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

Austin Lee
SID: 862084022
Email: alee235@ucr.edu
Github Repo Link: https://github.com/austinslee/CS170-project-1.git
November 8, 2021

In completing this assignment, I consulted

- 2_Blind Search_part2.pptx and 3_Heuristic Search.pptx from class slides provided in Elearn.
- Cplusplus.com for priority_queue functions. priority_queue - C++ Reference (cplusplus.com)
- Geekforgeeks.com for priority_queue operator overloading. STL Priority Queue for Structure or Class - GeeksforGeeks
- Consulted Project_1_The_Eight_Puzzle_CS_170_2021.pdf provided by Professor Keogh for instructions and report outline.
- Consulted Geeksforgeeks.com for measuring execution time of function Measure execution time of a function in C++ - GeeksforGeeks

Outline of this report:

- Cover page: Page 1
- My Report: Pages 2 -
- Sample trace on low depth problem, Page
- Sample trace on high depth problem, Page
- My code. Github link: https://github.com/austinslee/CS170-project-1.git

CS170: Assignment 1: The Angelica Puzzle

Austin Lee, SID: 862084022, November 11, 2021

Introduction:

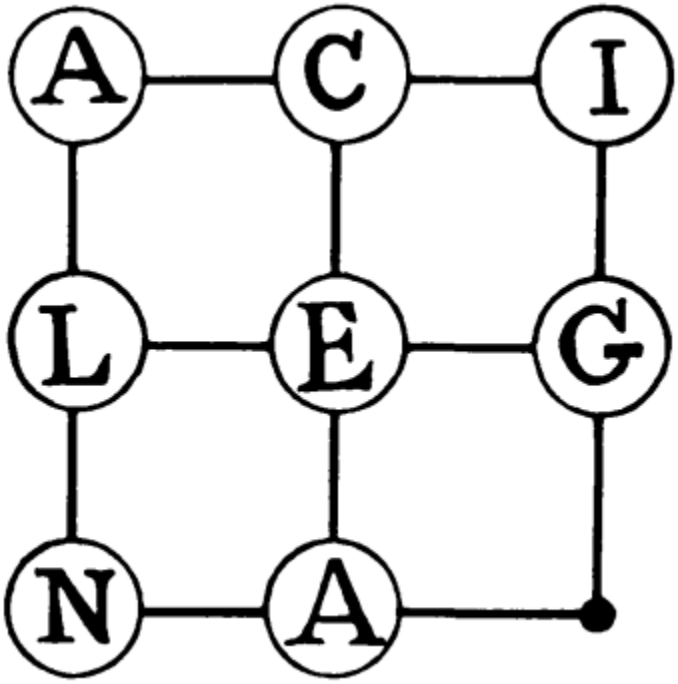
The Angelica puzzle is a puzzle introduced by Henry Ernest Dudeney, in his book 536 Puzzles & Curious Problems.

Quoting Henry Ernest Dedeney for his description of the puzzle, "The intersecting points eight lettered counters as shown in our illustration. The puzzle is to move the counters, one at a time, along the lines from point to vacant point until you get them in the order ANGELICA, thus: ."

A N G
E L I
C A .

Goal State

The initial state given by Henry Ernest Dedeney is



My goal for Professor Keogh's Project 1 is to solve this Angelica Puzzle using the knowledge of searches and artificial intelligence gained from his Fall 2021, Introduction to AI course. I will implement Uniform Cost Search and A* search with the Misplaced Tile and Manhattan Distance heuristics. I have written my code using C++ and have utilized the following libraries: iostream, stdio.h, stdlib.h, queue, vector, algorithm, and chrono.

My project has the following operators: Moving the blank space up, moving the blank space down, moving the blank space right, and moving the blank space left.

