CS 170 Project 1

By: Ryan Giron (SID: 862128210)
and
Simraj Singh (SID: 862155747)

Challenges in developing the project

The challenges in this project did not come in the actual development of the algorithm. It actually came in trying to figure out how we wanted to implement the data structures and make our code organized to where it can be updated easily. We first weren't sure how we wanted to represent the states of the problem, but we ended up deciding to have each state be contained inside a node and build a tree data structure to have each node have a parent node and also four possible children nodes (since there are four possible operations we can perform). We also needed to overload the comparator for the priority queue because we need to store nodes in the queue based on their total costs.

Our Design

We first implemented a Problem class which contains the 6 initial test states, an initial state which we will set depending on the selection the user makes, a static goal state, and a current state. We also implemented a node class in which each node has a state of the problem, a level cost, a heuristic, the blank space's x and y coordinates, a parent pointer, and 4 children pointers. We have a constructor for the root node, and we also have constructors for leaf nodes which set their states, their parents, and what type of heuristic they will be using to calculate their total cost. We implemented our search algorithms in our main function along with how to calculate the different heuristic values. We also implemented a function to trace back to find the solution to the puzzle by giving you all the moves that are needed to be made to reach the goal state.

Description of Algorithms

1) Uniform Cost Search

- This algorithm is just A* with a heuristic value of 0 (h(n) = 0). This implementation is essentially Breadth First Search. Each state we expand will have a cost of 1, so our algorithm will depend entirely on g(n).

2) A* with Misplaced Tile heuristic

- This algorithm is A* with a heuristic that is calculated based on the number of misplaced tiles in the current state in comparison to the goal state. In other words, h(n) = the number of tiles NOT in their goal state position. Again, each state we expand will have a cost of 1.

3) A* with Euclidean Distance heuristic

- This algorithm is A* with a heuristic that is calculated based on two factors, the number of misplaced tiles AND the distance away each tile is from their goal state. In other words:

  h(n) = (the number of tiles not in their goal state position)
  +
  (the distance tiles are away from their goal state)

  Again each state we expand will have a cost of 1. This takes into account the cost of future state expansions to ultimately minimize the overall states needed to reach the goal state.

Data

States Expanded:	Sample Test Cases					
	1	2	3	4	5	6
Uniform Cost Search	1	4	4	29	85525	181440
A* Misplaced Tile Heuristic	1	4	4	7	51452	181440
A* Euclidean Distance Heuristic	1	4	3	7	3116	181440

Max nodes in queue:	Sample Test Cases					
	1	2	3	4	5	6
Uniform Cost Search	0	5	4	18	24969	24175
A* Misplaced Tile Heuristic	0	5	4	7	19526	23363
A* Euclidean Distance Heuristic	0	5	3	7	1981	22659

Figure 1.1: This table shows the run results of each of the three algorithms for all provided test cases. Shown is the data for states expanded and maximum nodes in the queue. Afterall, the only important comparison of these 3 algorithms is time (number of nodes expanded) and space (the maximum size of the queue).

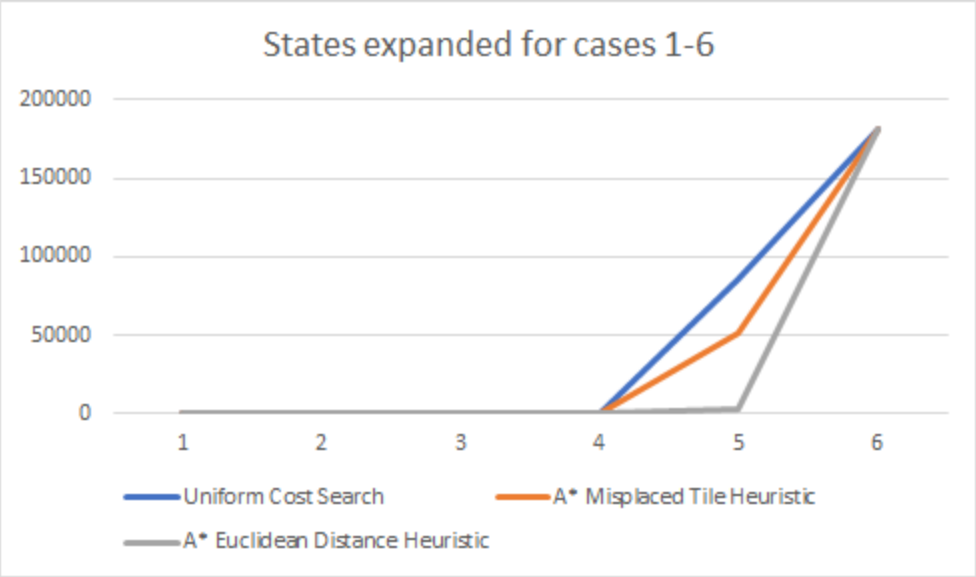

Figure 1.2: This graph shows the states expanded for each of the 3 algorithms across the 6 different test cases provided. The x-axis represents the provided test cases while the y-axis represents states expanded.

