CS205: Introduction to Artificial Intelligence, Dr. Eamonn Keogh

Project 1: 8 Puzzle

Name: Sourav Singha
SID: 862323554
Email Id: ssing263@ucr.edu
Date: May 14th 2022

In completing this assignment, I consulted:

- The Blind Search 1 & 2 and Heuristic Search lecture slides by Dr. E Keogh and notes annotated from lecture.
- Python 3.8 docs from https://docs.python.org/3.8/
- N-Puzzle board game https://en.wikipedia.org/wiki/15_puzzle
- Project 1 guidelines Dr. Eamonn Keogh for report structure and outline
- For sorting using the priority queue:
  Stackoverflow (https://stackoverflow.com/questions/8796886/is-it-safe-to-just-implement-lt-for-a-class-that-will-be-sorted),
  linuxtut (https://linuxtut.com/en/441a6bf31175ec339050/)

The source code and algorithms project uses are an original work and developed solely for fulfilling the requirement of CS 205 Artificial Intelligence Project 1 submission. The utility functions and logic that are referred to are the following:

- Priority Queue: iteration, add, get of items, setting the sorting order
- Python lambda functions and loop control sub routines
- Python string to function dispatcher.
- Project 1 guidelines Dr. Eamonn Keogh for function and variable naming conventions, and test cases.

Contents

Report Summary

Introduction

The 8-puzzle game is sliding puzzle game on a 3x3 grid space with 8 numbered heads (from 1 to 8) and one empty slot. The higher/ better versions have 4x4 grid space (15 puzzle), 5x5 (24 puzzle) and so on. The game can be played by sliding the tabs horizontally or vertically to move the empty space in the direction opposite of the move head. The problem to finding a final state can be expressed as a search space problem which on which different blind search or heuristic search approaches can be used to reach the solution if there is one. In this report the search space is defined a tree structure where initial node branches out to the next nodes(state) possible by single operation (cost 1). For the search approach/ algorithm the following have been used: Uniform Cost Search, A* with the Misplaced Tile heuristic, A* with the Manhattan Distance heuristic.

Design and Working

The command line interface allows for easy access to the application and the user can select from 5 predefined puzzle states or provide their own input manually.

The design approach followed the development involved mostly around unitizing the Tree Node concept in python using python class which provided variables for storing and retrieving the state, *gn, fn, hn* and pointer to parent Node for each individual nodes/states. This enables easy storing and updating the values of the cost parameters and tracing back the path to the initial state from the solution.

For storing traversal information and visited nodes record keeping the problem state class has been defined which also is used for expanding the nodes. The function to expand the current state into its children state works by performing the index check for the expansion then and also if the children have been created before by checking its hash of the state in the visited node list

The formulas used for popping the best node from the priority is as follows is based on the value of the f(n) (lowest is selected), where f(n) = g(n) + h(n), and the cost of g(n) = is equal to the path depth and h(n) depends on the search algorithm as follows:

1. Uniform Cost Search:
   h(n) = 0
2. A* with the Misplaced Tile heuristic:
   h(n) = Number of misplaced tiles
3. A* with the Manhattan Distance heuristic.
   h(n) = sum of row and column distance of tile from its correct position except the blank (‘0’)

(Ref: Heuristic Search lecture slides by Dr. Eamonn Keogh)*

Efforts required for changing to 15-puzzle or higher

The design has been made such a way to easily update the code to function for 15 puzzle or 24 puzzle or even higher. This can be done in the following steps:

1. Updating the goal state array

2. Manhattan Distance dictionary list of actual tile location/ goal state
3. Update loop sizes for from 3 to 4, 5, … as per requirement at the custom input array loop, expansion function, Manhattan, misplaced tile heuristic size.

