Intro to AI Project 1 Write Up

February 19, 2021

On my honor, I have neither received nor given any unauthorized assistance on this assignment. All of our work and ideas are our own. We did not use any other external sources.
- Michael Zhang, mbz27, Section 1

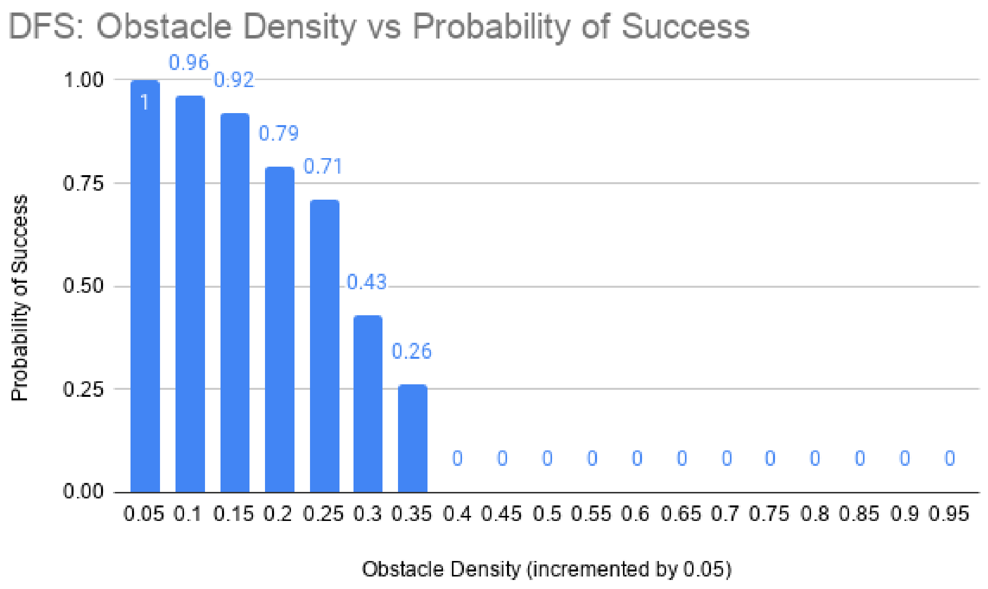
On my honor, I have neither received nor given any unauthorized assistance on this assignment. All of our work and ideas are our own. We did not use any other external sources.
- Prasanth Balaji, pmb162, Section 3

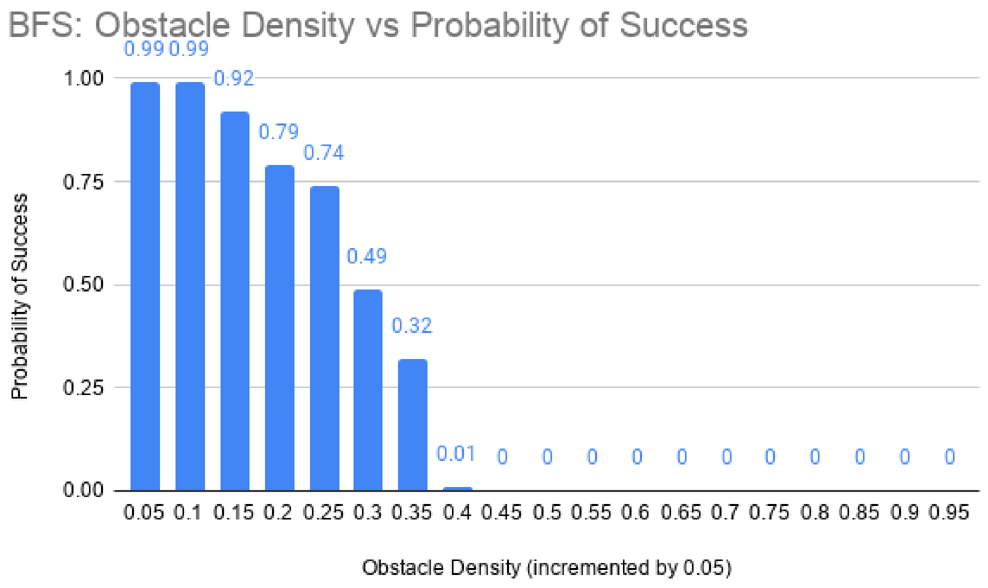
Group members: Michael Zhang, Prasanth Balaji

