Programming Assignment # 2

Vasanth Reddy Baddam

04/27/2020

I pledge that this test/assignment has been completed in compliance with the Graduate Honor Code and that I have neither given nor received any unauthorized aid on this test/assignment
Name: Vasanth Reddy Baddam
Signature: VB

---

1 Problem Description/Input Description/Output Description

A maze is a path or collection of paths, typically from an entrance to a goal. The main objective for any solver is to get to the terminal state/goal state from a given point. Maze solving is the act of finding a route through the maze from the start to finish. Some maze solving methods are designed to be used inside the maze by a traveler with no prior knowledge of the maze, whereas others are designed to be used by a person or computer program that can see the whole maze at once.

1.1 Input

The input to this would be the maze co-ordinates, i.e. the wall locations, trap location and other important location co-ordinates for an agent to look up. Below are the important inputs that take ti generate a random maze. Sample maze is given in figure 1

- Maze Size: This gives the dimension of the maze. Contains in the format of rows and columns in a given matrix
- Wall Locations: Walls are the one which separates the grids from one another
- Trap Locations: Both the wall and trap locations are randomly generated and are randomly distributed throughout the maze. Solver will be penalised for getting into trap which would ultimately result in increase of cost.
- Goal Location: This is where an agent have to reach to complete it's task. There would be only one goal location in the maze but we can change the number of goal locations as per our wish.

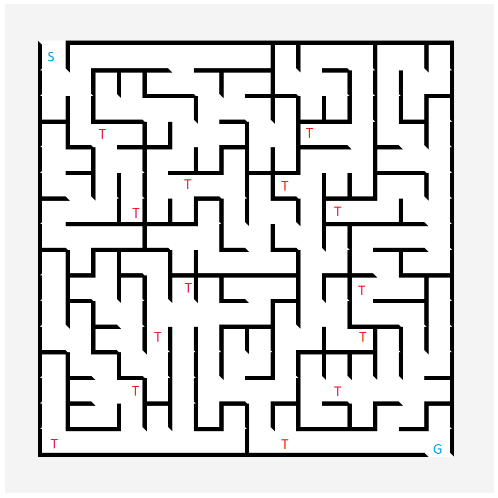


Figure 1: Maze sample

- Start Location: This is where our agent starts to search for its terminal point. Unlike, goal positions there would be only one start location.

1.2 Input designs

There is some cost induced into each step of an agent. We generally place some cost into each step it takes and cost for each time it go into traps.

- Cost for each step it takes would be +1
- Cost for each time it goes into a trap location would be +10

1.3 Objective function

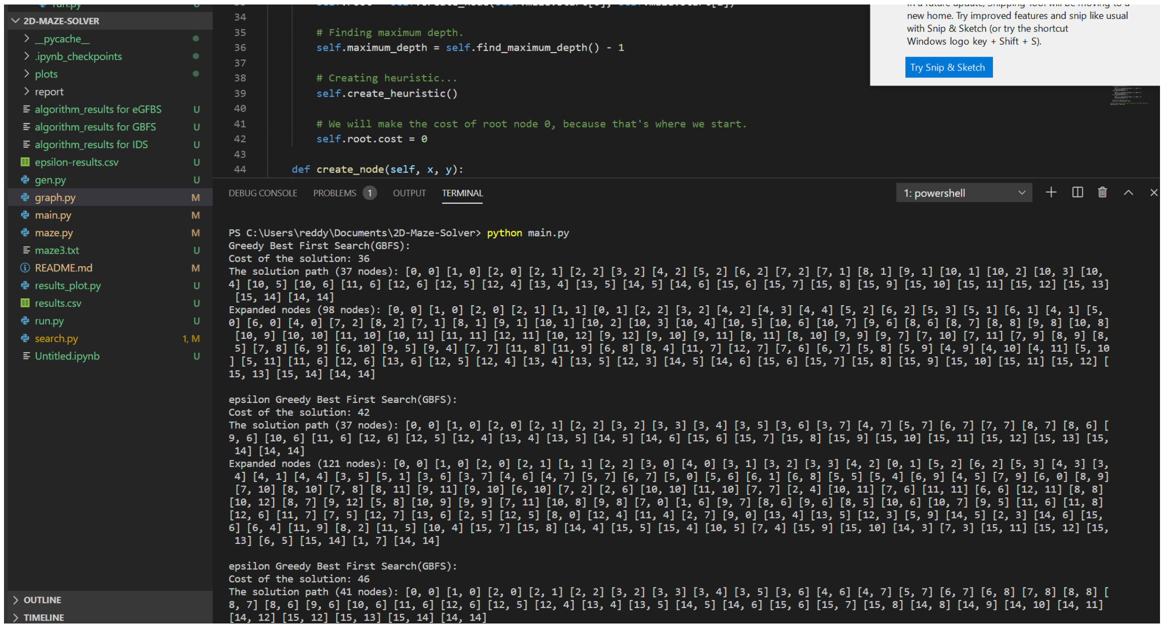
Maze has different solutions and can have n number of paths to reach the goal state. Sometimes it would take more cost to reach the goal state. So, the main objective of our problem is to minimize the total cost to reach a goal state. The target of an agent would be to obtain the optimal cost and reach the goal state.

