CS170 Project #1 Puzzle

Jingfang Guan
862218775
02/09/2023

The challenges that I encountered:

- Time management, I should have started the project earlier than I did so I have more time to optimise and reorganise my code.

- Code management, during the coding part of the project, I realised how unclean my coding style was. I was having a hard time finding my variables and function names. I coded everything in a single c++ file which is not clean at all.
-
- Explored set, at the beginning, I was struggling on how to put matrices into a set. I ended up changing the matrices to int vectors and storing them in a map of vectors.

- Pointer and swap pointer's structure, I accidentally deleted a line of node pointer that caused the matrix value to become empty int and. I used helper functions to find out something's wrong in the node structure and solved it at the end.

- The Euclidean distance, locating the misplaced tile and calculating the distance to the correct position is tricky.

- Design of the code, the program is not user friendly enough for avoiding invalid input.

Design in c++:

General:

- Main function for user input, choosing default or customising different puzzles, choosing 3 options of algorithm, uniform cost search, and A star with 2 different approaches.

- Structure class for node, node pointer class, print puzzle function

Algorithm:

- calculateCost: calculate misplaced tiles' distance for A* with the misplaced Tile heuristic method

- calculateEDist: calculate misplaced tiles' euclidean distance for A* with the euclidean distance method

- Struct comp: it is for priority queue in uniform cost search compared with only the node's depth.

- Struct comp_a: for priority queue in A* search for both approached compared with misplaced tile distance + node's depth

Tree search structure and faster search:

- Check visited and insert visited function: Using global map of int vector to keep track of node state to prevent loop in searches.

- Insert visited function: convert 2d matrices into 1d vector of int, and save the vector into a map for future use in checking if the states have been explored.

