Sneh Shah (sjs413) - Section 2

Rishi Shah (rrs141) - Section 5

Intro to AI Project 1

()

February 19, 2021

Certification of Academic Integrity

I certify that the work done on this project is my own and is not copied or taken from online or any other student's work. - Rishi Shah
I certify that the work done on this project is my own and is not copied or taken from online or any other student's work. -Sneh Shah

Breakdown of Work

Rishi coded the genMaze, makePath, DFS, BFS, and A* functions. Rishi answered the first 4 questions on the write-up.

Sneh coded strategy 1, strategy 2, and strategy 3, and the fireMaze functions. Sneh answered questions 5-8 on the write-up.

We worked together to come up with the pseudocode for all functions together and split up the coding as mentioned above.

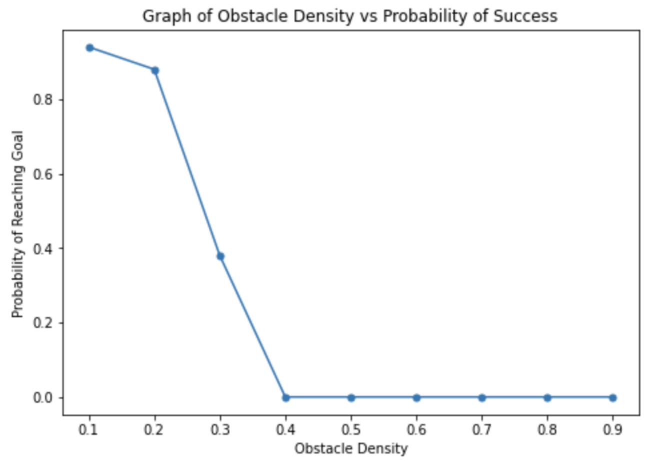
Question 1.

To generate a maze with a given dimension and probability p, we have a function that takes in these two parameters and goes through a nested for loop. For each block, we determine if the cell should be blocked or unblocked based on p. This function is called genMaze() and is included in the code.

Question 2.

DFS is better when determining if a path between two locations exists because we are not looking for a shortest path. The fringe in DFS is smaller than the fringe in BFS and therefore DFS will be faster when determining if a path exists. A plot of obstacle density p vs. probability of reaching goal is shown below.

```
[0.94, 0.88, 0.38, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

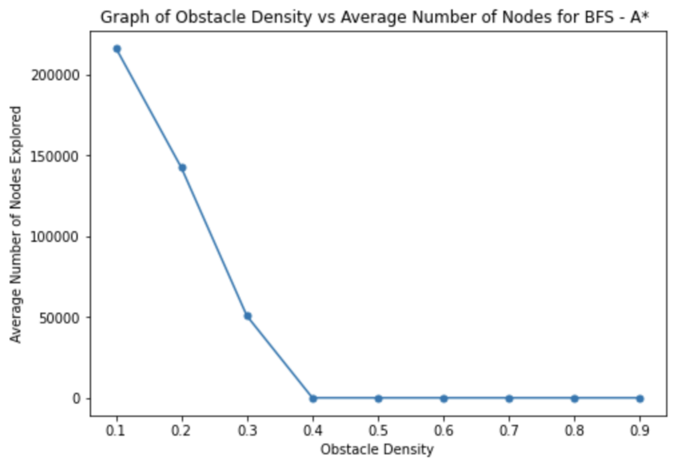


The plot shows that as p increases, the probability of reaching the goal decreases. This result makes sense because if p is high, the chances of a cell being blocked are high. The more cells that are blocked, the less likely it is to reach the goal because of all the obstacles.

Question 3.

Both BFS and A* algorithms are included in our code. The plot for number of nodes explored by BFS - number of nodes explored by A* vs. obstacle density p is shown below.

```
[216700.14, 142553.0, 50740.56, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```



Since A* uses a priority queue, it explores less nodes than BFS. In BFS, you could explore paths that do not lead to the goal, but since A* has a heuristic and ranks the children in the priority queue, it explores options that seem to have better outcomes first. This is why there is a large difference between the number of nodes explored by A* and BFS. We can see this difference decrease as p increases because it becomes more difficult to find a path when many cells are blocked. When there is no path from S to G, BFS and A* explore the same number of nodes because both algorithms check every possible solution before returning that there is no path available.

Question 4.

For DFS at p = 0.3, the largest dimension we can run is 4500.

For BFS at p = 0.3, the largest dimension we can run is 2700.

For A* at p = 0.3, the largest dimension we can run is 3200.

These dimensions are reflective of how large the fringe is for each algorithm. For BFS, since the fringe grows exponentially, the largest dimension is only 2700. However for A*, where less nodes are explored, the largest dimension is 3200.

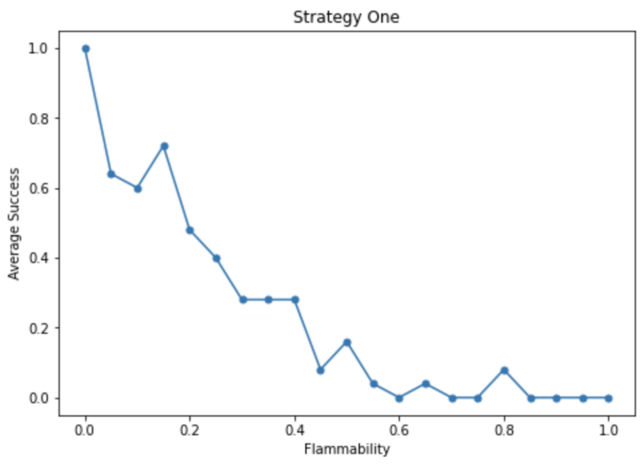
Question 5.

