Project 1

CS 205. Artificial Intelligence

Instructor: - Dr. Eamonn Keogh

Pratheek Chindodi Rajashekar
SID 862002345
pchin006@ucr.edu
1-March-2018

In completing this project, I consulted. . .

- Rohith Mohan, Graduate Quantitative Methods Center, UCR to understand data frames and plotting graphs in Python.
- Heuristic Search lecture slides to understand the theory of heuristics.
- Stuart J. Russell and Peter Norvig: Artificial Intelligence A Modern Approach. Third Edition
- http://artificialintelligence-notes.blogspot.com/2010/07/8-puzzle-problem.html
- https://www.geeksforgeeks.org/check-instance-8-puzzle-solvable/
- To understand the pseudo-code for the 8-puzzle Implementation - https://gist.github.com/thiagopnts/8015876
- https://algorithmsinsight.wordpress.com/graph-theory-2/a-star-in-general/ to understand the pseudo-code for A star algorithm
- CS170 Project Sample Report, Lorem Ipsum to understand the contents of the report
- https://www.geeksforgeeks.org/graph-plotting-in-python-set-1/ for plotting in python
- All subroutines used from heapq, to use functions such as pop and push.
- All subroutines used from matplotlib, to plot the graphs

CS205: Project1 - Summary

Pratheek Chindodi Rajashekar, SID 862002345

Introduction

The Eight Puzzle project gives a fine analysis and comparison of search algorithms that can be implemented to attain the goal state with the best possible moves. A set of puzzles is tested with search algorithms such as Uniform Cost Search, A Star algorithm with Misplaced Tile Heuristic and A Star with Manhattan Distance Heuristic. The analysis is done by the visualization of the different properties of the puzzle such as depth, time and number of states expanded for each puzzle. The first plot gives information about how the depth of the puzzle and time taken for a specific heuristic function are related. The second plot details the number of states expanded for each puzzle set. I implemented the project using Python. I used matplotlib library to plot the graphs.

Algorithms

The algorithms that are used are almost similar. The only difference is with the heuristic values. Different algorithms have different heuristic values.

Uniform Cost Search

This algorithm is very similar to the breadth first search algorithm. The uniform cost search works well for test cases such as Trivial, Very Easy, Easy and Doable. As the difficulty level of the eight puzzle set increases, the number of states increases rapidly for Uniform Cost Search. It takes more states to solve a given problem and hence the space complexity and time complexity increases for Uniform Cost Search algorithm. Also, the time complexity of the algorithm increases as it takes more time to run. Uniform Cost Search is A Star algorithm with h(n) value assigned to zero. Apart from the heuristic, the working of the algorithm is very much similar to that of A Star algorithm. It has a list to store the values of the open list, that is the states or nodes yet to be visited. Also, a closed list which keeps track of the nodes which are already visited/explored.

Misplaced Tile Heuristic

This heuristic function calculates the total number of tiles that are misplaced in the eight puzzle. Suppose, if there are 5 tiles which are not in their correct position or goal state, then the heuristic value for that particular puzzle set is five. Similarly, for a difficult puzzle such as “Oh Boy “, the heuristic value of the eight-puzzle set is nine. For impossible puzzle set, though the heuristic value is just 2, it is not solvable. Since only one inversion is there for the impossible puzzle set, it can never be solved. Similarly, any puzzle, which has odd number of inversions cannot be solved. For test cases such as

Trivial and Very Easy, the number of states expanded by A Star with Misplaced Tile Heuristic is more or less exact to the Uniform Cost Search. Only when
the difficulty level raises to cases such as Doable and Oh Boy, there is a significant decrease in the number of steps needed to reach the goal state.

Manhattan Distance Heuristic

In A Star with Manhattan Distance heuristic, each element of a test case is compared with the Goal State. Suppose the test case is as given below: -

Oh Boy	Goal State
8 7 1	1 2 3
6 0 2	4 5 6
5 4 3	7 8 0

All of the numbers in the initial state, which is Oh Boy is not in the goal state. Hence the distance is calculated for each number by computing the distance to its correct position in the goal state. Finally, the sum of all these distances add up to Manhattan Distance.