Introduction of Algorithms:

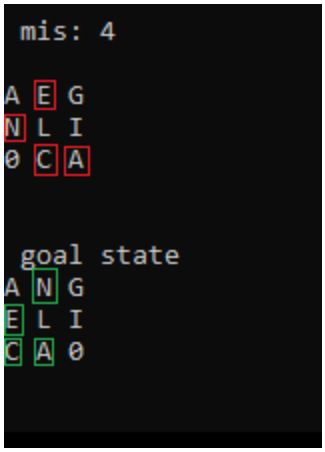
In this Project 1 report, I will be analyzing and comparing the following searches on the Angelica Puzzle: Uniform Cost Search, A* search using Misplaced Tile heuristic, and A* search using the Manhattan Distance heuristic.

Uniform Cost Search:

As stated by Professor Keogh in his Project 1 instructions and Blind search ppt, the "Uniform Cost Search is just A* with h(n) hardcoded to equal zero." and UCS(Uniform Cost Search) "expands the cheapest node. Where the cost is the path cost G(n). In this case, the h(n) represents the heuristic cost in the A* algorithm. So the Uniform Cost search will only take note of g(n) which is the cost needed to get to a node/state. In my project, the cost to get from one state to another is the amount of operations needed to get to that state. Please note that all operations have an equal cost of 1 to represent the same amount of effort is required in all of the operations. In other words, the g(n) would be equivalent to the depth of the expanded node.

The Misplaced Tile Heuristic:

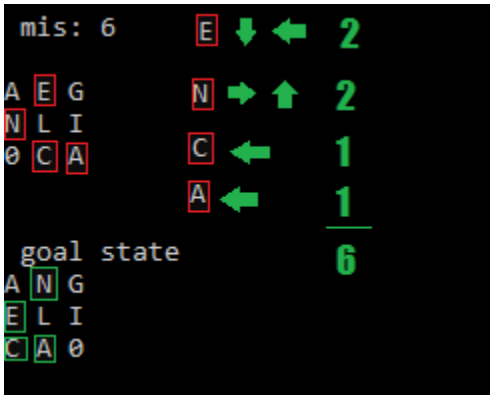
The Misplaced Tile Heuristic uses the A* search which takes the h(n) and g(n) and combines them to assign a node with a total cost. The search then expands the node with the lowest total cost until the goal state is achieved. The Misplaced Tile Heuristic counts the number of tiles that are not in the correct place. An example will be shown below.



As you can see the current state has 4 letters E, N, C, and A in the incorrect positions. This leads to a Misplaced Tile heuristic cost of 4. Please note that the Misplaced Tile heuristic does not count 0 which is a placeholder for the blank tile.

The Manhattan Distance:

The Manhattan Distance Heuristic uses the A* search which takes the h(n) and g(n) and combines them to assign a node with a total cost. The search then expands the node with the lowest total cost until the goal state is achieved. The Manhattan Distance calculates its h(n) by looking at all the tiles in the incorrect positions and adding the amount of operations needed to get the tiles in the correct position. An example will be shown below.

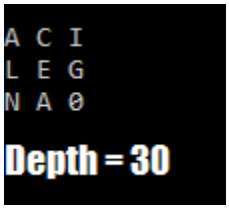


There are 4 letters in the incorrect positions: E, N, C, A. In order to get the E in the correct position, you must move it down once and left once for a cost of 2. You must move the N right once and up once for a cost of 2. You must move the C left once for a cost of 1. You must move the A left once for a cost of 1. In total you would have a total cost of 2 + 2 + 1 + 1 = 6 for the Manhattan Distance heuristics. I would like to note that you run into a slight problem when you have duplicate letters, in this case 'A', because

that letter may fit into multiple positions and the heuristic may not know which position may be the most optimal to move to. In order to solve this problem, I had my heuristic calculate the cost of moving the first A into the first position and the second A into the last position and add that up to an integer called firstSum. I then calculated the cost of moving the first A into the last position and the second A into the first position and summed it up into an integer called secondSum. I then compared the two sums and chose the sum with the lower cost. This way my algorithm will calculate the cost both ways and choose the most optimal pathing to achieve the lowest cost possible.

