CS205, Artificial Intelligence, Dr. Eamonn Keogh
The Eight Puzzle

Md Rayhanul Masud (SID: 862317259)
Email mmasu012@ucr.edu

May 13 2022

References The references that I have followed to complete the assignment

- The lecture contents related to the Blind Search and Heuristic Search topics

- Documentation of Python 3.10 https://docs.python.org/3

- The description of the project

- The attached sample report of the project description

Code Implementation The python code implementation of this project is solely original except the library/packages from python

- **heapq** to maintain a priority queue for the list of Eight Puzzle board states based on 3 different heuristics

- **deepcopy** to generate new board state

- **time** to calculate the execution time needed to run the code

- **matplotlib** to generate plots/figures to describe the comparisons identified among the 3 different heuristics to search for a solution of the given problem

Outline of the report

- Cover Page (Page 1)

- The description of 3 different heuristics (Page 2 to Page 6)

- The sample trace of a search (over the test cases) with the source code implementation (At the end of the report)

- The source code is also upload in https://github.com/mmasu012/Eight-Puzzle

1 Introduction

1.1 What is the Eight Puzzle

The Eight Puzzle is a puzzle where an orientation of 8 tiles numbering from 1-8 placed in a 3*3 matrix is given, and the puzzle solver needs to turn the orientation into a specific/given orientation called Goal state to solve the puzzle, by a series of sliding the tiles to the next possible empty blank position in the matrix.

This project is aimed to implement 3 known heuristics: Uniform Cost Search, A^* search with Misplaced Tiles and A^* search with Manhattan Distance to solve the puzzle. To implement them, Python (version 3) is chosen. The source code is attached with the this report. To further run the source code, the whole project is uploaded in github. The following sections of this report describe the basic understanding of the heuristics used, and provide comparative analysis based on the results found running the source code on several test cases.

2 Description of 3 heuristics used

2.1 Uniform Cost Search

When the algorithm attempts to choose which one of the current nodes in the queue is to be expanded, according to A^* search, the algorithm calculates a cost function adding the current cost $g(n)$ and the heuristic cost $h(n)$. $h(n)$ is considered to be zero for Uniform Cost Search approach.

2.2 A^* search with Misplaced Tiles

When the algorithm attempts to choose which one of the current nodes in the queue is to be expanded, according to A^* search, the algorithm calculates a cost function adding the current cost $g(n)$ and the heuristic cost $h(n)$. $h(n)$ is considered to be the total count of the misplaced tiles in the current orientation excluding the tile numbered zero as it is the blank grid of the given matrix.

Misplaced Tiles Heuristic Example

Goal State

1	2	3
4	5	6
7	8	0

Current State

1	2	3
8	**7**	6
5	**4**	0

In the given example above, the heuristic will find that the current state has $h(n) = 4$ because the four bold tiles in the current state are not matched with the goal state.

2.3 A^* search with Manhattan Distance

When the algorithm attempts to choose which one of the current nodes in the queue is to be expanded, according to A^* search, the algorithm calculates a cost function adding the current cost $g(n)$ and the heuristic cost $h(n)$ based on the Manhattan Distance between the tiles which are not in order with the goal state. Manhattan Distance between two grid in a matrix is the sum of the absolute value of the difference of the row indices and column indices excluding the tile numbered zero as it is the blank grid of the given matrix.

In the given example below, the heuristic calculates the Manhattan distance for the Misplaced tiles (bold faced in the example). $h(n) = 2(tilenumber\ 8) + 2(7) + 1(5) + 1(4) = 8$

Misplaced Tiles Heuristic Example

Goal State

1	2	3
4	5	6
7	8	0

Current State

1	2	3
8	**7**	6
5	**4**	0

3 Comparisons among 3 heuristics used

3.1 Test cases used for the experiment results

The project implementation considers the test cases provided in the project description.

Depth of solution to test cases {0, 2, 4, 8, 12, 16, 20, 24} left to right

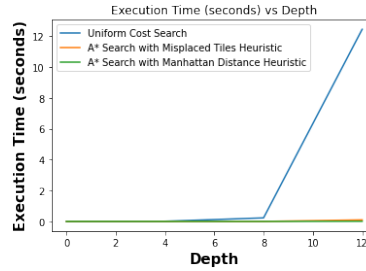| 1	2	3		1	2	3		1	2	3		1	3	6		1	3	6		1	6	7		7	1	2		0	7	2
4	5	6		4	5	6		5	0	6		5	0	2		5	0	7		5	0	3		4	8	5		4	6	1
7	8	0		0	7	8		4	7	8		4	7	8		4	8	2		4	8	2		6	3	0		3	5	8

3.2 Execution Time vs Depth

Figure 1: A comparison of 3 heuristics on the Eight Puzzle Problem increasing solution depth

- As Figure 1 suggests that for uniform cost search, the execution time is exponentially increasing as the depth of the solution increases.

- In case of other two heuristics, A^* search with Manhattan Distance is faster than A^* search with Misplaced tiles.

3.3 Node count (expanded) vs Depth

- As Figure 2 suggests that for uniform cost search, the number of node count expanded is exponentially increasing as the depth of the solution increases.

- In case of other two heuristics, A^* search with Manhattan Distance does not usually expand as much as nodes expanded in case of A^* search with Misplaced tiles.
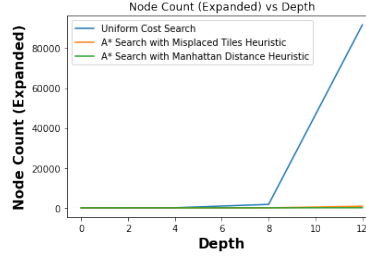
Figure 2: A comparison of 3 heuristics on the Eight Puzzle Problem for increasing solution depth
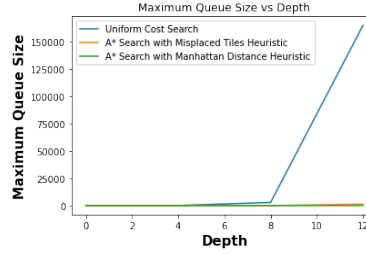

Figure 3: A comparison of 3 heuristics on the Eight Puzzle Problem for increasing solution depth

3.4 Maximum queue size vs Depth

- As Figure 3 suggests that for uniform cost search, the maximum queue size during the search at any time is exponentially increasing as the depth of the solution increases.

- In case of other two heuristics, A^* search with Manhattan Distance does not usually have as much as nodes in the queue in case of A^* search with Misplaced tiles.

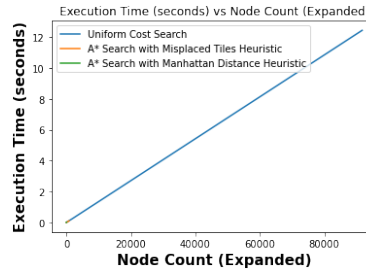## 3.5 Execution time vs Node count (expanded)

Figure 4: A comparison of 3 heuristics on the Eight Puzzle Problem for increasing node count

- As Figure 4 suggests that for uniform cost search, the execution time is linearly increasing as the number of node count expanded during the search increases.

- In case of other two heuristics, A^* search with Manhattan Distance needs less execution time than A^* search with Misplaced tiles, since A^* search with Manhattan Distance does not usually have as much as nodes expanded as that of A^* search with Misplaced tiles because of heuristic cost calculated.
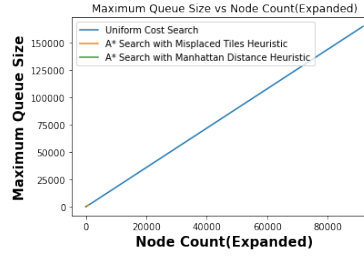
3.6 Maximum queue size vs Node count

Figure 5: A comparison of 3 heuristics on the Eight Puzzle Problem for increasing node count expansion

- As Figure 5 suggests that for uniform cost search, the maximum queue size at any instant of the search is linearly increasing as the number of node count expanded during the search increases.

- In case of other two heuristics, A^* search with Manhattan Distance finds smaller queue size than A^* search with Misplaced tiles at any instant, since A^* search with Manhattan Distance does not usually have as much as nodes expanded as that of A^* search with Misplaced tiles because of heuristic cost calculated.

4 Instructions to run the source code

The source is uploaded https://github.com/mmasu012/Eight-Puzzle. There is a jupyter notebook which can be run to test the implementation. There is another file to generate the plots used for analysis across 3 heuristics discussed in the report.

5 Results of the experiment

		Max Queue Size	Node Expaned	Running Time
Depth = 8				
	Uniform	2918	1743	0.2
	Misplaced	29	18	0
	Manhattan	21	12	0
		Max Queue Size	Node Expaned	Running Time
Depth = 12				
	Uniform	165004	91661	12.4
	Misplaced	1325	818	0.1
	Manhattan	82	54	0
		Max Queue Size	Node Expaned	Running Time
Depth = 16				
	Uniform			
	Misplaced	65035	37856	5
	Manhattan	1274	770	0.1

Figure 6: Results for depth 0, 2, 4

6 Conclusion

- Eight puzzle problem needs huge computation power when the solution depth is bigger.

- Manhattan distance heuristic performs better than the Misplaced Tiles heuristics in respect of space and time complexities.

- Uniform cost search needs a huge amount of execution time for solution depth 16 20 and 24.

	Max Queue Size	Node Expaned	Running Time
Depth = 0			
Uniform	1	0	0
Misplaced	1	0	0
Manhattan	1	0	0

	Max Queue Size	Node Expaned	Running Time
Depth = 2			
Uniform	4	3	0
Misplaced	3	2	0
Manhattan	3	2	0

	Max Queue Size	Node Expaned	Running Time
Depth = 4			
Uniform	82	57	0.01
Misplaced	8	4	0
Manhattan	8	4	0

Figure 7: Results for depth 8, 12, 16

	Max Queue Size	Node Expaned	Running Time
Depth = 20			
Uniform			
Misplaced	1556989	853624	147.7
Manhattan	36070	22298	3.1

	Max Queue Size	Node Expaned	Running Time
Depth = 24			
Uniform			
Misplaced			
Manhattan	487095	311068	52.3

Figure 8: Results for depth 20, 24

- Manhattan distance can work for all the test cases in a very short execution time, where as when the depth is 24, even misplaced tiles needs huge amount of time.

- Due to taking more amount of execution time, the comparative analysis presents for the testcases with solution depth upto 12.

- All the statistics experimented from solution depth 0 to depth 24 have been attached in the report.