Element	Manhattan Distance (Initially 0)
8	3
7	3
1	2
6	2
2	2
5	2
4	2
3	2
Total Manhattan Distance =	**18**

From Figure 1, it can be observed that as the difficulty level of the puzzle set increases, the depth of A Star algorithm with Manhattan Distance Heuristic increases. Only the first 5 puzzles in Table 1, have the same depth for all the algorithms. Furthermore, at higher difficulty levels, A Star algorithm with Manhattan Distance Heuristic takes lesser time to reach the goal state from the initial state of the puzzle set. From Table 2, it is clearly visible that the number of nodes expanded for Manhattan Distance Heuristic is lesser than the other two algorithms. This justifies that A Star with Manhattan Distance Heuristic gives an optimal solution at higher difficulties of the puzzle sets.

Data Sets

In addition to the five test cases of puzzles that were given, four more test cases were considered to perform an analysis on the different algorithms. These four puzzles are given as follows:-

Puzzle1 = [3, 4, 0, 6, 1, 8, 2, 5, 7]

3 4 0
6 1 8
2 5 7

Puzzle2 = [6, 0, 3, 2, 1, 8, 2, 5, 7]

6 0 3
2 1 8
2 5 7

Puzzle7 = [6, 5, 4, 8, 0, 7, 3, 2, 1]

6 5 4
8 0 7
3 2 1

Puzzle5 = [8, 6, 7, 2, 5, 4, 3, 0, 1]

8 6 7
2 5 4
3 0 1

Table 1 :- Comparison of Heuristics

Difficulty	Misplaced Tile Heuristic		Manhattan Distance Heuristic	
	Depth	Time(seconds)	Depth	Time(seconds)
Trivial	0	0	0	0
Very Easy	1	0.000153064	1	0.00016903
Easy	2	0.00021505	2	0.0003390
Doable	4	0.00036692	4	0.0003910064
Puzzle2	15	0.5923	15	0.0465
Oh Boy	22	27.0492	24	4.802
Puzzle1	26	63.47	28	40.5473
Puzzle7	28	80.4105	30	72.76739
Puzzle5	31	85.5456	33	78.6205

Figure 1 : Comparison of Misplaced Tile Heuristic and Manhattan Distance Heuristic