For our strategy three, we tried to consider what the fire would look like in future states. We do this by adding the number of neighbors that are on fire to the heuristic when using A* to recalculate the path. We also only recalculate the path when the next cell in the current path is on fire or has neighbors that are on fire. This latter condition means that this next cell is in danger of catching on fire. This accounts for future states because we are predicting what the fire could look like in the future, as if a cell has neighbors that are on fire, that cell is likely to be on fire in the near future. Additionally, by adding the total number of neighbors on fire to the heuristic, we are avoiding selecting paths that are more likely to catch on fire. This algorithm can be called CNN (Check Neighbors' Neighbors) since we are checking the neighbors' neighbors for any surrounding fires.

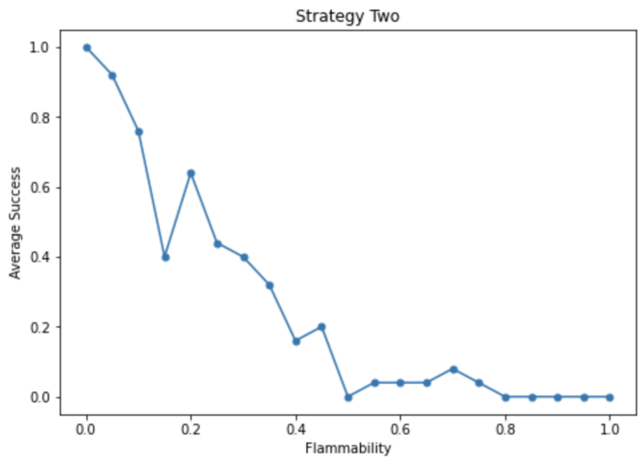
Question 6.

The plots for average strategy success rate vs. flammability q at p= 0.3 for strategies 1,2, and 3 are shown below.

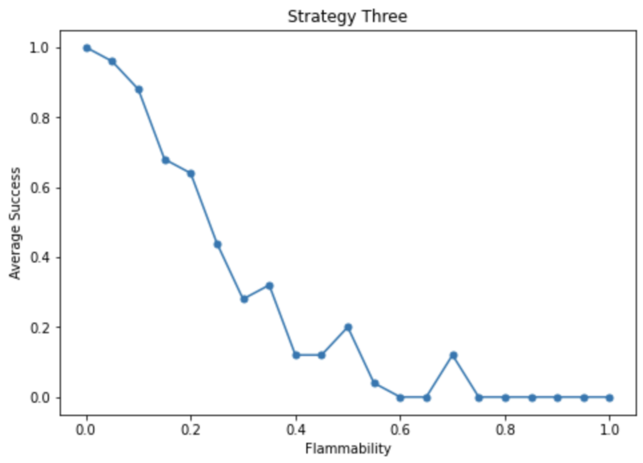
```
[1.0, 0.64, 0.6, 0.72, 0.48, 0.4, 0.28, 0.28, 0.28, 0.08, 0.16, 0.04, 0.0, 0.04, 0.0, 0.0, 0.08, 0.0, 0.0, 0.0, 0.0]
```



```
[1.0, 0.92, 0.76, 0.4, 0.64, 0.44, 0.4, 0.32, 0.16, 0.2, 0.0, 0.04, 0.04, 0.04, 0.08, 0.04, 0.0, 0.0, 0.0, 0.0, 0.0]
```



```
[1.0, 0.96, 0.88, 0.68, 0.64, 0.44, 0.28, 0.32, 0.12, 0.12, 0.2, 0.04, 0.0, 0.0, 0.12, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```



All three plots show us that as q increases, the success rate decreases. Although there are some spikes, the trend of all graphs is generally decreasing. If q is high, the likelihood of the fire spreading is also high which decreases the chances of the agent successfully reaching the goal. However with strategies 2 and 3, we see that the success rate is generally higher since we are recalculating the path where in strategy 1 we stick to the original path. For example, at $q = 0.2$, the success rate for strategy 1 is 0.48, the success rate for strategy 2 is 0.64, and the success rate for strategy 3 is 0.64.

Regarding the time each strategy takes to run, Strategy 1 takes the smallest amount of time as it involves running A* only once. Strategy 2, on the other hand, takes the longest time to run as you are running A* for every step you take in the maze. Strategy 3 is designed to be a bit faster than Strategy 2 since you don't rerun A* every time. You're only rerunning it when the next node you're going to is on fire or has neighbors that are on fire.

Question 7.

If we had unlimited computational resources, we can improve strategy 3 by solving for all possible paths to the maze. Between all the paths we find, we can choose the one that's the least risky in terms of obstacles/fires and distance from goal. To do this, we can use a heap similar to how we used in strategy 3 and assign weights to the path depending on those aforementioned factors. You can rerun this "find all paths" algorithm when you are close to a fire. To measure this closeness to a fire, you can also keep track of the fire's locations, so that you can calculate its distance from your current location. If the fire is a certain threshold away from you (say 2-3 spaces), then you can find all the possible paths again from your current spot and choose the most appropriate one.

Question 8.

If we only had ten seconds to make a decision on where to move next, we would build an algorithm that makes the agents check their neighbor and their neighbors' neighbors to move in the direction of the goal. The agent would be greedily solving the maze, since calculating the entire path to the goal would take time and computation. By just checking their current surroundings, the agent will take less time before making each move. If the goal is on the bottom right, the agent will go down or right if possible. If the agent cannot move down or right because of fires, then the agent will back track by moving up or left until they can travel down a new path.

For this type of algorithm, we value speed more than length of a path, as you only have 10 seconds to make a move. However, despite this, we have to consider that the longer that you're in the maze, the more fires will spread.