Homework 1

Zhengyang Ling(zl35111), Yalin Liu(yl1255), Yujia Liu(yl1261)

February 19, 2021

1

(1)

Problem 1

```
def maze(d,p): m = np.random.binomial(1,1-p,size=(d,d))
m[0,0]=1
m[d-1,d-1]=1
plt.matshow(m, cmap=plt.cm.gray)
plt.text(x=0, y=0, s='s')
plt.text(x=d-1, y=d-1, s='g')
plt.show()
return m
```

(2)

Problem 2
Code in dfs.py

If we want to proof maze is reachable, we only need to prove there exists a route from source to goal. BFS will traverse all the possible node. Compare with BFS, DFS will find a possible route at first. Thus, it is a better choice than BFS.

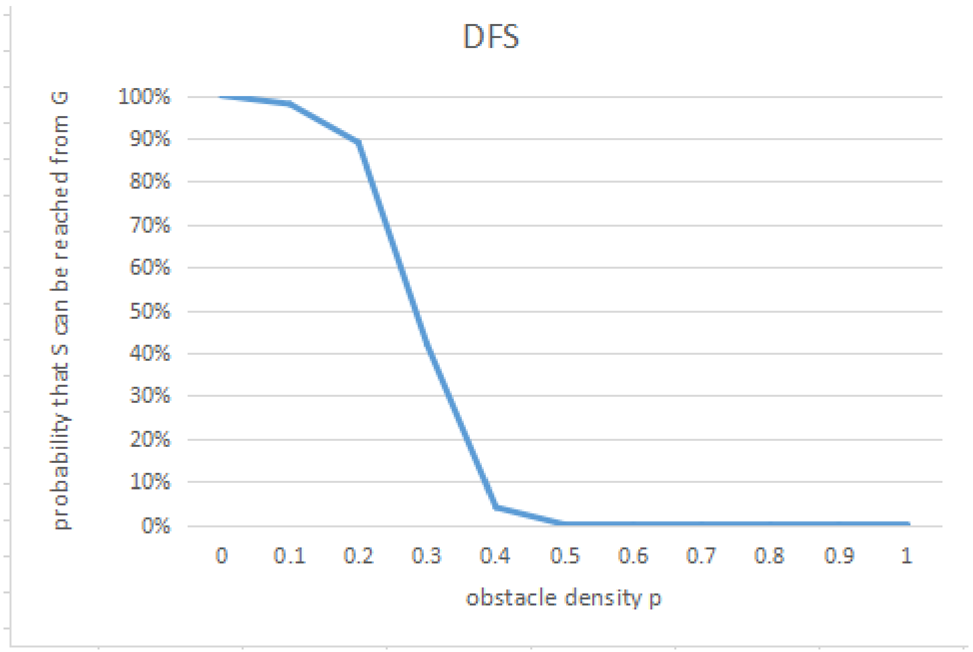

Figure 1: ‘obstacle density p’ vs ‘probability that S can be reached from G’(100*100 400times/0.1).

(3)

Problem 3
Code in bfs.py and Astar.py

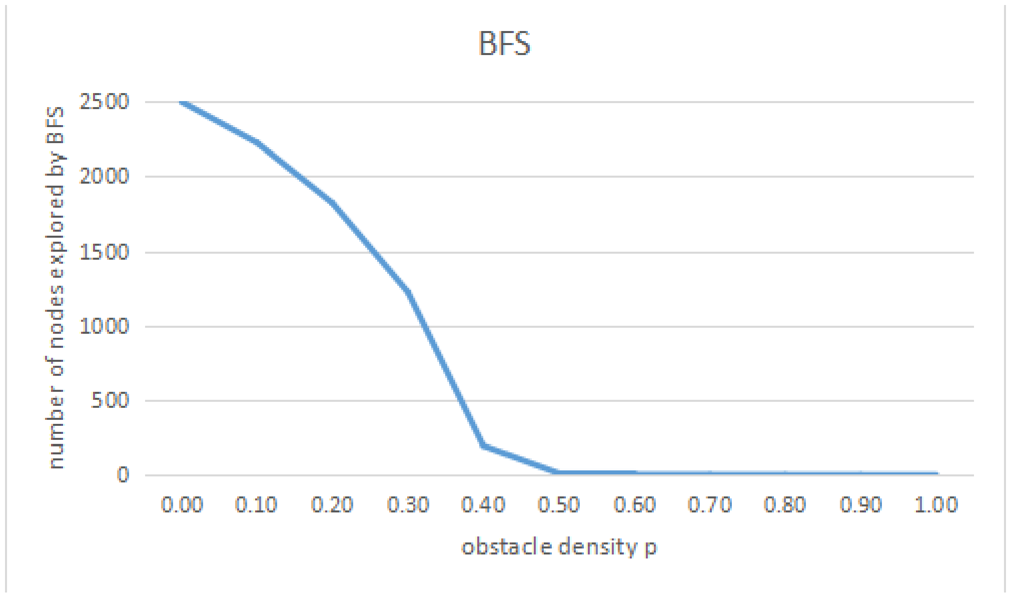

Figure 2: ‘number of nodes explored by BFS’ vs ‘obstacle density p’(50*50 200times/0.1).

