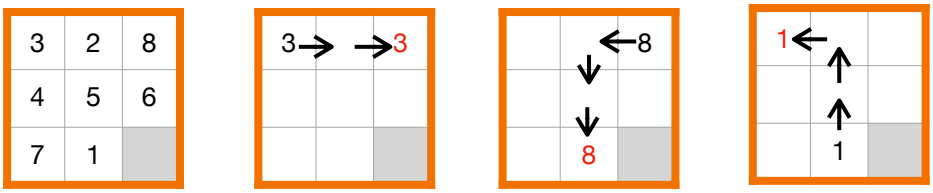


h(n) = 7 as there are 6 tiles misplaced from the goal state

Figure 3 Misplaced Tile heuristic Calculation

3. A* with Manhattan Distance as heuristic

The heuristics used in this A* algorithm is Manhattan distance. h(n) is computed by calculating the number of moves required for a particular tile to reach its position in the goal state. This can be easily computed using the absolute different in the initial and final positions of the row and column. Figure 4 computes of the h(n) using Manhattan distance. [Example taken from Lecture Slide 3 (Page 28)]



Initial State 2 places 3 places 3 places Total = 8 places

Figure 4 Manhattan Distance heuristics calculation

Problem Space:

1. **Initial state** The start node is the initial state of the problem
2. **Operators** The directions in which we can slide our blank is the operator.[Up, Down, Left, Right]
3. **Goal state** The goal node to reach.
4. **Cost of each operation** The cost of each operation, moving left or up, is the same (considered 1) in our space.

Note : The Uniform Cost search is a A* search with h(n) set to 0. The g(n) is computed and the nodes are explored based on the minimum g(n).

Comparison of the Search algorithms :

All the three algorithms were subject to different test cases provided in the project description at solution at different depth. The Figure 5 represents the nodes expanded during each algorithms.

It can be clearly viewed that the A* is the fastest as the number of the nodes explored is relatively smaller. The A* star algorithm depends on the heuristics, if the best heuristics is the used, A* search will be the fastest search.

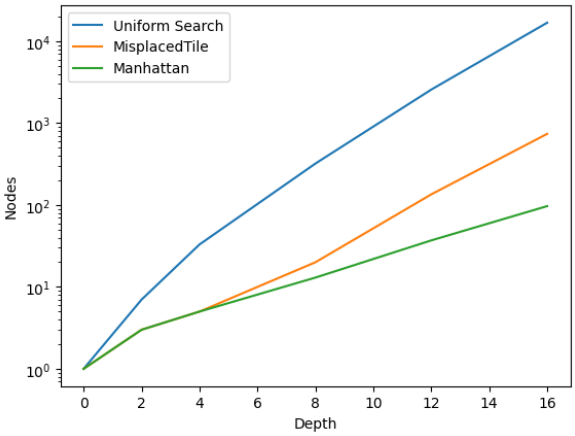


Figure 5. Comparison of the three algorithms of 8 puzzle on different depth.

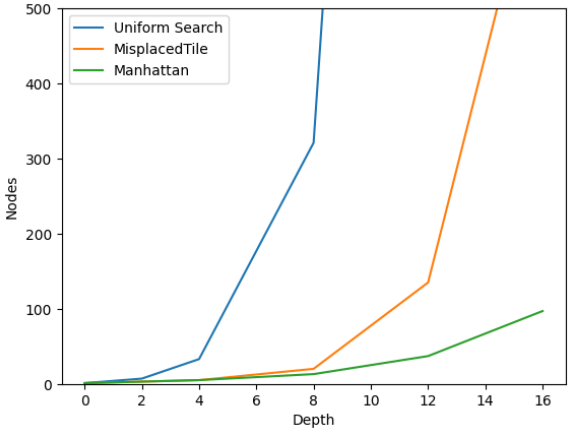


Figure 6. Detailed view of A* algorithm with different heuristics.

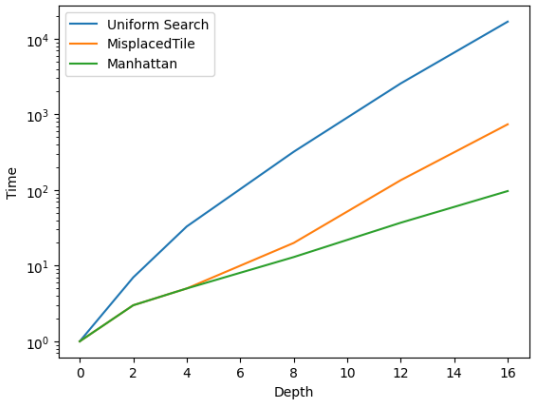


Figure 7. Time taken for each of the search algorithm to execute

In figure 5, the y axis is set to the **log** scale of the nodes expanded for a clear visualization. The Figure 6 is a detailed view of the A* algorithm to understand the importance of heuristics.