Program start:

```
Welcome to jguan050 8 puzzle solver.
Print puzzle
Type '1' to use a default puzzle, or '2' to enter your own puzzle.
2
Enter number for the puzzle at 0, 0
8
Enter number for the puzzle at 0, 1
7
Enter number for the puzzle at 0, 2
1
Enter number for the puzzle at 1, 0
6
Enter number for the puzzle at 1, 1
0
Enter number for the puzzle at 1, 2
2
Enter number for the puzzle at 2, 0
5
Enter number for the puzzle at 2, 1
4
Enter number for the puzzle at 2, 2
3
8 7 1
6 0 2
5 4 3
Run with this puzzle? (y/n)
y
Choose a algorithm, enter 1 for uniform cost search, 2 for A* with the Misplaced Tile heuristic, 3 for A* with the Eucli
dean Distanceheuristic.
3
8 7 1
6 0 2
5 4 3

8 7 1
6 4 2
5 0 3

8 7 1
0 6 2
5 4 3

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

Test cases:

Trivial:

1 2 3

4 5 6

7 8 0

Very easy:

1 2 3

4 5 6

7 0 8

Easy:

1 2 0

4 5 3

7 8 6

Medium:

0 1 2

4 5 3

7 8 6

Hard:

8 7 1

6 0 2

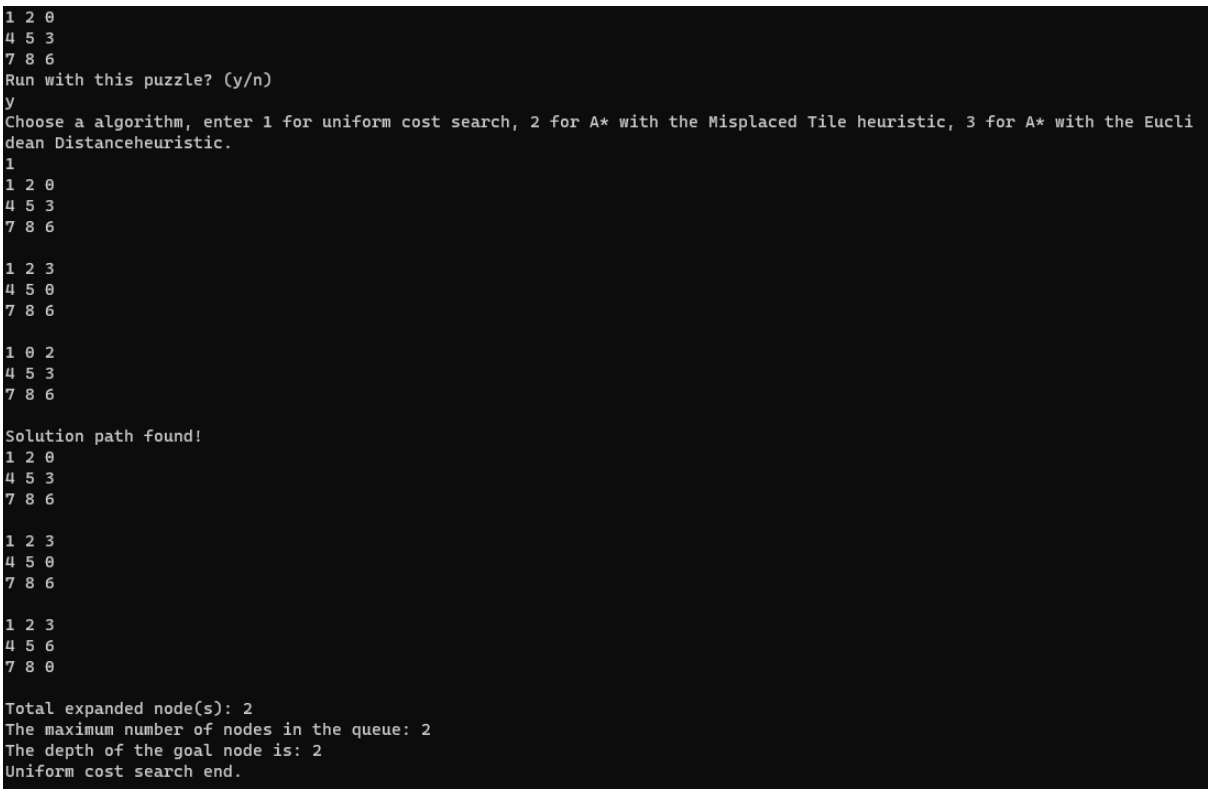
5 4 3

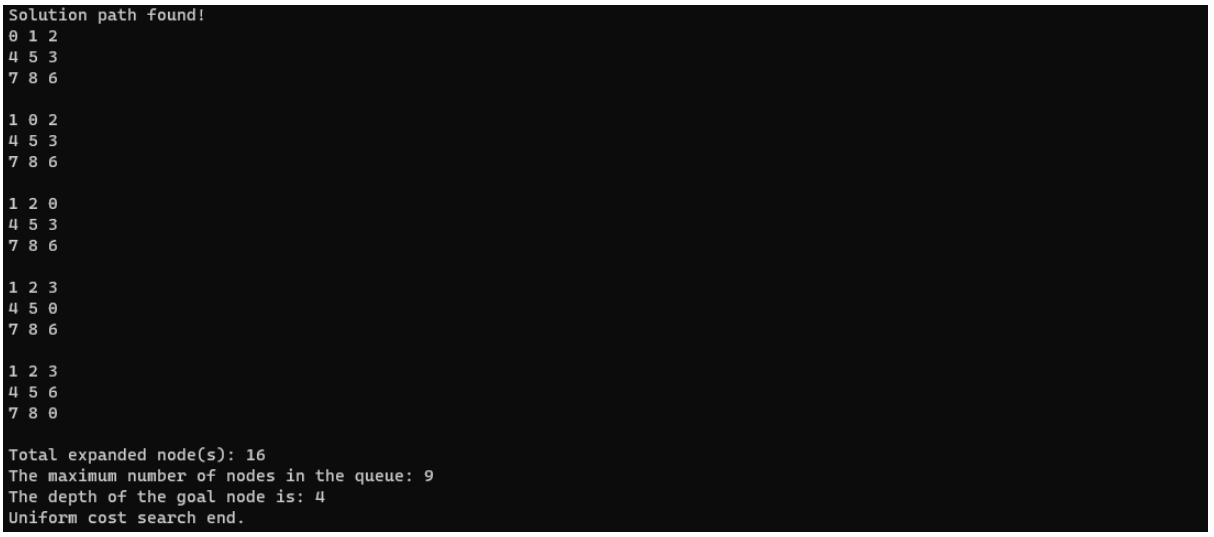
Heuristic functions comparing:

- From trivial to easy, 3 methods have very close to the same numbers of explored nodes and peak priority queue elements.