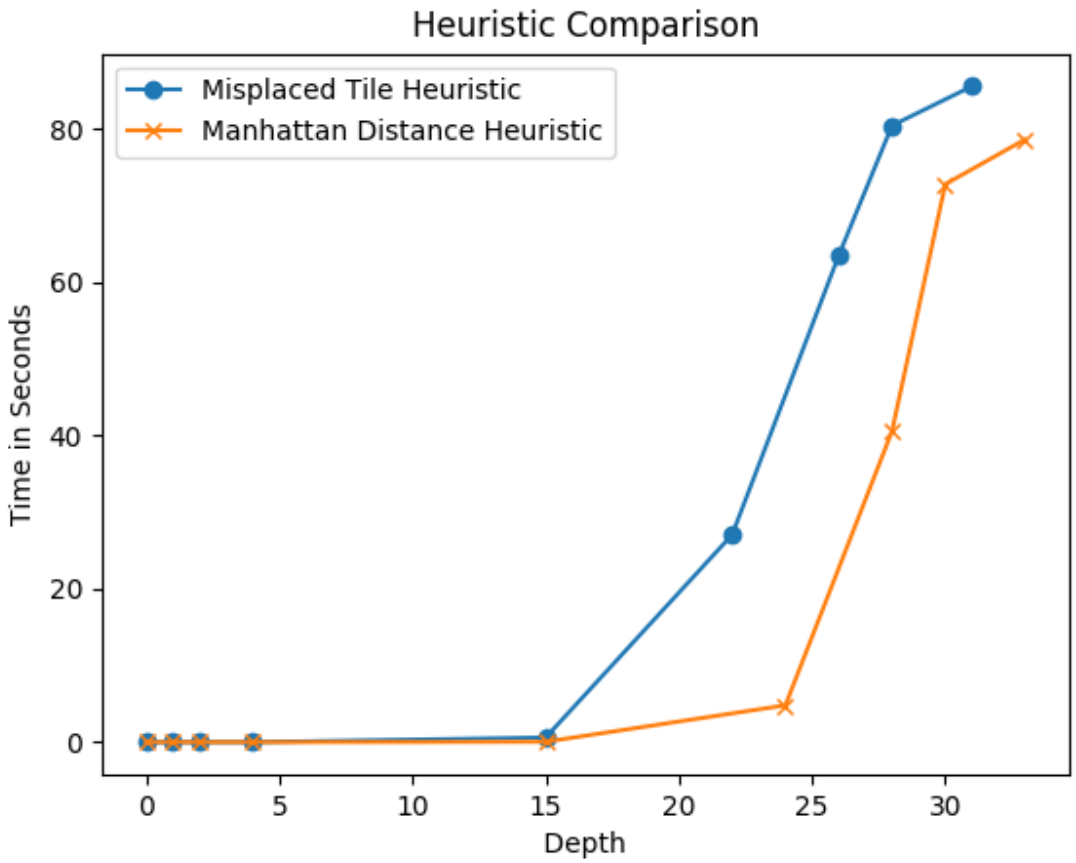


Table2 : - Number of Nodes Expanded for each puzzle

Difficulty	Uniform Cost Search	Misplaced Tile Heuristic	Manhattan Distance Heuristic
Trivial	0	0	0
Very Easy	7	3	3
Easy	13	4	6
Doable	33	7	7
Puzzle2	11599	8777	658