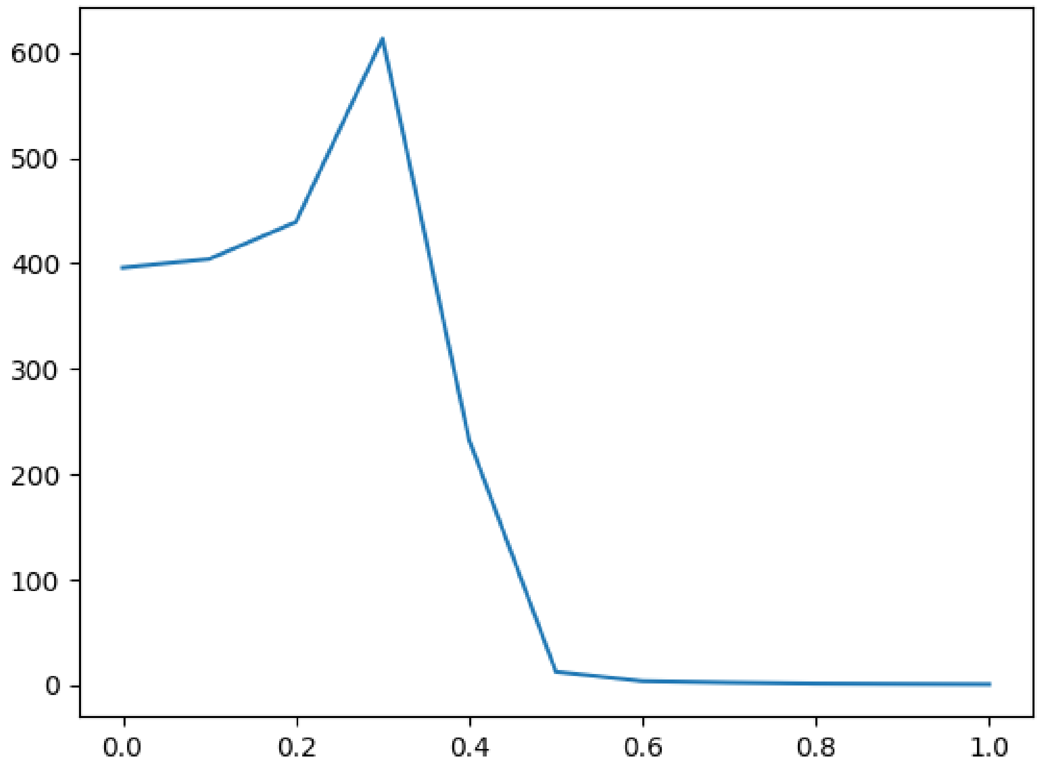

Figure 3: ‘number of nodes explored by A∗’ vs ‘obstacle density p’(50*50 200times/0.1.

(4)

Problem 4

In One Minute:

DFS about 8100*8100 The size that can be processed is closely related to the internal structure of the maze. It’s especially slow when there is no way to go near the end, even dozens of times.

BFS about 250*250 The size that can be processed is related to the internal structure of the maze. When dealing with mazes of the same size (p=0.3), the difference basically will not exceed twice.

A* about 1000 * 1000 The size that can be processed is closely related to the internal structure of the maze. It’s especially slow when there is no way to go near the end, even dozens of times.

(The slow speed of DFS, BFS and A* may be related to my need to display the shortest path or my old computer)

(5)

Problem 5
Code in Strategy3.py

I gave our strategy 3 a name: Safety FOrecast Search (SFOS) SFOS do a fire situation forecast to show the fire's fastest steps to each grid. for example 2 for fire 0 for grid.(and fire will not burn the wall or obstacle)
(0 , 0 , 0 , 0),
(0 , 0 , 0 , 0),
(0 , 2 , 0 , 0),
(0 , 0 , 0 , 0),

SFOS will create a new maze with same d and change cell become this (0 for fire , from 1 to infinity is the fire's fastest steps to each grid) (3 , 2 , 3 , 4),
(2 , 1 , 2 , 3),
(1 , 0 , 1 , 2),
(2 , 1 , 2 , 3),

When maze has a node which have been fired, we assign to each node a piority which is decided by after how many steps this node will be fired in the worst situation according to the node which have been fired. So we can be sure that the path we are going to take will not be burned by fire. After we go one step, fire may become more , and then we do the fire situation forecast again(yes every step).Repeat this process until the end.