Comparison

The performance of the search algorithms has been performed against the following puzzles test cases:

1. By depth:

Depth 0	Depth 2	Depth 4	Depth 8	Depth 12	Depth 16	Depth 20	Depth 24
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

2. By level of difficulty:

Trivial	Very easy	Easy	Doable	Oh boy!
1 2 3	1 2 3	1 2 0	0 1 2	8 7 1
4 5 6	4 5 6	4 5 3	4 5 3	6 0 2
7 8 0	7 0 8	7 8 6	7 8 6	5 4 3

Observations

The values have been observed for the solution depth, number of nodes expanded, max queue size and time taken of individual executions

1. For known difficulty levels

Algorithm\Problem State		Trivial	Very Easy	Easy	Doable	Oh Boy
Uniform Cost Search	Solution Depth	0	1	2	4	22
	Node expanded	0	3	5	15	86810
	Max Queue size	1	5	6	16	25010
	Time	0.0	0.0	0.0	0.0	5.489135
A* Misplaced Tile heuristic	Solution Depth	0	1	2	4	22
	Node expanded	0	1	2	4	6461
	Max Queue size	1	3	3	4	3749
	Time	0.0	0.0	0.0	0.0	0.435065
A* Manhattan Distance heuristic	Solution Depth	0	1	2	4	22
	Node expanded	0	1	2	4	347
	Max Queue size	1	3	3	4	227
	Time	0.0	0.0	0.0	0.0	0.031520