1.4 Outputs

Expected outputs would be in the given format as shown in the figure 2

1.5 Constraints

As the maze size increases, the number of nodes to visit increases drastically which would make for any naive algorithm to go to all the nodes. As the number of nodes to expand increases, naive agent



```
        # Finding maximum depth.
        self.maximum_depth = self.find_maximum_depth() - 1

        # Creating heuristic...
        self.create_heuristic()

        # We will make the cost of root node 0, because that's where we start.
        self.root.cost = 0

    def create_node(self, x, y):
```

```
PS C:\Users\reddy\Documents\2D-Maze-Solver> python main.py
Greedy Best First Search(GBFS):
Cost of the solution: 36
The solution path (37 nodes): [0, 0] [1, 0] [2, 0] [2, 1] [2, 2] [3, 2] [4, 2] [5, 2] [6, 2] [7, 2] [7, 1] [8, 1] [9, 1] [10, 1] [10, 2] [10, 3] [10,
4] [10, 5] [10, 6] [11, 6] [12, 6] [12, 5] [12, 4] [13, 4] [13, 5] [14, 5] [14, 6] [15, 6] [15, 7] [15, 8] [15, 9] [15, 10] [15, 11] [15, 12] [15, 13]
 [15, 14] [14, 14]
Expanded nodes (98 nodes): [0, 0] [1, 0] [2, 0] [2, 1] [1, 1] [0, 1] [2, 2] [3, 2] [4, 2] [4, 3] [4, 4] [5, 2] [6, 2] [5, 3] [5, 1] [6, 1] [4, 1] [5,
0] [6, 0] [4, 0] [7, 2] [8, 2] [7, 1] [8, 1] [9, 1] [10, 1] [10, 2] [10, 3] [10, 4] [10, 5] [10, 6] [10, 7] [9, 6] [8, 6] [8, 7] [8, 8] [9, 8] [10, 8]
 [10, 9] [10, 10] [11, 10] [10, 11] [11, 11] [12, 11] [10, 12] [9, 12] [9, 10] [9, 11] [8, 11] [8, 10] [9, 9] [9, 7] [7, 10] [7, 11] [7, 9] [8, 9] [8,
 5] [7, 8] [6, 9] [6, 10] [9, 5] [9, 4] [7, 7] [11, 8] [11, 9] [6, 8] [8, 4] [11, 7] [12, 7] [7, 6] [6, 7] [5, 8] [5, 9] [4, 9] [4, 10] [4, 11] [5, 10
] [5, 11] [11, 6] [12, 6] [13, 6] [12, 5] [12, 4] [13, 4] [13, 5] [12, 3] [14, 5] [14, 6] [15, 6] [15, 7] [15, 8] [15, 9] [15, 10] [15, 11] [15, 12] [
15, 13] [15, 14] [14, 14]

epsilon Greedy Best First Search(GBFS):
Cost of the solution: 42
The solution path (37 nodes): [0, 0] [1, 0] [2, 0] [2, 1] [2, 2] [3, 2] [3, 3] [3, 4] [3, 5] [3, 6] [3, 7] [4, 7] [5, 7] [6, 7] [7, 7] [8, 7] [8, 6] [
9, 6] [10, 6] [11, 6] [12, 6] [12, 5] [12, 4] [13, 4] [13, 5] [14, 5] [14, 6] [15, 6] [15, 7] [15, 8] [15, 9] [15, 10] [15, 11] [15, 12] [15, 13] [15,
 14] [14, 14]
Expanded nodes (121 nodes): [0, 0] [1, 0] [2, 0] [2, 1] [1, 1] [2, 2] [3, 0] [4, 0] [3, 1] [3, 2] [3, 3] [4, 2] [0, 1] [5, 2] [6, 2] [5, 3] [4, 3] [3,
 4] [4, 1] [4, 4] [3, 5] [5, 1] [3, 6] [3, 7] [4, 6] [4, 7] [5, 7] [6, 7] [5, 0] [5, 6] [6, 1] [6, 8] [5, 5] [5, 4] [6, 9] [4, 5] [7, 9] [6, 0] [8, 9]
 [7, 10] [8, 10] [7, 8] [8, 11] [9, 11] [9, 10] [6, 10] [7, 2] [2, 6] [10, 10] [11, 10] [7, 7] [2, 4] [10, 11] [7, 6] [11, 11] [6, 6] [12, 11] [8, 8]
[10, 12] [8, 7] [9, 12] [5, 8] [10, 9] [9, 9] [7, 11] [10, 8] [9, 8] [7, 0] [1, 6] [9, 7] [8, 6] [9, 6] [8, 5] [10, 6] [10, 7] [9, 5] [11, 6] [11, 8]
[12, 6] [11, 7] [7, 5] [12, 7] [13, 6] [2, 5] [12, 5] [8, 0] [12, 4] [11, 4] [2, 7] [9, 0] [13, 4] [13, 5] [12, 3] [5, 9] [14, 5] [2, 3] [14, 6] [15,
6] [6, 4] [11, 9] [8, 2] [11, 5] [10, 4] [15, 7] [15, 8] [14, 4] [15, 5] [15, 4] [10, 5] [7, 4] [15, 9] [15, 10] [14, 3] [7, 3] [15, 11] [15, 12] [15,
 13] [6, 5] [15, 14] [1, 7] [14, 14]

epsilon Greedy Best First Search(GBFS):
Cost of the solution: 46
The solution path (41 nodes): [0, 0] [1, 0] [2, 0] [2, 1] [2, 2] [3, 2] [3, 3] [3, 4] [3, 5] [3, 6] [4, 6] [4, 7] [5, 7] [6, 7] [6, 8] [7, 8] [8, 8] [
8, 7] [8, 6] [9, 6] [10, 6] [11, 6] [12, 6] [12, 5] [12, 4] [13, 4] [13, 5] [14, 5] [14, 6] [15, 6] [15, 7] [15, 8] [14, 8] [14, 9] [14, 10] [14, 11]
[14, 12] [15, 12] [15, 13] [15, 14] [14, 14]
```