```
Welcome to jguan050 8 puzzle solver.
Print puzzle
Type '1' to use a default puzzle, or '2' to enter your own puzzle.
2
Enter number for the puzzle at 0, 0
1
Enter number for the puzzle at 0, 1
2
Enter number for the puzzle at 0, 2
3
Enter number for the puzzle at 1, 0
4
Enter number for the puzzle at 1, 1
5
Enter number for the puzzle at 1, 2
6
Enter number for the puzzle at 2, 0
7
Enter number for the puzzle at 2, 1
8
Enter number for the puzzle at 2, 2
0
1 2 3
4 5 6
7 8 0
Run with this puzzle? (y/n)
y
Choose a algorithm, enter 1 for uniform cost search, 2 for A* with the Misplaced Tile heuristic, 3 for A* with the Eucli
dean Distanceheuristic.
1
1 2 3
4 5 6
7 8 0

Solution path found!
1 2 3
4 5 6
7 8 0

Total expanded node(s): 0
The maximum number of nodes in the queue: 1
The depth of the goal node is: 0
Uniform cost search end.
--------------------------------
```

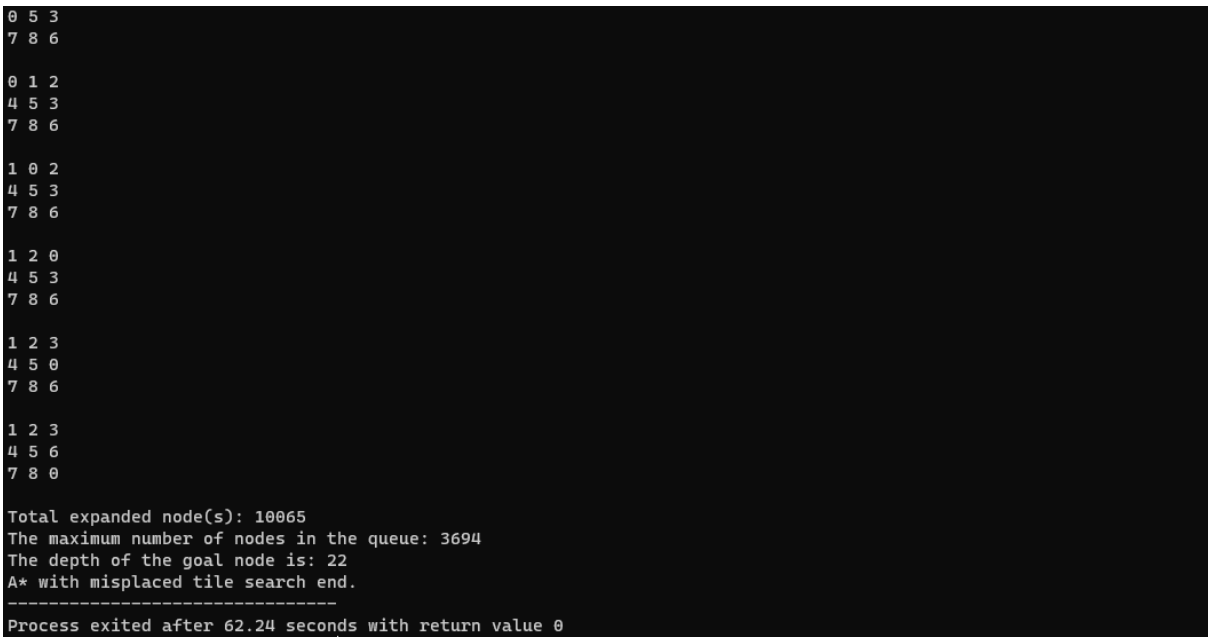
```
Welcome to jguan050 8 puzzle solver.
Print puzzle
Type '1' to use a default puzzle, or '2' to enter your own puzzle.
1
Choose a algorithm, enter 1 for uniform cost search, 2 for A* with the Misplaced Tile heuristic, 3 for A* with the Eucli
dean Distanceheuristic.
1
1 2 3
4 5 6
7 0 8