Comparison of Algorithms on Sample Puzzles

In order to test my algorithms on the efficacy of different solutions with wildly varying solutions, I chose 4 different initial states with optimal solutions of 2, 13, 21, 30 depth. Please note that in order to test time taken, I used the chrono library. The time taken changed slightly on every run so I took the average time taken.



	Uniform Cost	Misplaced Tile	Manhattan Distance
Depth of Solution	2	2	2
Nodes Explored	5	3	3
Max queue size	8	4	4
Time taken(s)	0.000495	0.000459	0.000464

The puzzle with a solution depth of 2 had all three algorithms with very similar statistics. They all took around 0.00045 seconds to complete the puzzle and had very similar nodes explored and max queue size with Uniform Cost only slightly more nodes explored and max queue size.

	Uniform Cost	Misplaced Tile	Manhattan Distance
Depth of Solution	13	13	13
Nodes Explored	2807	185	84
Max queue size	2873	198	81
Time taken(s)	0.517113	0.007472	0.003455

The puzzle with a solution depth of 13 had Misplaced Tiles twice as slow as the Manhattan Distance. Uniform Cost about 15 times slower than Manhattan Distance. Added to that Misplaced tiles had almost double nodes explored and max queue size than Manhattan Distance. While Uniform Cost had 33.416 times the nodes explored than Manhattan Distance and 35.47 times the max queue size than Manhattan Distance.

	Uniform Cost	Misplaced Tile	Manhattan Distance
Depth of Solution	21	21	21
Nodes Explored	65832	5822	1479
Max queue size	48503	5812	1364
Time taken(s)	298.244533	2.410811	0.185023

The puzzle with a solution depth of 21 had Misplaced Tiles 13.03 times slower than Manhattan Distance. Uniform Cost about 1611.93 times slower than Manhattan Distance. Added to that Misplaced tiles had 3.94 times the nodes explored and 4.26 times the max queue size than Manhattan Distance. While Uniform Cost had 44.51 times the nodes explored than Manhattan Distance and 35.56 times the max queue size than Manhattan Distance.

	Uniform Cost	Misplaced Tile	Manhattan Distance
Depth of Solution	N/A	30	30
Nodes Explored	N/A	99523	17751
Max queue size	N/A	58032	14533
Time taken(s)	N/A	628.189	17.346

The puzzle with a solution depth of 30 had Misplaced Tiles 36.16 times slower than Manhattan Distance. Added to that Misplaced tiles had 5.60 times the nodes explored and 3.99 times the max queue size than Manhattan Distance. I was unable to achieve a solution with Uniform Cost search because of a lack of time. I ran the program for an hour but a solution has yet to be reached. However, I can theorize that the time needed for Uniform Cost search would be well over 7.8 hours. Based on the trend that I have observed, as the solution depth increases so does the time difference between Uniform Cost and Manhattan Distance. When solution depth was 13, Uniform Cost was 13 times slower. When solution depth was 21, Uniform Cost was 1611 times slower. If this trend continues, then at minimum Uniform cost would be 1611 times slower(probably a lot more than this) than Manhattan Distance for a solution depth of 30. Based on this I can do 17.346 seconds (Manhattan Distance time taken) * 1611 to get 27944.406 seconds which when translated to hours is roughly 7.8 hours. This means I can theorize that calculating Uniform Cost for a depth of 30 would take 7.8+ hours.

Based on the data acquired, I can say with confidence that the difference between the three algorithms are negligible on easier puzzles. But as the complexity of the puzzle and solution depth increases, the time taken, nodes explored, and max queue size increases exponentially from one another. The best algorithm by far would be the A* algorithm with Manhattan Distance heuristics, the second best algorithm would be the A* algorithm with the Misplaced Tiles heuristics. The worst algorithm would be the Uniform Cost Search algorithm.