(6)

Problem 6
Code in Strategy3.py Strategy2.py Strategy1.py,
Strategy1 agent often burned or rushed into the fire, but the fastest calculation.
Strategy2 agent still often burned but better.Because they are greedy and want to go the closest way, they often walk one cell around the firee, and then the fire spread. And calculation takes longer time.
Strategy3 is our SFOS. I have explained in the Problem 5.But the calculation takes the longest.But! Agent will never be burned in our SFOS!
The same point: all the Strategy need to go step by step and use bfs. When the origin fire is close to the bottom right corner or/and the diffusion rate is high or occupy the only way you can pass, agent may not succeed in going out because the road or the end point has been ignited.

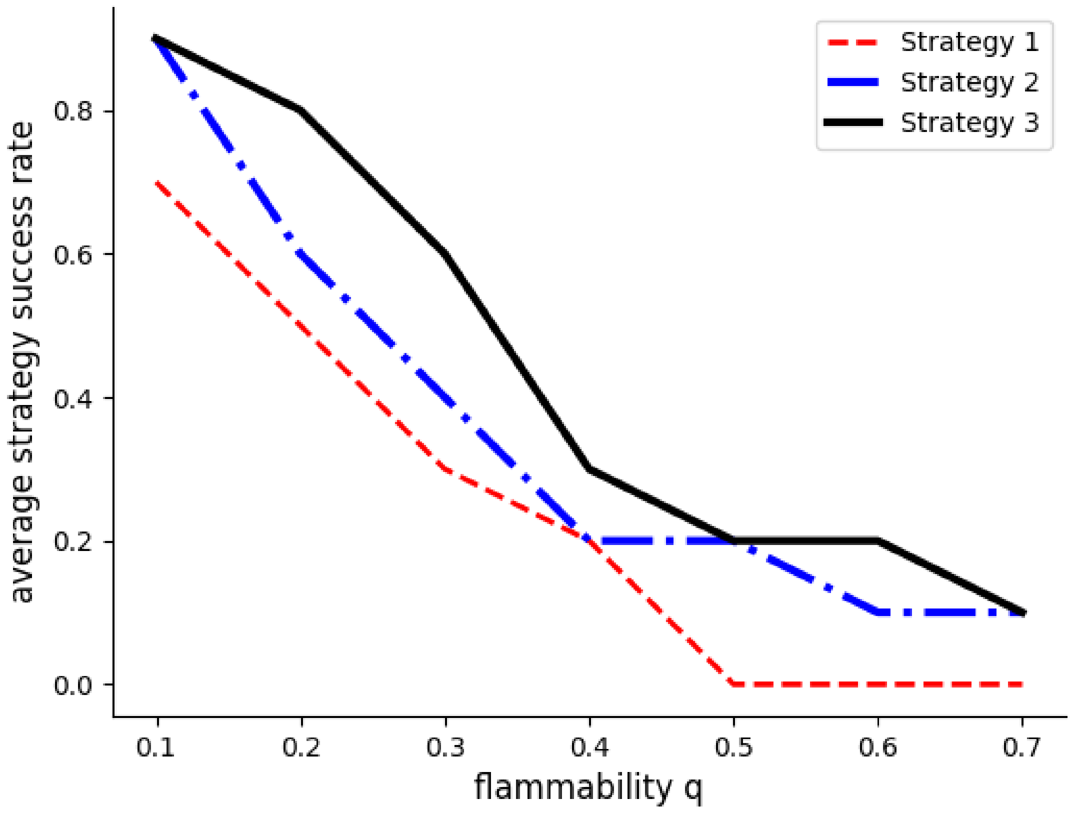

Figure 4: ‘average strategy success rate’ vs ‘flammability q’ at p = 0.3, 40*40, 10times/0.1q

(7)

Problem 7

If we have unlimited computational resources, we will update our SFOS. Currently our SFOS can only predict the worst case: the fire’s fastest steps to each grid. We will combine the fire diffusion rate(q) to rigorously calculate the expected value of the fire that will spread to each grid and keep the decimal point. At the same time, use a better wayfinding system combine bfs and A*: While walking the shortest path, try to step on the grid with a large prediction number as much as possible, and make a dangerous choice when there are no any other options. For example, when agent can only choose to cross the grid with a predicted value of 1.We maybe will rename this ”Improve Strategy 3”.

(8)

Problem 8

If we could only take ten seconds between moves, we will use A* algorithm to explored as many node as we can in this ten seconds to try to go down right to get closer to the end. If we are in very serious trouble, we cannot move any step from the bottom right within ten seconds, I will think about ten more seconds on the spot even if a step is wasted