Project 1 - 8 Puzzle Solver
Brittany Seto
ID 861058099
bseto001@ucr.edu
5-November-2015

CS170 Introduction to Artificial Intelligence
Instructor: Dr. Eamonn Keogh

In completing this project I consulted:

- Heuristic Search.ppt
- Python heapq: https://docs.python.org/2/library/heapq.html
- Python math: https://docs.python.org/2/library/math.html
- Python list: http://effbot.org/zone/python-list.htm
- Nilsson,N.J.: Principles of Artificial Intelligence Tioga Publishing Co. 1980 Pages 92-93, 96-102 (A* Algorithm and General Search Algorithm)

A trace of the Manhattan distance A*:

```
ucrwpa3-3-216-131:EightPuzzleSolver BrittanySeto$ python 8PuzzleSolver.py
Welcome to Brittany's 8 puzzle solver!
Type "1" to use a default puzzle, or "2" to enter your own puzzle: 1

1. Uniform Cost Search
2. A* with the Misplaced Tile Heuristic
3. A* with the Manhattan distance heuristic

Enter your choice of algorithm: 3
A* with the Manhattan distance heuristic
1 2 3
4 0 6
7 5 8

Best state to expand with a g(n) = 1 and h(n) =  1 (expanded: 1 )
1 2 3
4 5 6
7 0 8

Best state to expand with a g(n) = 2 and h(n) =  0 (expanded: 2 )
1 2 3
4 5 6
7 8 0
GOAL!
Max queue size = 5
```

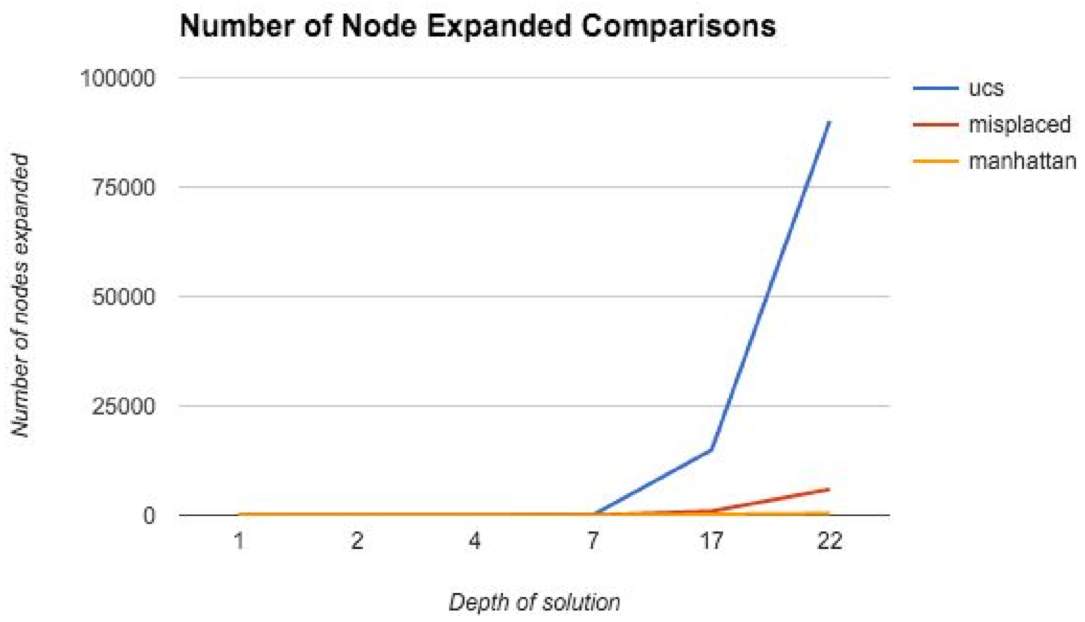
Report

By implementing all three different searches, Uniform Cost search, and A* with two different heuristics, the Eight puzzle can be solved.

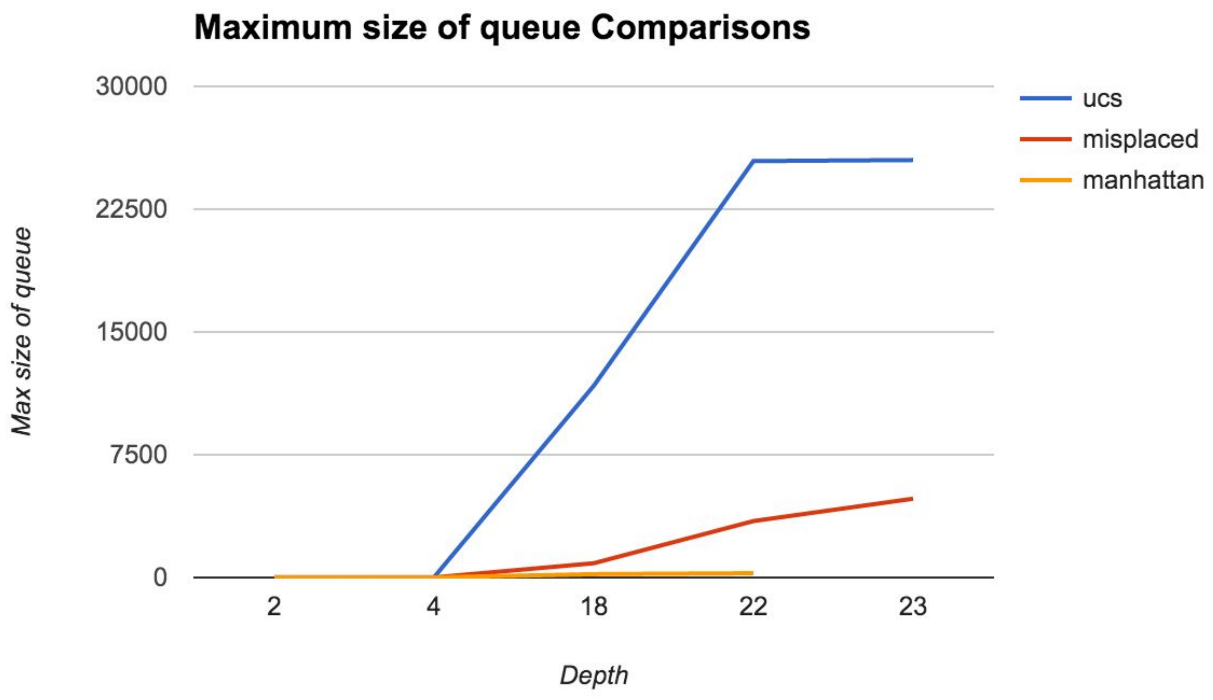
Implementation:
For the search algorithm, I mainly followed the General Search Algorithm. I passed in the heuristic function according to which search the user inputs to calculate the h(n).

- Uniform Search Algorithm: I implemented this search by creating a function that returns zero each time it is called.
- A* with the Misplaced Tile heuristic: To implement this heuristic, I went through the current state's list and compared it to the goal state. If the tile at the same indexes do not match up, then the counter for the number of misplaced tiles is incremented.
- A* with the Manhattan Distance heuristic: To implement this heuristic, I put both the current state and the goal state into a matrix. This was simpler for me to calculate the Manhattan distance. After creating the two matrixes, I compared the indexes on each matrix. If the current state's value does not match the goal state's value, it finds the indexes of where the value is supposed to be. It saves the index values into a variable and I calculated h(n) by taking the absolute value of the difference of the x-coordinates and added it to the absolute value of the difference of the y-coordinates for each misplaced tile.

Findings:



	1	2	4	7	17	22
ucs	3	9	21	124	14925	90130
misplaced	1	2	4	12	925	5939
manhattan	1	2	4	7	211	431



	2	4	18	22	23
ucs	8	16	11711	25439	25502
misplaced	5	4	872	3448	4814
manhattan	5	4	204	262	574

From the data collected for the number of nodes expanded, the most efficient search algorithm is the A* with the Manhattan Distance heuristic. The worst algorithm between the three is Uniform Cost Search because it is only based on the weighted depth. A* is optimally efficient for any given heuristic, which means no other optimal algorithm is guaranteed to expand fewer nodes than A*. For the A* algorithm, the depth and the heuristic are taken into account for the weight. This gives the search algorithm a more accurate way of figuring out which node to expand.

Using a heuristic function with higher values is always better as long as it does not overestimate. This is shown by the graph above. A* using the Manhattan distance will expand fewer nodes on average than using A* with the Misplaced tile heuristic. This is because the Manhattan distance heuristic is admissible. It is admissible because any move can only move one tile one step closer to the goal.