1 2 3
4 5 6
0 7 8

1 2 3
4 0 6
7 5 8

Solution path found!
1 2 3
4 5 6
7 0 8

1 2 3
4 5 6
7 8 0

Total expanded node(s): 2
The maximum number of nodes in the queue: 2
The depth of the goal node is: 1
Uniform cost search end.
--------------------------------
Process exited after 9.142 seconds with return value 0
Press any key to continue . . .
```



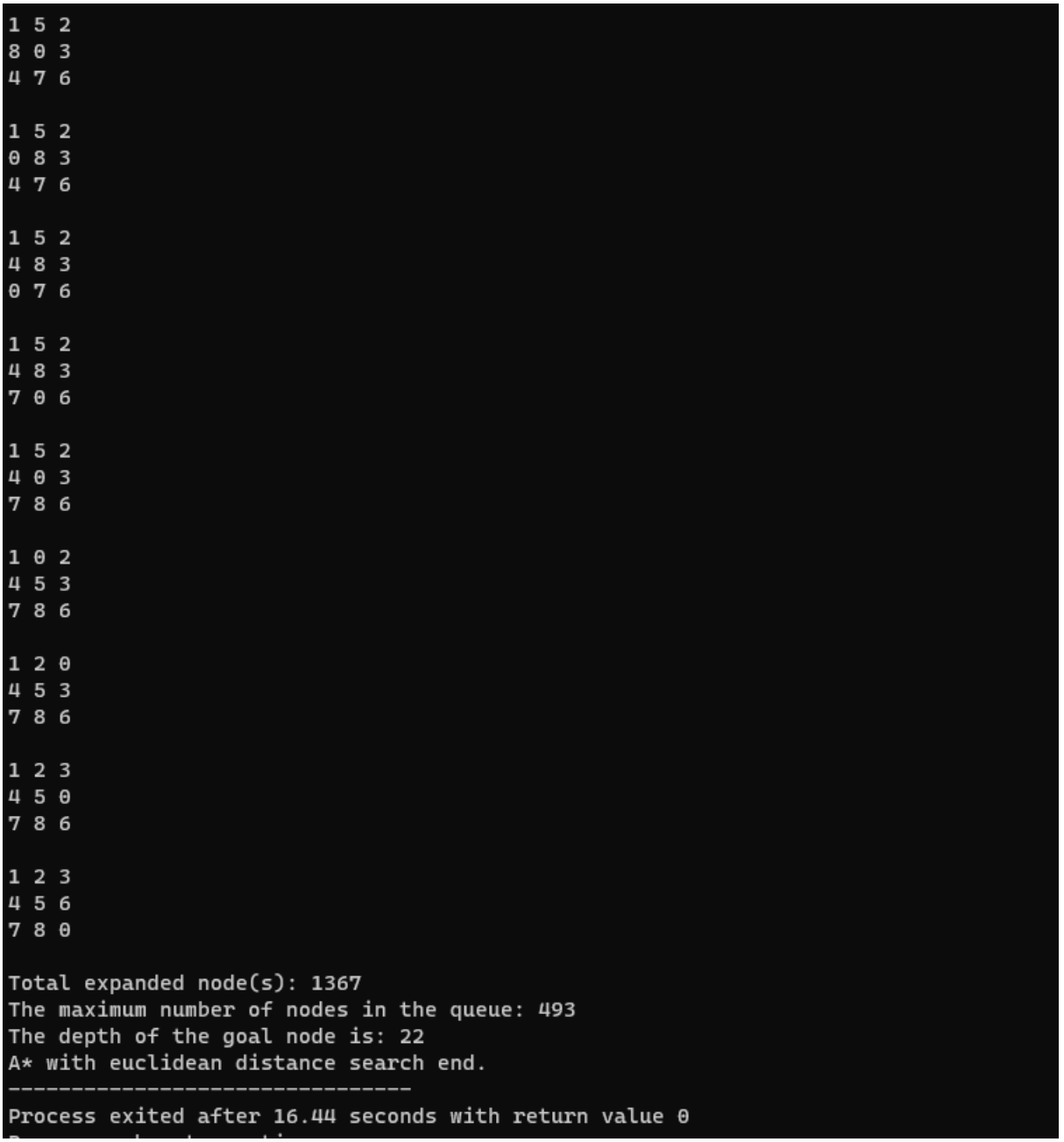


- Below is the result of my code after running a difficult 8-puzzle that uniform cost search takes more than 20 minutes.

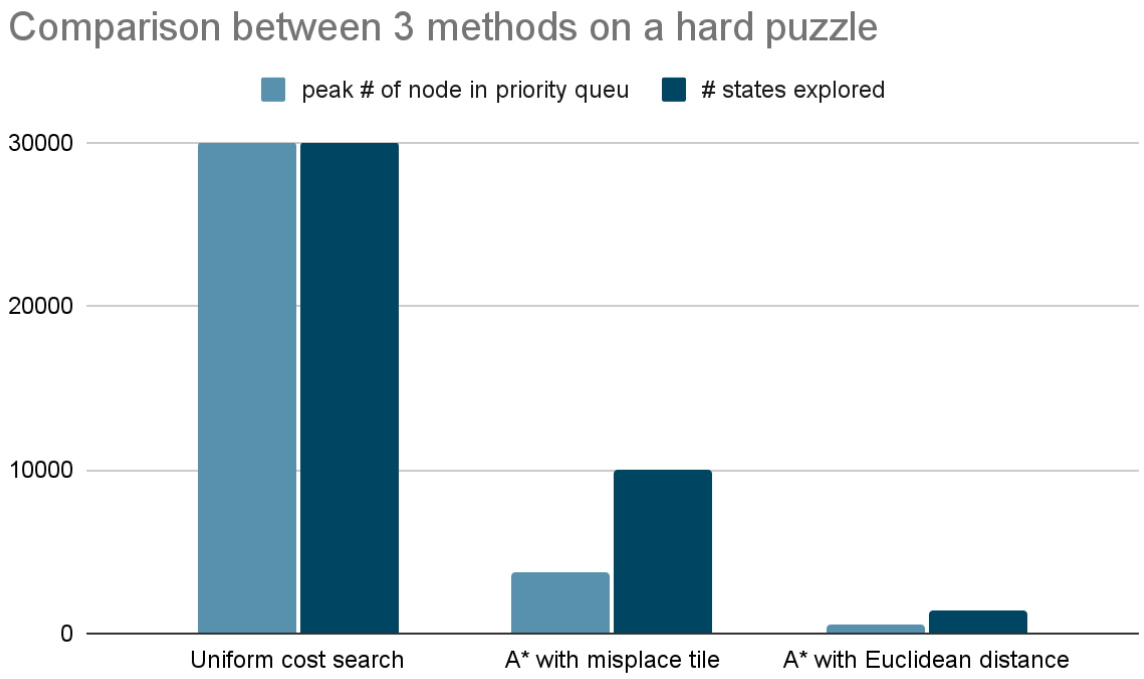
- A* with misplaced tile (10065 nodes explored, 3694 in queue at peak, 62.24 seconds runtime)



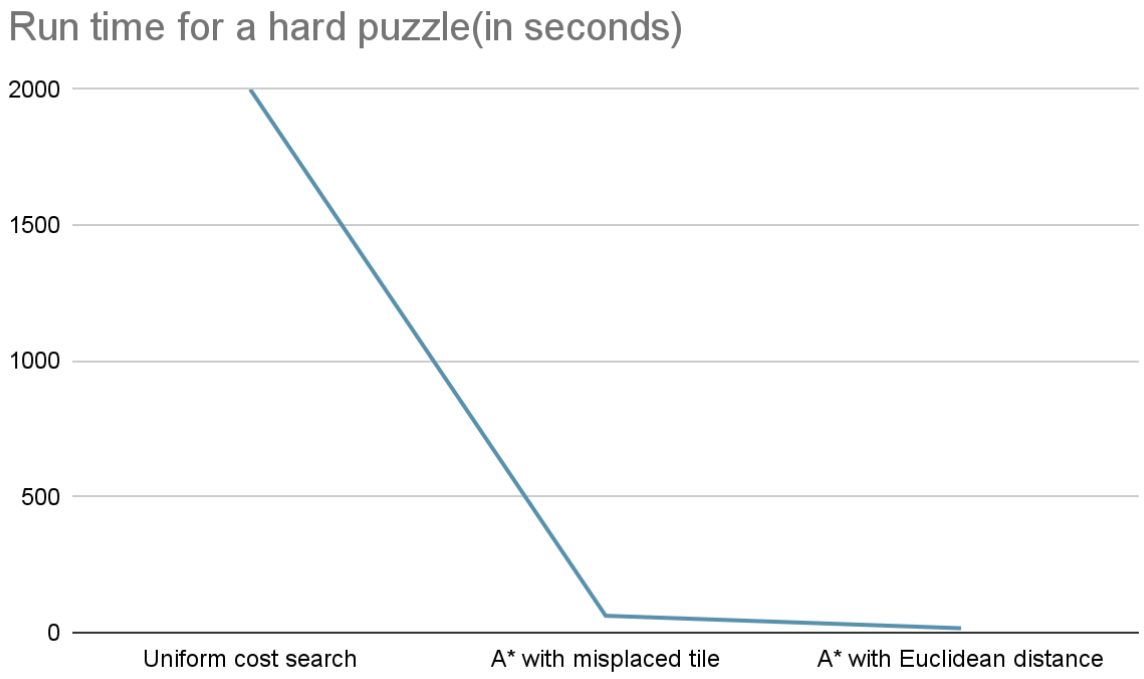
