Project 1 CS 205. Artificial Intelligence

Kamalika Poddar Instructor: Dr. Eamonn Keogh

kpodd001@ucr.edu
21-February-2018

In completing this homework, I consulted…

- https://www.youtube.com/watch?v=dRMvK76xQJI (for Uniform Cost Search)
- https://www.youtube.com/watch?v=6TsL96NAZCo (for A* Search)
- http://code.activestate.com/recipes/499304-hamming-distance/ (misplaced tiles)
- https://stackoverflow.com/questions/12526792/manhattan-distance-in-a (manhattan distance)
- https://gist.github.com/flatline/838202/ca0d35b0ce7e5d9ec8-6b77b0490baba4cda87980 (I referred to this link for a-star algorithm)
- I also referred to the project sample report for making this report.

CS-205/ ARTIFICIAL INTELLIGENCE

PROJECT REPORT

INTRODUCTION

This is my first project under Dr. Eamonn Keogh for CS 205- Artificial Intelligence. I used Python as the programming language to complete the project. In this project, I tried to compare algorithms like uniform cost search, A-star with Misplaced Tile heuristic and A-star with Manhattan distance Heuristic. Below is the detailed report of the work done for this project.

UNIFORM COST SEARCH ALGORIHTM

In this type of an algorithm, we find the goal state by calculating the cost of each path and selecting the path which reaches goal with minimum cost. Heuristic is set to zero for uniform cost search. Uniform cost search is also referred to as Breadth First Search.

A (*) STAR ALGORITHM

A-Star reaches the goal state by summing up both the cost function and the heuristic. A-Star is considered to be a far more efficient algorithm than uniform cost search. For the 8-Puzzle problem we can solve it by Misplaced Tile Heuristic or by calculating Manhattan Distance.

A* with Misplaced Tile Heuristic

This algorithm works same as calculating Hamming Distance. The numbers which are not in the exact same state as that of the goal state are assigned to be 1 and the numbers which matches the goal state are assigned as 0. The total of all the numbers which are misplaced is referred to as Misplaced Tile Heuristic.

A* with Manhattan Distance Heuristic

For a 8-Puzzle problem, the Manhattan Distance is defined as the difference between the two co-ordinates. The summation of the distance between the number of misplaced tiles with their goal states is referred to as Manhattan Distance.

RESULTS

- Below is the screenshot showing the output results when the given the input is Doable={[0,1,2,],[4,5,3],[7,8,6]} and the type of algorithm performed is A* with Manhattan Distance. The number of nodes expanded are 25 and time elapsed is 2.65 milliseconds.

```
2/20/2018 22:34:21
Welcome to 8-Puzzle
Please enter your choice of puzzle
1. Trivial
2. Very_Easy
3. Easy
4. Doable
5. Oh_boy
6. Impossible
7. any type of combination you want to add
Enter your Choice -> 4
4
0 1 2
4 5 3
7 8 6

Three types of algorithm are:
 1. Uniform Cost Search
 2. Misplaced Tile Heuristic
 3. Manhattan Distance
Enter the type of algorithm to perform3
1 0 2
4 5 3
7 8 6

1 2 0
4 5 3
7 8 6

1 2 3
4 5 0
7 8 6

1 2 3
4 5 6
7 8 0

A Star with Manhattan distance Heuristic 25 states

time elapsed 0.00265097618103 seconds

Process finished with exit code 0
```

- Below is the screen shot of the results where a user inputs array: {[1,2,3], [4,0,8], [7,6,5]} and the output shows 47 nodes expanded using Manhattan Distance Heuristic.

```
2/20/2018 22:36:7
Welcome to 8-Puzzle
Please enter your choice of puzzle
1. Trivial
2. Very_Easy
3. Easy
4. Doable
5. Oh_boy
6. Impossible
7. any type of combination you want to add
Enter your Choice -> 7
7
Enter the type of a random puzzle to be solved
num:1
num:2
num:3
num:4
num:0
num:8
num:7
num:6
num:5
Array:  [1, 2, 3, 4, 0, 8, 7, 6, 5]
1 2 3
4 0 8
7 6 5

Three types of algorithm are:
 1. Uniform Cost Search
 2. Misplaced Tile Heuristic
 3. Manhattan Distance
Enter the type of algorithm to perform3
1 2 3
4 6 8
7 0 5

1 2 3
4 6 8
7 5 0

1 2 3
4 6 0
7 5 8

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

A Star with Manhattan distance Heuristic 47 states

time elapsed 0.00715398788452 seconds

Process finished with exit code 0
```

- Below is the table showing findings of nodes expanded by the three algorithms and amount of time elapsed.
  The puzzles taken as are:
  Puzzle1= {[1,2,3],
  [4,0,8],
  [7,6,5]}

  Puzzle 2 = {[1,2,0],
  [4,5,3],
  [7,8,6]}

Puzzle	Uniform Cost Search		Misplaced Tiles		Manhattan Distance	
	Time	Nodes	Time	Nodes	Time	Nodes
Very_easy	0.2081 ms	4	0.00067ns	4	0.000674ns	4
Easy	0.3575 ns	7	0.0013 ns	7	0.651 ms	4
Puzzle 2	9.4 ms	15	3.95 ns	19	1.765 ms	10
Doable	2.354 ms	31	0.00701 ns	31	5.98 ms	25
Puzzle 1	20.42 ms	115	39.20 ns	125	16.53 ms	47

CONCLUSION

- According to the results above we observe that Misplaced tiles heuristic takes the least amount time in expanding nodes and it expands the highest number of nodes.
- Uniform cost search has greater time complexity than the other two algorithms.
- With the help of this project I could learn a lot about programming. This is my first programming project. I admit that I am not well versed with programming but I have given my heart and hard work in doing this project.

ABOUT THE CODE

The below code consists of 269 lines including whitespaces and comments. Firstly, a function is defined for Manhattan Distance and Misplaced Tile Heuristic and a third function is defined for uniform cost search where heuristic is set to zero. A main() function contains the input to the default puzzle and assigns it to the respective search to be performed. A class “Node” is then declared to set the features of the node. A method “astar” contains the expanding states of the algorithm and the output state is printed in the main function.

First page of the code as instructed by the instructor

```
import time
from datetime import datetime

now = datetime.now()
print '%s/%s/%s %s:%s:%s' % (now.month, now.day, now.year, now.hour, now.minute,
now.second)

#declare goal as a global variable
goal = [[1, 2, 3],[4, 5, 6],[7, 8, 0]]
#declaring range as three
n=3

#function to define Manhattan Distance Heuristic
def man_dist(puzzle):
    mand = 0
    for i in range(n):
        for j in range(n):
            count = puzzle.retrive(i, j) - 1
            if (count == 0):
                continue
            x = abs((count-1)/n)
            y = (count - 1) % n
            mand = mand + abs(x - i) + abs(y - j)
            #print "The Manhattan Distance is" + str(mand)
    return mand

#function to define Misplaced Tile Heuristic
def misplaced_tile(puzzle):
    mispl = 0
    for i in range(n):
        for j in range(n):
            if (puzzle.retrive(i, j)-1)!= (goal[i][j]):
                mispl += 1
                #print "The Misplaced Tiles are" + str(mispl)
    return mispl

#function to define Uniform cost search where heuristic is equal to zero
def ucst(puzzle):
    puzzle=puzzle
    return 0

#define a class which describes different features of the Node
class Node:

    def __init__(self,h_score=0, depth=0, parent= None):
        self.h_score = h_score
        self.depth = depth
        self.parent = parent
        self.state = []
        for i in range(n):
            self.state.append(goal[i][:])

    def astar(self, heuristic):

        def solvable(puzzle):
            if puzzle.state == goal:
continue. . . .
```

Last page of the code as instructed by the instructor

Continue...

```
    else:
        #add the elements in an array to design your own puzzle
        array= list()
        print ('Enter the type of a random puzzle to be solved')
        for i in range(9):
            n= input("number:")
            array.append(int(n))
        print 'Array:'+ str(array)
        inputVaribale= array

    #take in the indices of the rows and column of the matrix
    locations1 = [0, 0, 0, 1, 1, 1, 2, 2, 2]
    locations2 = [0, 1, 2, 0, 1, 2, 0, 1, 2]

    #assign the values taken from the input array to the indices of the array
    for i in range(9):
        p.assign(locations1[i], locations2[i], inputVaribale[i])
    print p

    print "Three types of algorithm are:\n 1. Uniform Cost Search \n 2. Misplaced Tile
Heuristic\n 3. Manhattan Distance"
    a = input('Enter the type of algorithm to perform')
    # start the timer
    start = time.time()

    if (a==1):
        path, numbers = p.astar(ucst)
        path.reverse()
        for i in path:
            print i
        print "Uniform Cost Search takes " + str(numbers)+ " states"

    if (a== 2):
       path,numbers = p.astar(misplaced_tile)
       path.reverse()
       for i in path:
          print i
       print "A Star with Mispalced Tile Heuristic " + str(numbers) +" states"

    if (a ==3):
        path, numbers = p.astar(man_dist)
        path.reverse()
        for i in path:
            print i
        print "A Star with Manhattan distance Heuristic " + str(numbers)+ " states"

    # end the timer
    end = time.time()
    # calculate the time elapsed and print it
    elapsed = end - start
    print "\n time elapsed " + str(elapsed) + " seconds"

if __name__ == "__main__":
    main()
```