Conclusion:

Based on my observations of the three algorithms, it can be concluded that:

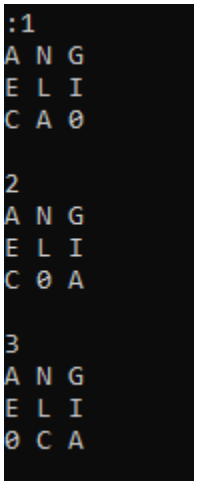
- It can be seen that out of the three algorithms, the fastest algorithm is Manhattan Distance, the second Misplaced Tiles, and last Uniform Cost.
- The difference in the time taken to solve the puzzles increases exponentially based on the complexity of the puzzle and the solution depth.
- The difference in the number of nodes explored and max queue size increase slowly based on the complexity of the puzzle and the solution depth.
- While both the Manhattan Distance and Misplaced Tiles improved on the Uniform Cost algorithm and utilize the same A* algorithm, there is a clear difference between the two. This emphasizes the importance of a good heuristic on the efficacy of the search algorithm.

Solution to puzzles:

Solution to puzzles read from biggest number to smallest number. For example, puzzle with solution depth 2 would have 3 be the initial state and 1 be the goal state.

Please follow github link for full code: https://github.com/austinslee/CS170-project-1.git
To run project, do cmake . and make. Then run the executable ./go
To switch between the different depth puzzles and algorithm used, you must comment and uncomment sections of the code. The puzzles are located in the angelica/game.cpp in the Game constructor. To switch between algorithms, go to the angelica/main.cpp and after auto time_start declaration comment and uncomment the wanted algorithm.

Puzzle with solution depth 2:



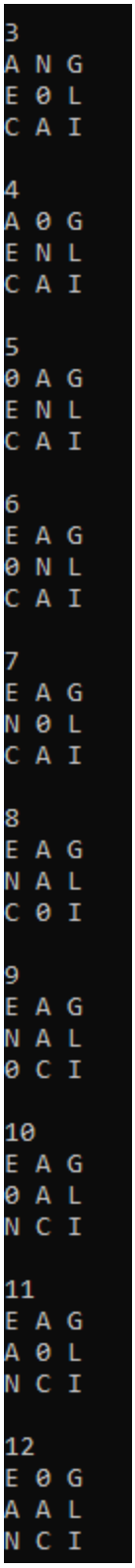
Puzzle with solution depth 13:

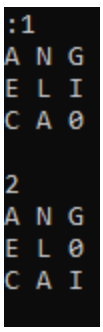
```
5
A N G
L 0 I
E C A

6
A N G
L I 0
E C A

7
A N G
L I A
E C 0

8
A N G
L I A
E 0 C

9
A N G
L 0 A
E I C

10
A N G
0 L A
E I C

11
A N G
E L A
0 I C

12
A N G
E L A
I 0 C

13
A N G
E L A
I C 0

14
A N G
E L 0
I C A
```

```
:1
A N G
E L I
C A 0

2
A N G
E L I
C 0 A

3
A N G
E L I
0 C A

4
A N G
0 L I
E C A
```

Puzzle with solution depth 21:


12
E 0 G
A A L
N C I
13
0 E G
A A L
N C I
14
A E G
0 A L
N C I
15
A E G
N A L
0 C I
16
A E G
N A L
C 0 I
17
A E G
N 0 L
C A I
18
A E G
N L 0
C A I
19
A E G
N L I
C A 0
20
A E G
N L I
C 0 A
21
A E G
N L I
0 C A
22
A E G
0 L I
N C A


3
A N G
E 0 L
C A I
4
A 0 G
E N L
C A I
5
0 A G
E N L
C A I
6
E A G
0 N L
C A I
7
E A G
N 0 L
C A I
8
E A G
N A L
C 0 I
9
E A G
N A L
0 C I
10
E A G
0 A L
N C I
11
E A G
A 0 L
N C I
12
E 0 G
A A L
N C I


:1
A N G
E L I
C A 0
2
A N G
E L 0
C A I

Puzzle with solution depth 30: