State-Space Search: Run-Time Differences in Non-Heuristic and Heuristic Algorithms on an Eight Puzzle Problem

Shawn Lee *861090401*
November 5, 2016

ABSTRACT

Search algorithms are the fundamental requirements for any kind of search problems implemented by various data structures such as a search tree. This report summarizes and compares three algorithms: uniform cost search, A* with a misplaced tile heuristic, and A* with a Manhattan distance heuristic. This report also provides a general outline of the implementation of the Eight Puzzle Program and the three algorithms listed above.

1. INTRODUCTION

This report is a summary and comparison of the different search algorithms - one that which is non-heuristic and two of which are - based on the time and space differences while solving the Eight Puzzle Problem.

The report is organized into six sections. In the second Section, I will present the test case used and the results the algorithms generate to solve the Eight Puzzle Problem. The next two Sections summarizes the results and makes comments on the utility of the different heuristics, respectively. Section 5 yields the conclusion of the report. The last section is an outline and commentary about the implementation of the Eight Puzzle Program.

2. TEST CASES AND ALGORITHM RESULTS

For any 3x3 test case in the Eight Puzzle Program, there is a possibility that the test case is not solvable.

4	2	8
6	0	3
7	5	1

Table 1: First solvable test case

1	2	3
4	5	6
7	8	0

Table 2: Goal State

Table 1 shows a solvable test case used to generate results of each algorithm, wherein Table 2 shows the desired goal state for the 3x3 grid.

	Expanded	Max Nodes in Queue	Depth
Uniform Cost	80983	18858	20
A* with Misplaced Tile	5078	1838	20
A* with Manhattan Distance	1818	656	22

Table 3: Results of test case

Table 3 shows the results after running the solvable test case as presented in Table 1. In the table:

- *Expanded* defines the time taken to solve the test case
- *Max Nodes in Queue* defines the space required to solve the test case
- *Depth* defines the number of moves required to find the solution as presented in Table 2

In another example,

8	7	2
1	3	5
6	0	4

Table 4: Second test case

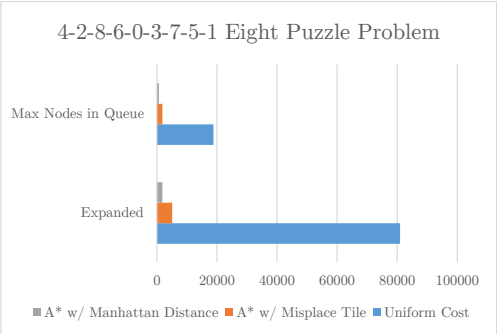
Expanded	Max Nodes in Queue	Depth
55850	14747	19
2511	933	19
387	146	21

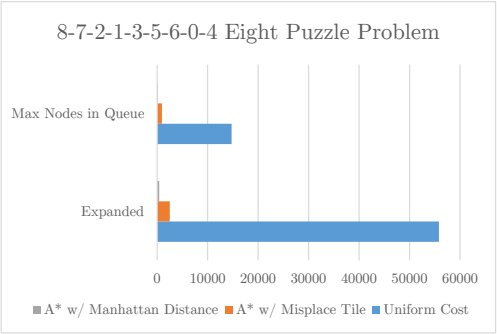
Table 5: Results of test case

Table 4 shows another test case for which the Eight Puzzle Problem was able to solve. The results are also provided in the table next to it.

3. SUMMARY OF RESULTS

So far I have considered two cases. In the first case, it is apparent that the Uniform Cost Search is much slower than the two A* algorithms. Similarly, it can be observed that is true as well in the second case. Below are the graphs that represent the results of the first and second cases, respectively.





4. COMPARING DIFFERENT ALGORITHMS

Since Uniform Cost Search does not rely on heuristics, such that $h(n)$ is hard-coded to 0, I observe that it is much slower than the two A* algorithms, which do rely on heuristics. As seen in both test cases, the Uniform Cost Search expands 80983 and 55850 nodes in contrast to the 5078 and 2511 nodes that the A* with Misplaced Tile Heuristic expands. By dividing the results of Uniform Cost Search by A* with Misplaced Tile Heuristic, we can conclude that including the Misplaced Tile Heuristic is approximately 15 times more time efficient in the first case, and 22 times more time efficient in the second case. Furthermore, it is estimated to be 10 times more space efficient in the first case, and 15 times more space efficient in the second case.

Although the differences may seem less significant between the two heuristic algorithms, it can be observed that the Manhattan Distance Heuristic is more efficient than the Misplaced Tile Heuristic and even more so in contrast to Uniform Cost Search. By dividing the results of the Misplaced Tile Heuristic over the Manhattan Distance Heuristic, we can conclude that the Manhattan Distance Heuristic is approximately 2.5 times more time efficient in the first case, and 6 times more time efficient in the second case. Also, it is 3 times more space efficient in the first case, and 6 times more space efficient in the second case.

Similarly, by doing the same calculations with Uniform Cost Search, we can conclude that the Manhattan Distance Heuristic is approximately 44 times more time efficient in the first case, and 144 times more time efficient in the second case. Furthermore, it is approximately 28 times more space efficient in the first case, and 100 times more space efficient in the second case.

5. CONCLUSION OF FINDINGS AND RESULTS

From the calculations and the graphs from each of test case, it can be concluded that the Manhattan Distance Heuristic is the most efficient, followed by the Misplaced Tile Heuristic, and lastly Uniform Cost Search. Furthermore, including a heuristic to any search algorithm ensures a quicker search time to the goal state over one without. In this Eight Puzzle Program, the time and space differences occur with the use of different search algorithms, notably with heuristics being the more efficient. Hence, we see that the Manhattan Distance Heuristic gives a good approach to reaching the goal state in Eight Puzzle Problems due to its effectiveness.

6. IMPLEMENTATION OF THE EIGHT PUZZLE PROGRAM

In this section, I describe the process and design of an efficient implementation of my Eight Puzzle Program.

6.1 Compiler and Language: g++ 5.3, C++11

6.2 General Problem: Find a path from a *start state* to a *goal state* given:

- A goal test to see if a given state is the goal state
- Given a parent state, generate a child state that does a swap between the blank and its neighboring tiles

6.3 Design and Implementation: A *state* is a representation of a physical configuration. In this case, I represent the state as a vector<**int**>. Although vector <vector<**int**>> was an obvious option, a 1D vector is more efficient to traverse and requires less code to implement the program.

A *node* is a data structure constituting part of a search tree that include the *depth*, *path cost* $g(n)$, and *heuristic cost* $h(n)$. This was represented in a struct. A class contains this struct, inherited by the class that implements the Eight Puzzle Problem. Path cost increments by one each time a new child is created

from switching a blank and neighboring tile. Heuristic cost increments by:

- the number of tiles that differ from the goal state in the Misplaced Tile Heuristic, and
- the distance from current tile to the goal state's tile in the Manhattan Distance Heuristic.

I followed the pseudocode as found in the reference below to implement the general tree search that which handles repeated states and searches using Uniform Cost Search, A* with Misplaced Tile Heuristic, or A* with Manhattan Distance Heuristic.

In Uniform Cost Search, the *fringe* - which contains not yet expanded nodes - and *expanded* - which contains all expanded nodes - are implemented using **queue**s. However, to track heuristic cost, the two A* search algorithms use **priority_queue**s instead.

6.4 Default and Trace: The following are automatically generated defaults of my Eight Puzzle Program:

1	2	3
4	0	6
7	5	8

Table 6: Default Test

1	2	3
4	5	6
7	8	0

Table 7: Default Goal State

Note that the default state is to solve a 3x3 grid; however, the program allows the user to test any *N by N* grid, where *N* is an integer. A *N by N* solution will be automatically generated based on the *N by N* grid containing the values $1, 2, 3, ..., N^2 - 2, N^2 - 1, 0$.

In **Section 8: Appendix**, three typescripts can be found which are based on the default test case that run through each of the above listed search algorithms.

6.5 Makefile: A Makefile is provided to compile and test the program. To use the Makefile:

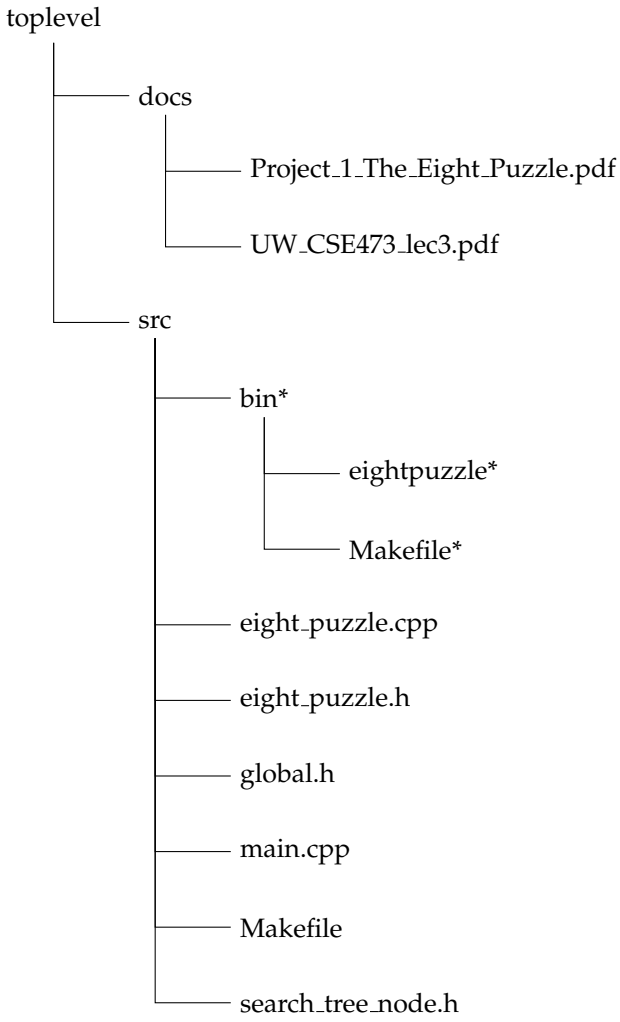
1. Enter the src directory and type make in the terminal.
2. Enter the bin directory (cd bin/) and type make check in the terminal.
3. A command-line interface will allow the user to select and enter any tests.