Table 1: Observations for trivial, very easy, easy, doable, oh boy! Puzzles

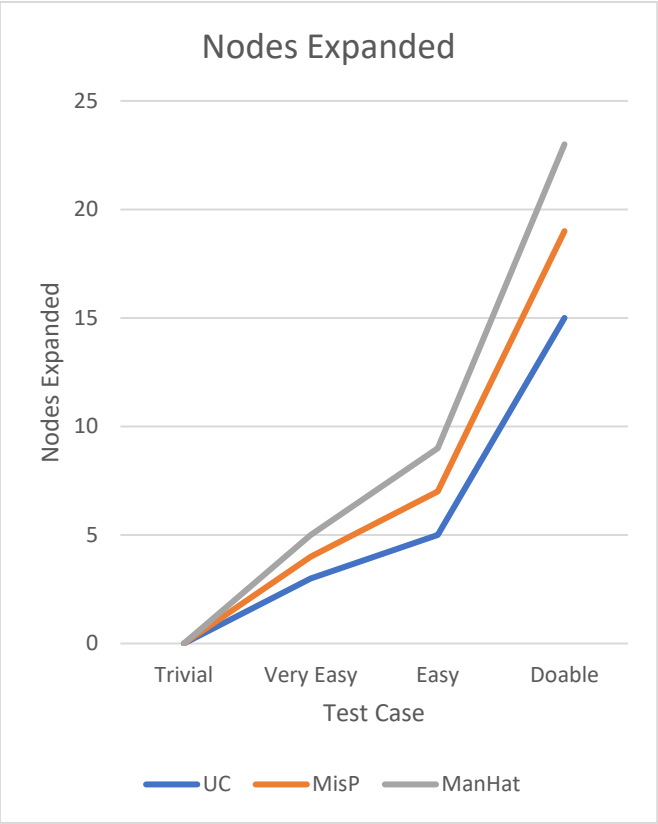


Figure 1: Nodes expanded vs difficulty level

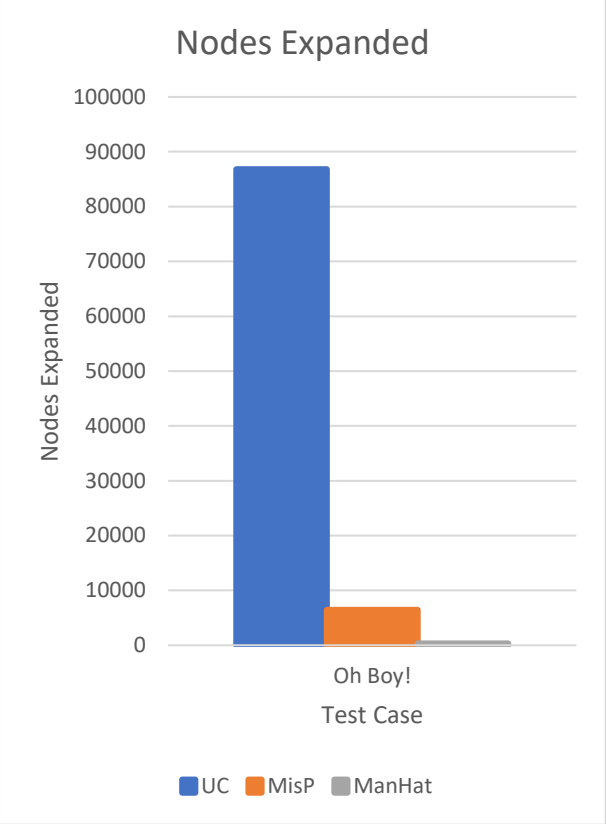


Figure 2: Nodes expanded for Oh Boy! Test case

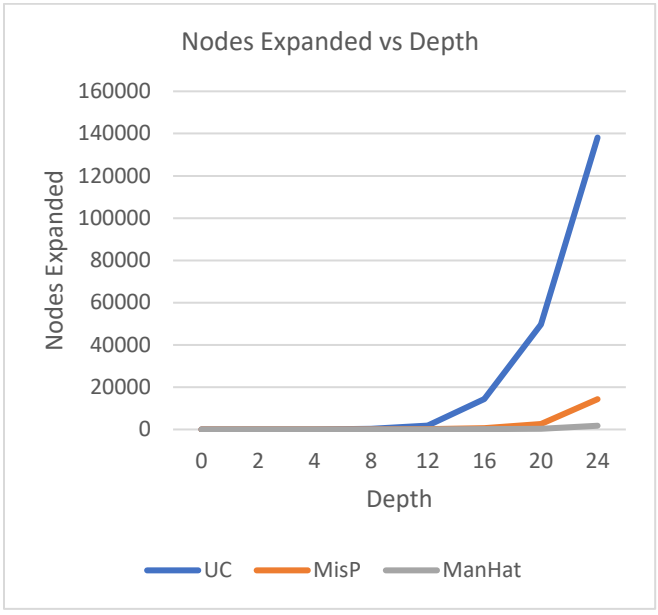
In table 1 the summarized data clearly shows as the depth of the final state increases the uniform cost algorithm perform poorly both in term of nodes expanded and time consumed. For the Oh Boy! Test case the bar chart from the figure 2 shows this specifically how the number of node expanded drastically is higher than of the heuristic algorithms A* with the Misplaced Tile heuristic, A* with the Manhattan Distance heuristic.

On further study of the data, we can also conclude that the A* with Manhattan Distance heuristic always outperform the Misplaced Tile heuristic.

2. For known difficulty levels

Algorithm\Problem Depth		0	2	4	8	12	16	20	24
Uniform Cost Search	Node expanded	0	5	32	309	1809	14350	49612	138126
	Max Queue size	1	6	26	196	1137	6767	18349	24148
	Time	0.0	0.0	0.0	0.015669	0.109777	0.974885	3.354098	9.663781
A* Misplaced Tile heuristic	Node expanded	0	2	4	18	119	647	2542	14318
	Max Queue size	1	3	6	16	84	401	1525	7406
	Time	0.0	0.0	0.0	0.0	0.0	0.047313	0.184277	1.105043
A* Manhattan Distance heuristic	Node expanded	0	2	4	12	36	188	313	1765
	Max Queue size	1	3	6	12	28	121	210	982
	Time	0.0	0.0	0.0	0.0	0.0	0.015369	0.021993	0.128795