Figure 2: Sample Output

has to traverse to all the possible paths and would cause the problem to take more time and space. As naive agent doesn't take any cost as input it would lead us to the path with maximum cost. Moreover, with the increased nodes the naive algorithms cannot be able to scale up as they re curse through many nodes. In reality, this is not feasible.

2 Algorithms

2.1 Greedy Best First Search

The idea of Best First Search is to use an evaluation function to decide which adjacent is most promising and then explore. Best First Search falls under the category of Heuristic Search or Informed Search.

One of the simplest best-first search strategies is to minimize the estimated cost to reach the goal. That is, the node whose state is judged to be closest to the goal state is always expanded first. For most problems, the cost of reaching the goal from a particular state can be estimated but cannot be determined exactly. A function that calculates such cost estimates is called a heuristic function, and is usually denoted by the letter h. $h(n)$ = The less cost when an agent jumps into a cell.

- GBFS is not optimal. It produces the cost which is not optimal.
- GBFS can be stuck in loop sometimes. This is explained in Presentation. Can be referred from that.

- In order to overcome the above two problem, epsilon GBFS is introduced

Algorithm 1 Greedy Best First Search

```
0: procedure eGreedyBestFirstSearch(maze, start, end)
0:   Create an empty Priorty Queue
0:   PriorityQueue p;
0:   p.insert(start)
0:   while p is not empty do
0:     sort p in ascending order
0:     G = p.pop(0)
0:     if G is not the goal then
0:       For each neighbor V of G
0:       if V is unvisited then
0:         mark V as Visited
0:         p.insert(V)
0:       Mark G as examined
       Else: Exit
```