Oh Boy	227572	230534	48946
Puzzle1	445245	444500	287359
Puzzle7	513612	487136	445046
Puzzle5	519218	497063	453334

Figure 2 :- Number of Nodes Expanded for each puzzle

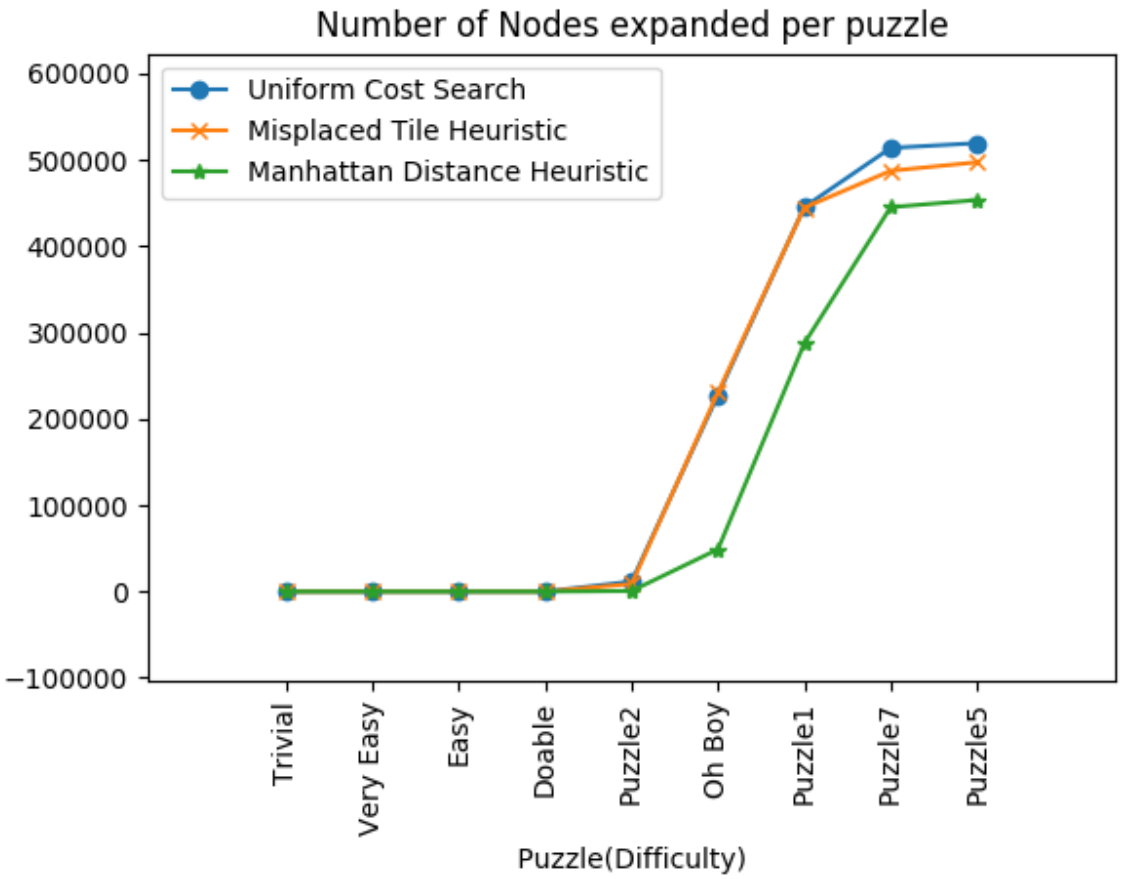
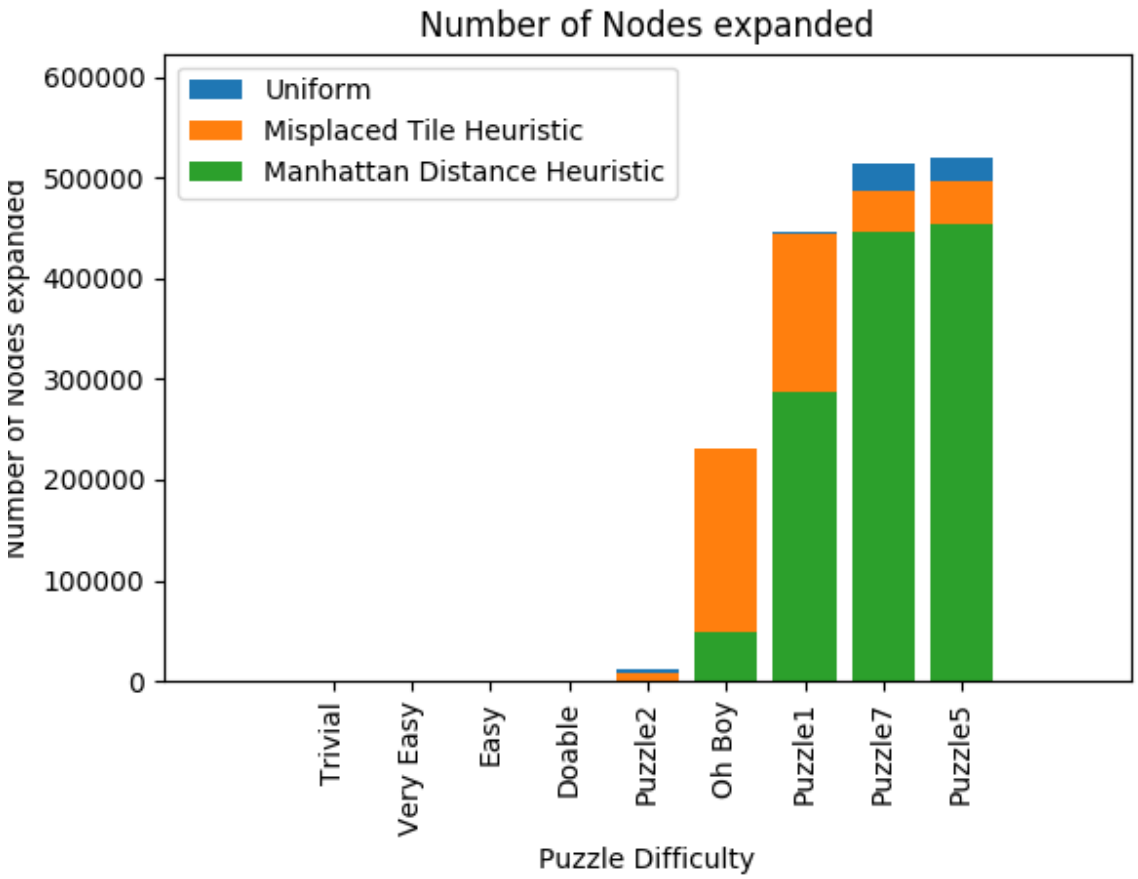