Table 2: Observations for puzzles with known depth levels



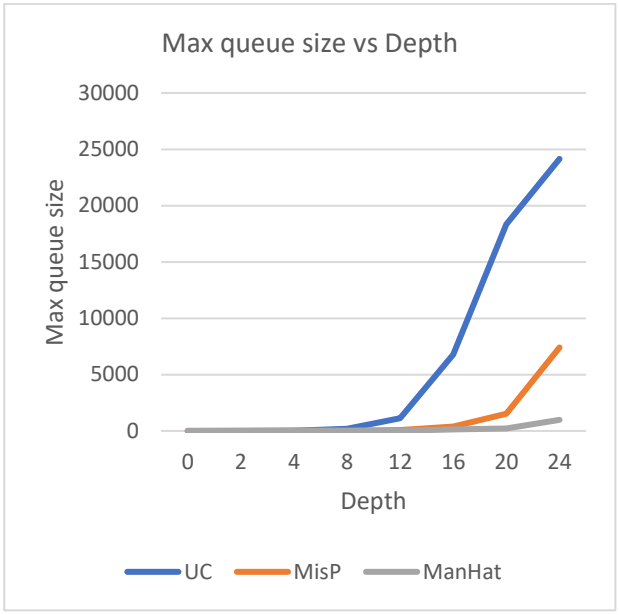
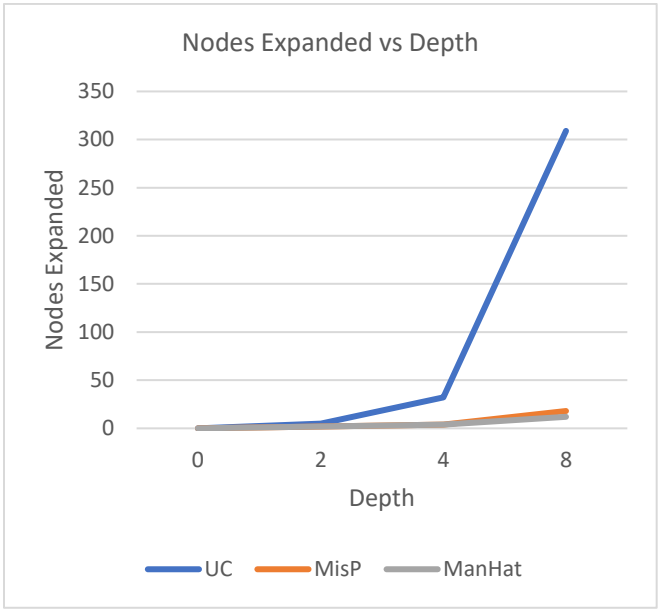


Figure 3: (left) Nodes expanded vs Depth, (right) Max queue size vs Depth expanded vs Depth



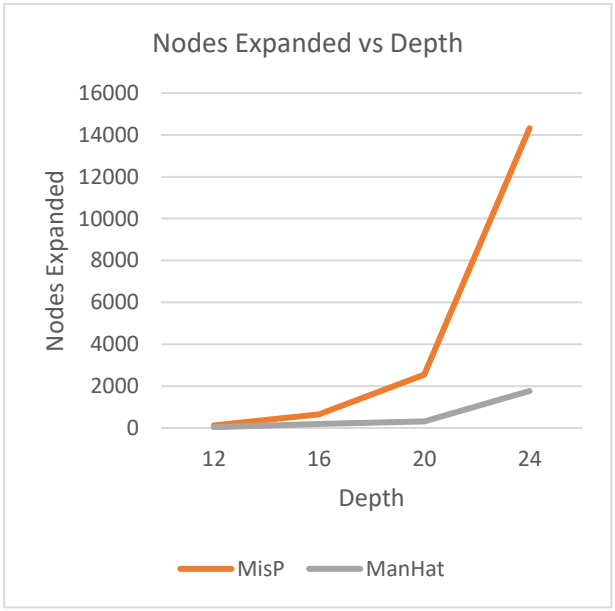


Figure 4: Nodes expanded vs Depth: (left) for depth<=8, (right) A* heuristics for test cases with depth >= 12

From the data tables on the test cases with known depth levels we can observe that the number of nodes expanded and max queue size is highest for uniform cost search. It can also be further overserved that both the heuristic algorithms perform better than the blind search algorithm: uniform cost search.