3 epsilon Greedy Best First Search

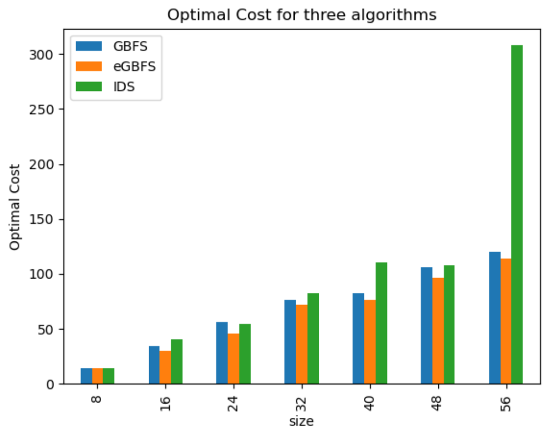
Inspired to develop this algorithm from epsilon feature used in Reinforcement Learning algorithms. It is same as the Greedy Best First Search but with a little modification. In order to overcome the problems associated GBFS some features have been added. Randomness have been added to Greedy Search. Algorithm is epsilon times random otherwise it'll be greedy. Time Complexity is $O(n * logn)$

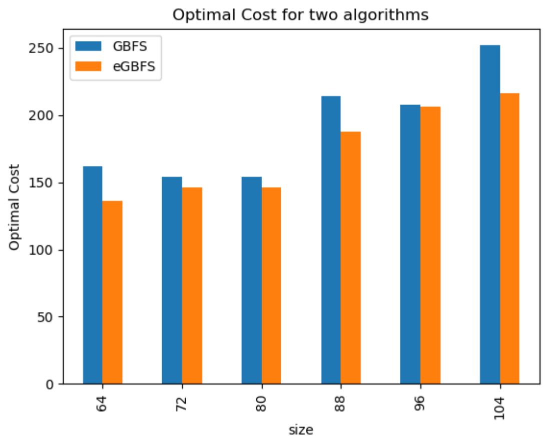
Algorithm 2 epsilon Greedy Best First Search

```
0: procedure GreedyBestFirstSearch(maze, start, end, e)
0:   Create an empty Priorty Queue
0:   PriorityQueue p;
0:   p.insert(start)
0:   while p is not empty do
0:     if e is greater than random value between 0 and 1 then
0:       Shuffle p
       Else: sort p in ascending order
0:     G = p.pop(0)
0:     if G is not the goal then
0:       For each neighbor V of G
0:       if V is unvisited then
0:         mark V as Visited
0:         p.insert(V)
0:       Mark G as examined
       Else: Exit
```

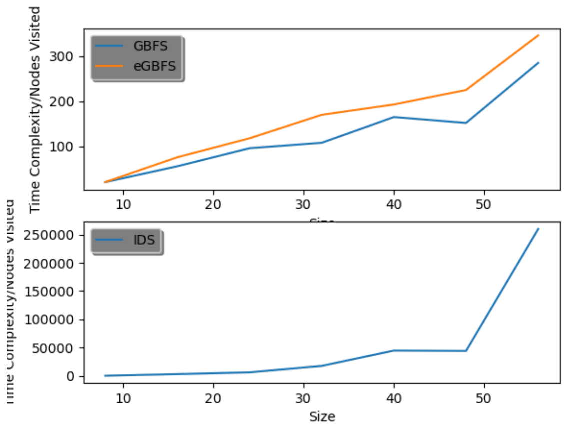
4 Experiment and Results

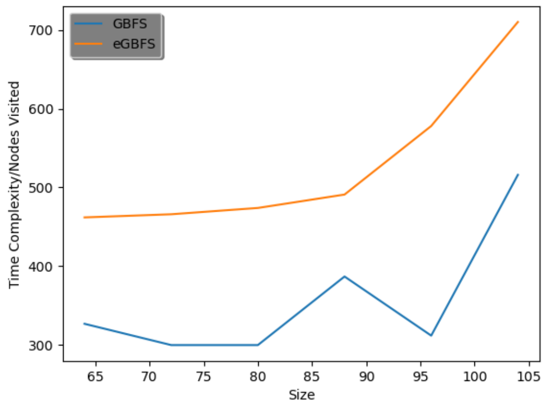
4.1 Optimal Cost



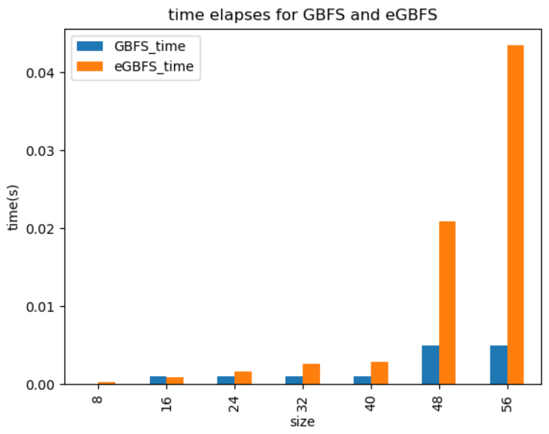


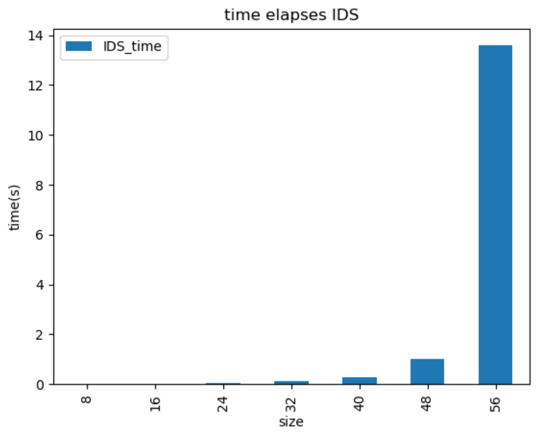
4.2 Time Complexity

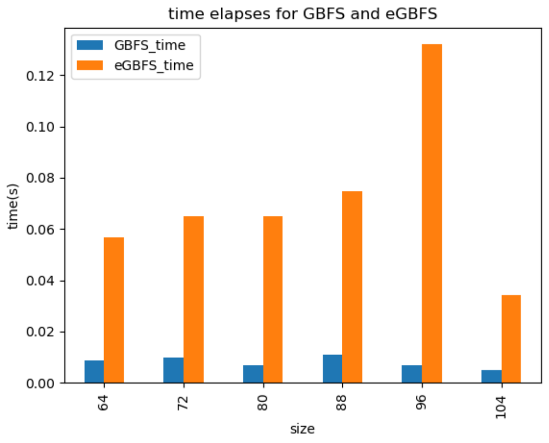




4.3 Time Elapsed





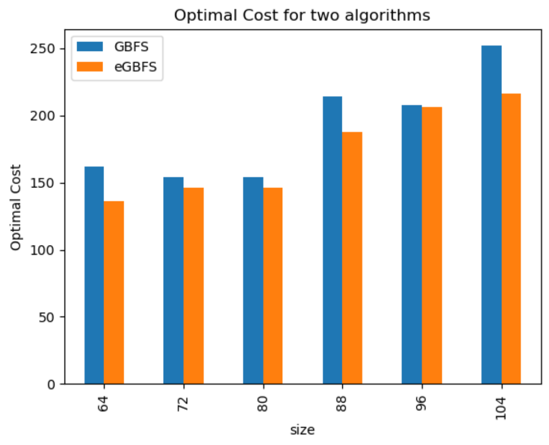


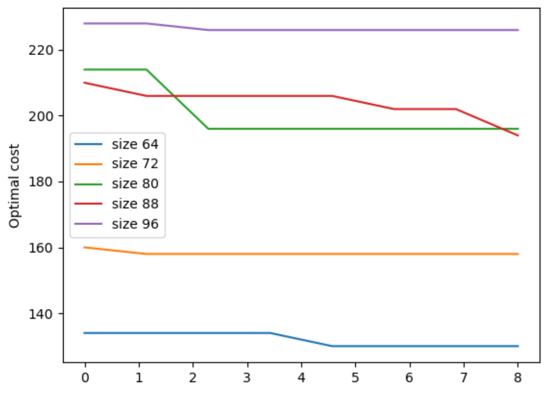
- It can be inferred that eGBFS is giving us the optimal cost path for the algorithm

- Naive method like IDS is giving non optimal cost and unable to scale upto large space problems.

5 Sensitive Specs

There are two important sensitive inputs in the above problem. They are the dimension of the maze and the epsilon value for the eGBFS. As the dimension of the maze increases, the naive algorithm IDS is unable to scale where as GBFS is unable to give the optimal cost of the problem. For eGBFS, the epsilon value plays an important role in deciding whether the algorithm to give an optimal value. As the epsilon value is increasing the optimal cost of the problem is getting better. It can be said that high randomness is giving the best results. This can be seen from the below plot.


Optimal Cost for two algorithms
GBFS
eGBFS
Optimal Cost
size


Optimal cost
size 64
size 72
size 80
size 88
size 96