Figure 3: - Bar Graph of the Number of Nodes Expanded per each puzzle



Conclusion:

- For most of the test cases in Uniform cost search, the number of states expanded is greater than that of A Star with Misplaced Tile and Manhattan Distance Heuristics. As the difficulty of the puzzle increases, the space complexity and time complexity of Uniform Cost Search increases.

- A Star algorithm with Misplaced Tile Heuristic performs better than Uniform Cost Search as the difficulty of the puzzle increases. When the difficulty is comparatively lower, the number of states expanded for Misplaced Tile Heuristic is almost equal to that of Uniform Cost Search.

- A Star with Manhattan Distance Heuristic performs faster and better than other algorithms as the difficulty of the puzzle set increases. Manhattan Distance Heuristic is better in terms of both Space and Time Complexity. A Star with Manhattan Distance Heuristic provides an optimal solution as the difficulty of the puzzle set increases.

Output of 8 Puzzle Solver

```
/Users/pratheek/PycharmProjects/p1/venv/bin/python /Users/pratheek/PycharmProjects
Welcome to CS205 8 puzzle
Enter 1 to use a default puzzle
Enter 2 to specify your own puzzle
Select your choice 1
Select the level of difficulty
1. Trivial
2. Very Easy
3. Easy
4. Doable
5. Oh Boy
6. Impossible
3
Your choice is Easy
1. Uniform Cost Search
2. A Star with Misplaced Tile Heuristic
3. A Star with Manhattan Distance Heuristic
Select your choice of algorithm 3

Initial State:
1 2 0
4 5 3
7 8 6


The given puzzle is solvable
1 2 3
4 5 0
7 8 6

1 2 3
4 5 6
7 8 0

Goal State is achieved


Depth is 2
Time taken is 0.000303983688354 seconds
Total number of states expanded = 6

Process finished with exit code 0
```

Source Code: -

```
from heapq import heappush, heappop
import time
from math import*


class Search:
    def __init__(self, initial_state=None):
        self.initial_state = Node(initial_state)
        self.goal = [1, 2, 3, 4, 5, 6, 7, 8, 0]

    def uniform_cost(self):
        return self.cost()

    def misplaced(self):
        return sum([1 if self.numbers[i] != self.goal[i] else 0 for i in xrange(8)])

    def man_dist(self):
        goal_state =[1, 2, 3, 4, 5, 6, 7, 8, 0]
        return sum(abs(a - b) for a, b in zip(self.numbers, goal_state))

    def reconstruct_path(self, end):
        path = [end]
        state = end.parent
        while state.parent:
            path.append(state)
            state = state.parent
        return path

    def search(self, heuristic):
        # Initialize Open List to store the nodes yet to be explored
        open = Queue()
        # Add the initial state to the Open List
        open.add(self.initial_state)
        # Initialize the closed list which stores the nodes already visited
        closed = set()
        moves = 0
        print ('\nInitial State:')
        print (open.retrieve())
        print('\n')
        start = time.time()
        overall_count = 0
        #Until the open list has nodes to be explored
        while open:
            # Retrieve the head of the open list
            current = open.head()
            if current.numbers[:-1] == self.goal:
                # Start timer if the initial state has not yet reached the goal state
                end = time.time()
                print('The given puzzle is solvable')
                path = self.reconstruct_path(current)
                # Path is reversed to obtain the path from initial state
                for state in reversed(path):
                    print(state)
                    print
                print("Goal State is achieved \n \n ")
                print ('Depth is ' + str(len(path)))
                print ('Time taken is ' + str(end - start) +' seconds' )
```

```
def main():
    print("Welcome to CS205 8 puzzle")
    print("Enter 1 to use a default puzzle")
    print("Enter 2 to specify your own puzzle")
    x = int(input('Select your choice'))

    if x==1:
        print('Select the level of difficulty')
        print('1. Trivial\n2. Very Easy ')
        print('3. Easy\n4. Doable')
        print('5. Oh Boy\n6. Impossible')
        y = int(input(''))
        if y==1:
            print('Your choice is Trivial')
            puzzle = [4, 1, 2, 7, 0, 3, 8, 5, 6]
        elif y==2:
            print('Your choice is Very Easy')
            puzzle = [1, 2, 3, 4, 5, 6, 7, 0, 8]
        elif y==3:
            print('Your choice is Easy')
            puzzle = [1, 2, 0, 4, 5, 3, 7, 8, 6]
        elif y==4:
            print('Your choice is Doable')
            puzzle = [0, 1, 2, 4, 5, 3, 7, 8, 6]
        elif y==5:
            print('Your choice is Oh Boy')
            puzzle = [8, 7, 1, 6, 0, 2, 5, 4, 3]
        elif y==6:
            print('The number of inversions are odd')
            print('The problem can not be solved')
            exit(0)
        else:
            print('Wrong choice')
    elif x==2:
        puzzle = []
        i=0
        while len(puzzle) < int(9):
            i+=1
            item = int(input('Enter the values of element %d:'%i))
            puzzle.append(item)
            print(puzzle)
    else :
        print('Invalid choice')
        exit(0)

    print('1. Uniform Cost Search')
    print('2. A Star with Misplaced Tile Heuristic')
    print('3. A Star with Manhattan Distance Heuristic')
    algo = int(input('Select your choice of algorithm'))

    if algo == 1:
        solver = Node(puzzle)
        a = solver.search(1)
    if algo == 2:
        solver = Node(puzzle)
        a = solver.search(2)
    if algo == 3:
        solver = Node(puzzle)
        a = solver.search(3)
print('Total number of states expanded = ' + str(a))
```