Also, we may observe from the line graphs in Figure 4 (left & right subfigures) that the Manhattan Distance heuristic outperform Misplaced tile heuristic.

Conclusion

The results of the project help to re-strength the realization that than the heuristic algorithms outperform the blind search algorithm as it only expands the nodes that has the highest known possibility of reaching closer to the goal state.

Even further the Manhattan Distance heuristic outperforms the Misplaced tile heuristic as it calculates the distance cumulative sum of the vertical and horizontal step difference of all the tiles from its desired state. Whereas the Misplaced tile algorithm only calculates the total number of misplaced tiles in the puzzle, and from the results of the project it is further validated.

Sample trace

Level Easy

```
*********************************************
CS 205, Project 1: "8-puzzle Solver"

By Name: Sourav Singha, SID: 862323554
*********************************************


Select the choice for the type of puzzle
1. Trivial
2. Very Easy
3. Easy
4. Doable
5. Oh Boy
6. Custom Input

Enter your choice: 3
Select the choice for the type of Search Algorithm
1. Uniform Cost Search
2. A* Misplaced Tile Heuristic
3. A* Manhattan Distance Heuristic

Enter your choice: 1

Expanding state

[1, 2, 0]
[4, 5, 3]
[7, 8, 6]

The best state to expand with a g(n) =  0  and h(n) =  0  is...
[1, 2, 0]
[4, 5, 3]
[7, 8, 6]

[Omitting to save space]

The best state to expand with a g(n) =  2  and h(n) =  0  is...
[1, 2, 3]
[4, 0, 5]
[7, 8, 6]
```

```
Goal state found
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]

Solution depth is:  2
Number of nodes expanded:  5
Max queue size:  6
Time to finish:  0.0  seconds

Traceback from goal to Initial Puzzle state

[1, 2, 3]
[4, 5, 6]
[7, 8, 0]

[1, 2, 3]
[4, 5, 0]
[7, 8, 6]

[1, 2, 0]
[4, 5, 3]
[7, 8, 6]

Terminating...

Process finished with exit code 0
```

Level Hard

```
**********************************************
CS 205, Project 1: "8-puzzle Solver"

By Name: Sourav Singha, SID: 862323554
**********************************************


Select the choice for the type of puzzle
1. Trivial
2. Very Easy
3. Easy
4. Doable
5. Oh Boy
6. Custom Input

Enter your choice: 6

Enter the puzzle in single space separated manner, use 0 to denote space/blank
Enter inputs for row 1: 7 1 2
Enter inputs for row 2: 4 8 5
Enter inputs for row 3: 6 3 0
Select the choice for the type of Search Algorithm
1. Uniform Cost Search
2. A* Misplaced Tile Heuristic
3. A* Manhattan Distance Heuristic

Enter your choice: 3

Expanding state

[7, 1, 2]
```

```
[4, 8, 5]
[6, 3, 0]

The best state to expand with a g(n) =  0  and h(n) =  0  is...
[7, 1, 2]
[4, 8, 5]
[6, 3, 0]

The best state to expand with a g(n) =  1  and h(n) =  11  is...
[7, 1, 2]
[4, 8, 5]
[6, 0, 3]

[Omitting to save space]

Goal state found
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]

Solution depth is:  20
Number of nodes expanded:  313
Max queue size:  210
Time to finish:  0.10029315948486328  seconds

Traceback from goal to Initial Puzzle state

[1, 2, 3]
[4, 5, 6]
[7, 8, 0]

[Omitting to save space]

[7, 1, 2]
[4, 8, 5]
[6, 3, 0]

Terminating...

Process finished with exit code 0
```

Source code Reference

The original source code used for developing and preparing the report can be found at

Github url: https://github.com/ssing263/CS-205-Artificial-Intelligence/blob/main/8_Puzzle.py