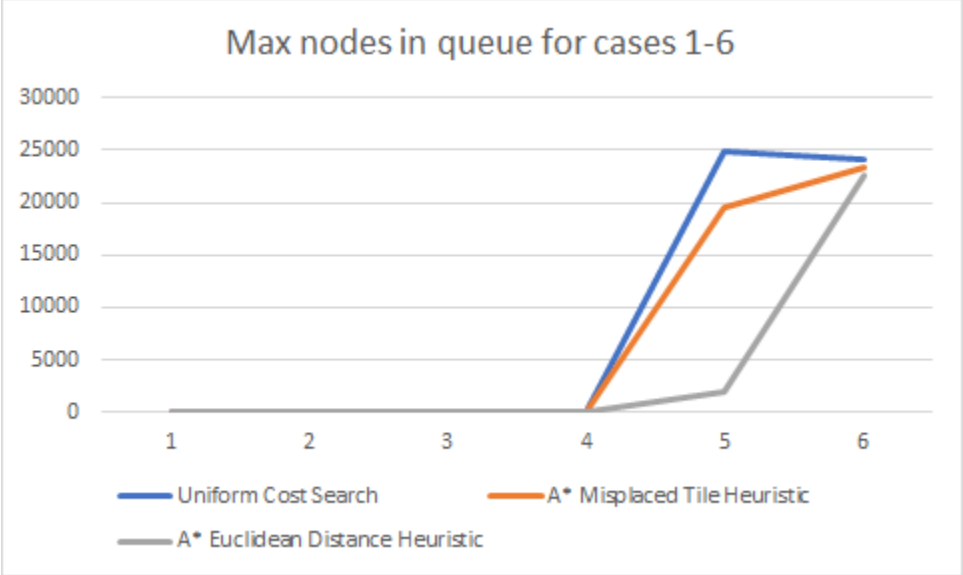

Figure 1.3: This graph shows the maximum nodes in the queue for each of the 3 algorithms across the 6 different test cases provided. The x-axis represents the provided test cases while the y-axis represents the maximum number of nodes in the queue.

Answer to #2

Here is the output of our code for the provided problem from the assignment page:

```
Please enter a number between 1 - 6 to select the difficulty that the AI should solve, or press 7 to make your own puzzle.
7
Please enter the first row: 103
Please enter the second row: 426
Please enter the third row: 758
Please enter the the number of the algorithm
1. Uniform Cost Search
2. A* Misplaced Tile Heuristic
3. A* Euclidean Distance Heuristic
3
```

```
INITIAL STATE
1 0 3
4 2 6
7 5 8
--------------------
1 2 3
4 0 6
7 5 8
--------------------
1 2 3
4 5 6
7 0 8
--------------------
1 2 3
0 4 6
7 5 8
--------------------
1 2 3
4 5 6
7 8 0
--------------------
```

```
STEPS TO GET TO SOLUTION:
1 2 3
4 0 6
7 5 8
---------------------------
1 2 3
4 5 6
7 0 8
---------------------------
1 2 3
4 5 6
7 8 0
---------------------------
```

```
To solve this problem the search algorithm expanded a total of: 5 nodes
The maximum number of nodes in the queue at any one time: 7
SUCCESS
```

Comparison of Algorithms

In conclusion, by testing the three different heuristic functions across the provided test cases of varying difficulty, we are able to achieve a good understanding of the efficiency of each of them. As we can see from the results, the Euclidean distance heuristic performed the best, followed by the Misplaced Tile heuristic, and lastly the Uniform Cost Search heuristic.

As seen in Figure 1.2 and Figure 1.3, the Euclidean heuristic computes the solution more efficiently in both time and space. However, they will all have the same Big O runtime as also shown in the case where the puzzle is impossible to solve (181440 nodes expanded). Their Big O is O(b^d) where b is the branching factor and d is the depth of the solution tree. They will also have the same Big O for the space complexity which is O(b^d).