Findings:

Depth 20:

Both the A* algorithms were able to complete the search from depth 0 to 20. I was running all the algorithm sequentially, but Uniform Cost Search was able complete the search for depth 20 only when I ran the algorithm individually. The number of nodes expanded at depth 20 for the A* with misplaced tile is 3764 and the maximum queue size is 2168.

Depth 24:

The A* algorithm with Manhattan Distance was able complete the puzzle at depth 24 also. The number of node expanded were 1870 and the maximum queue size was 1098.

Conclusion

From the figure above we can see that:

- The A* algorithm with Manhattan distance performs the **best** when the solution depth increases. It is the most **optimal** search algorithm.
- The A* algorithm with misplace tile works **better** under the feasible solution depth. It is a **complete** algorithm.
- The Uniform Cost Search algorithms is a complete algorithm under a **limited** solution depth. It is not complete when is the depth crosses 16.

In conclusion, we can use an A* search for our puzzles provided that we are sure of our heuristics to be the best so that we can explore as many solutions as possible at the minimum time required.

Test Cases and Results

The nodes expanded includes the goal states as well.

TestCase 1: Depth 0:
Input : 1,2,3,4,5,6,7,8,0

1. Uniform Cost Search:
Nodes Expanded : 1
Max Queue : 0
Time Executed : 0 sec

2. A* with Misplaced Tile:
Nodes Expanded : 1
Max Queue : 0
Time Executed : 0 sec

3. A* with Manhattan Tile:
Nodes Expanded : 1
Max Queue : 0
Time Executed : 0 sec

TestCase 2: Depth 2:
Input : 1,2,3,4,5,6,0,7,8

1. Uniform Cost Search:
Nodes Expanded : 7
Max Queue : 8
Time Executed : 8.9E-5 sec

2. A* with Misplaced Tile:
Nodes Expanded : 3
Max Queue : 3
Time Executed : 3.3E-5 sec

3. A* with Manhattan Tile:
Nodes Expanded : 3
Max Queue : 3

Time Executed : 3.0E-5 sec

TestCase 3: Depth 4:
Input : 1,2,3,5,0,6,4,7,8
1. Uniform Cost Search:
Nodes Expanded : 33
Max Queue : 28
Time Executed : 0.0004 sec

2. A* with Misplaced Tile:
Nodes Expanded : 5
Max Queue : 6
Time Executed : 4.5E-5 sec

3. A* with Manhattan Tile:
Nodes Expanded : 5
Max Queue : 6
Time Executed : 4.0E-5 sec

TestCase 4: Depth 8:
Input : 1,3,6,5,0,2,4,7,8
1. Uniform Cost Search:
Nodes Expanded : 321
Max Queue : 210
Time Executed : 0.003 sec

2. A* with Misplaced Tile:
Nodes Expanded : 20
Max Queue : 16
Time Executed : 0.0002 sec

3. A* with Manhattan Tile:
Nodes Expanded : 13
Max Queue : 12
Time Executed : 0.0001 sec

Code Repo : https://github.com/nanan010/CS205AI/blob/main/AI.ipynb
Language used : Python

References:

1. [1] Sliding Puzzle: https://en.wikipedia.org/wiki/Sliding_puzzle
2. A* search : https://en.wikipedia.org/wiki/A*_search_algorithm
3. String to int : https://blog.finxter.com/python-split-string-convert-to-int/
4. 8 puzzle : https://blog.goodaudience.com/solving-8-puzzle-using-a-algorithm-7b509c331288
5. Plot : https://www.geeksforgeeks.org/plot-multiple-lines-in-matplotlib/
6. Nd array : https://stackoverflow.com/questions/34472814/use-a-any-or-a-all

7. Numpy array : https://numpy.org/doc/stable/reference/generated/numpy.arange.html
8. Numpy array : https://numpy.org/doc/stable/reference/generated/numpy.asarray.html
9. Execution Time : https://pynative.com/python-current-date-time/
10. Plot: https://www.geeksforgeeks.org/how-to-plot-logarithmic-axes-in-matplotlib/
11. Class Lecture Slide: Slides 2,3