Please note that g++-5 and c++11 are prerequisites to using the Makefile. The user may edit the CC variable in the Makefile to g++-4 if necessary.

6.5 Project Directory:

You may find this project at:

https://github.com/shawnjzlee/EightPuzzle



* - this directory or file is generated after make is executed.

7. REFERENCES

[1] Chapter 3 - Problem Solving using Search. (n.d.). Lecture. Retrieved November 2, 2016, from courses.cs.washington.edu/courses/cse473/12au/slides/lect3.pdf

8. APPENDIX

```
./eightpuzzle
Type "1" to use a default puzzle, or "2" to enter your own puzzle.
1
Override default goal state?:
        1.      Yes
        2.      No
2
Enter your choice of algorithm:
        1.      Uniform Cost Search
        2.      A* with Misplaced Tile heuristic
        3.      A* with Manhattan distance heuristic
1
The best state to expand with a g(n) = 0 is...
1 2 3
4 0 6
7 5 8

Expanding this node...
The best state to expand with a g(n) = 1 is...
1 0 3
4 2 6
7 5 8

Expanding this node...
The best state to expand with a g(n) = 1 is...
1 2 3
4 5 6
7 0 8

Expanding this node...
The best state to expand with a g(n) = 1 is...
1 2 3
0 4 6
7 5 8

Expanding this node...
The best state to expand with a g(n) = 1 is...
1 2 3
4 6 0
7 5 8

Expanding this node...
The best state to expand with a g(n) = 2 is...
1 2 3
4 0 6
7 5 8

Expanding this node...
The best state to expand with a g(n) = 2 is...
0 1 3
4 2 6
7 5 8

Expanding this node...
The best state to expand with a g(n) = 2 is...
1 3 0
4 2 6
7 5 8

Expanding this node...
The best state to expand with a g(n) = 2 is...
1 2 3
4 5 6
0 7 8

Expanding this node...
Solution found.
1 2 3
4 5 6
7 8 0

To solve this problem, the uniform cost expanded a total of 16 nodes.
The maximum number of nodes in the queue at any one time was 9.
The depth of the goal node was 2.
```

```
./eightpuzzle
Type "1" to use a default puzzle, or "2" to enter your own puzzle.
1
Override default goal state?:
        1.      Yes
        2.      No
2
Enter your choice of algorithm:
        1.      Uniform Cost Search
        2.      A* with Misplaced Tile heuristic
        3.      A* with Manhattan distance heuristic
2
The best state to expand with a g(n) = 0 and h(n) = 3 is...
1 2 3
4 0 6
7 5 8

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

Expanding this node...
Solution found.
1 2 3
4 5 6
7 8 0

To solve this problem, the A* with misplaced tile heuristic expanded a total of 7 nodes.
The maximum number of nodes in the queue at any one time was 5.
The depth of the goal node was 2.
```

```
./eightpuzzle
Type "1" to use a default puzzle, or "2" to enter your own puzzle.
1
Override default goal state?:
        1.      Yes
        2.      No
2
Enter your choice of algorithm:
        1.      Uniform Cost Search
        2.      A* with Misplaced Tile heuristic
        3.      A* with Manhattan distance heuristic
3
The best state to expand with a g(n) = 0 and h(n) = 5 is...
1 2 3
4 0 6
7 5 8

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

Expanding this node...
Solution found.
1 2 3
4 5 6
7 8 0

To solve this problem, the A* with manhattan distance heuristic expanded a total of 7 nodes.
The maximum number of nodes in the queue at any one time was 5.
The depth of the goal node was 2.
```