And then A* with Euclidean distance (1367 nodes explored, 493 at peak in queue, 16.44 seconds runtime.



(The runtime above of the actual solving is ~10 seconds for manual input of the puzzles' state as initial)

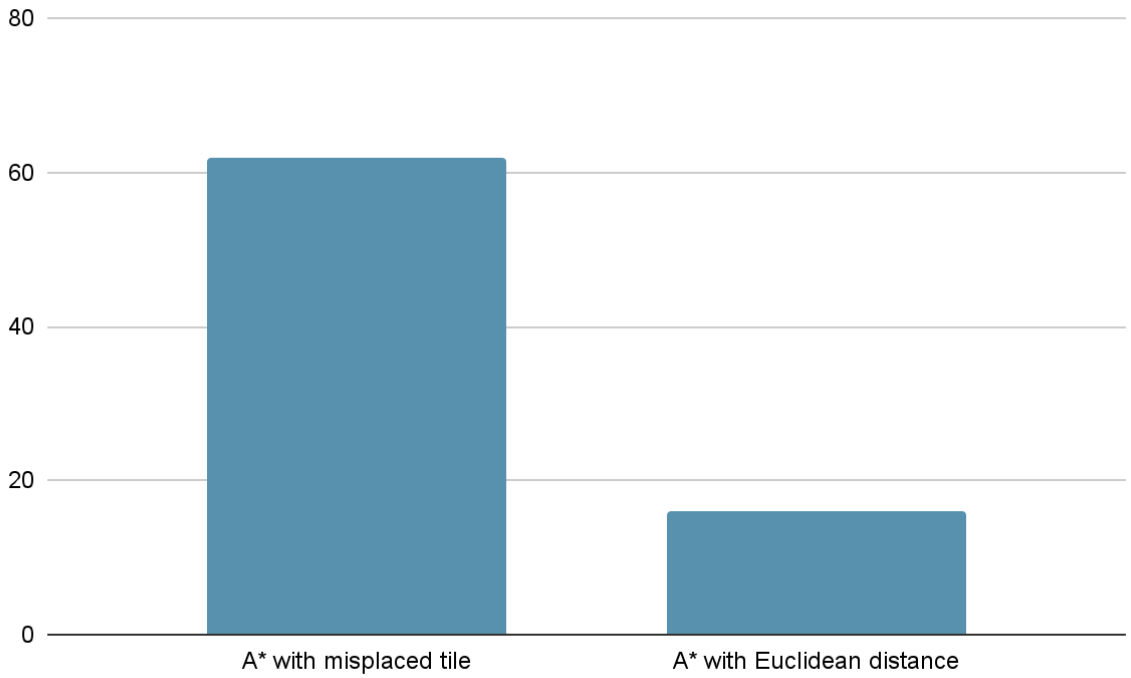


(In this graph, values of uniform cost search is NOT accurate but to showcase how much more time it takes to find the solution path)



(Same as last graph, time spent with uniform cost search is too long)

The graph below is the runtime difference between misplaced tile and euclidean distance. (Both runtime has included ~10 seconds of non-algorithmic runtime, sor it should be 50 seconds (misplaced), versus 6 seconds (euclidean)



Conclusion:

- For shallow problems, such as test cases trivial to medium, 3 methods have similar outcomes as well as runtime.

- As the solution path becomes longer, harder to find, both heuristics algorithm consider more than the depth of the tree search
    - Misplaced tiles prioritises node's states that have more correct tile positions
    - Euclidean distance prioritises the smallest total distance of misplaced tiles

- As a result, Euclidean distance A* search works more efficiently as the puzzle becomes harder since the results show that Euclidean distance A* search saves ~80% of the runtime that a weaker heuristic, misplaced tile A* search takes.