Part 1

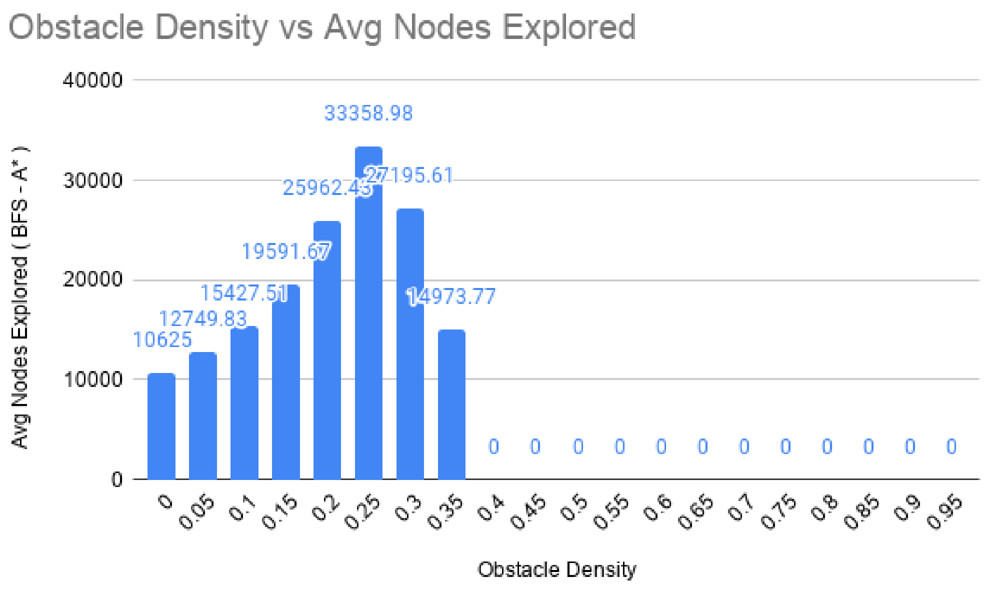
2. DFS is better than BFS in the case that you need to determine whether a path between two locations is possible. This question does not necessarily ask for the most optimal path nor the fastest algorithm, since both algorithms have the same time complexity and we only need to find one path that leads to the goal. DFS is exponentially more efficient than BFS in terms of space complexity, as there will be significantly less items on the fringe once a path is found. DFS is considered to be linear, as we are analyzing the children of only one node at a time resulting in O(b * k). On the other hand, BFS

is considered to be exponential, as we are analyzing the children of multiple nodes at a time and going level by level resulting in O (b to the power of k).





3. If there is no path from S to G, then there is no difference in the amount of nodes explored. Since both algorithms need to explore every possible node to conclude that there is no path to the goal (ie. the worst case). On the other hand if there is a path, then there would be a difference since A* is more efficient.



4.

- DFS at p = 0.3: dimension = 10,000

- BFS at p = 0.3: dimension = 9,000

- A* at p = 0.3: dimension = 6000

Part 2

5. FUSE algorithm - Fire Utility Star Euclidean algorithm
For our strategy 3 we started out considering a method to simulate the fire

on the maze in order for the algorithm to take into account future fire placement. We gave each tile a utility value which is the number of steps until the simulated fire reaches that tile. In order to get a somewhat accurate utility value we decided to run the fire simulations multiple times and averaging out the number of steps. There were a few technical problems with this approach:

1. If the initial fire is surrounded by blocks, or the fire is not spreading, the fire will never reach the any tiles which causes some simulations to run on forever

2. In the case where the fire is free to spread, the path chosen was usually skewed to the opposite edge of the maze from fire (essentially hugging the edge of the maze). This was a problem because the chosen path was very long and did not prioritize going towards the goal but to just run away from the fire.

To address problem 1 we limited the number of steps the fire can make towards a tile thus the simulation will end after a fixed number of steps so it does not run on forever. But with this "fix" there will be cases where the fire simulation will simply not have enough steps to reach the tile and this will result in the majority of the tiles getting assigned the same utility. This was another problem since when the fire does not spread the player simply picks a completely random path to the goal and can just run back and forth between two tiles resulting in another infinite loop.

The reason why the fix for problem 1 does not work is because it deals with utility value ties (when comparing two tiles with the same utility values) completely randomly.

To address both problem 2 and the problems with "fixing" problem 1 we incorporated the euclidean distance heuristic from A*. Instead of dealing with utility value ties randomly we will compare the tiles' euclidean distance heuristics so that it will prioritize picking the most optimal path to the goal as opposed to just running away from the fire or picking a random path. This solves both problems since now the player will not hug the edge of the maze and it will not simply pick a random path to the goal.

To summarize how strategy 3 orders its priority queue:

1. It simulates fire multiple times and keeps track of the number of steps to reach a specific tile.

    - We will limit the number of steps the fire can make towards that tile

2. Average the number of steps for that specific tile which will be it's utility value

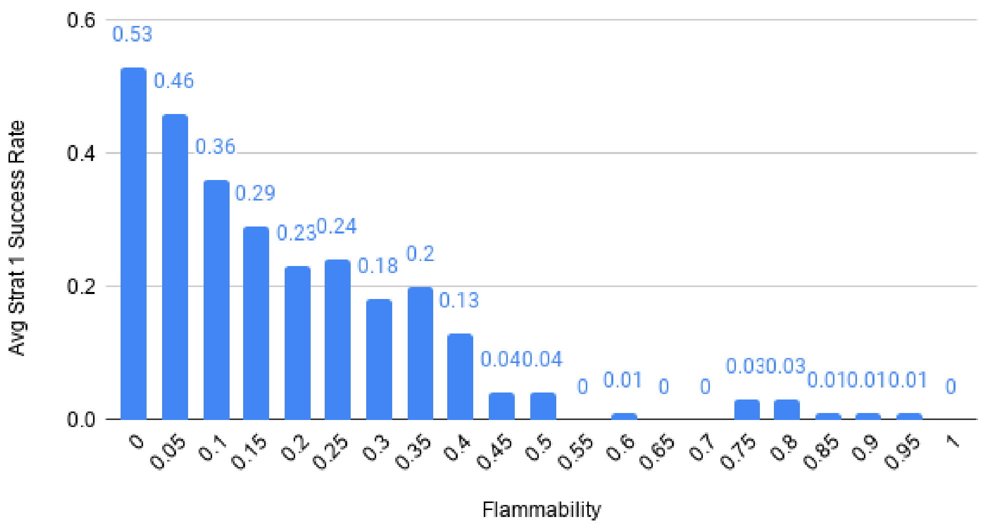
    - A higher utility value means the tile is less likely to catch on fire

    - A low utility value means the tiles is more likely to catch on fire

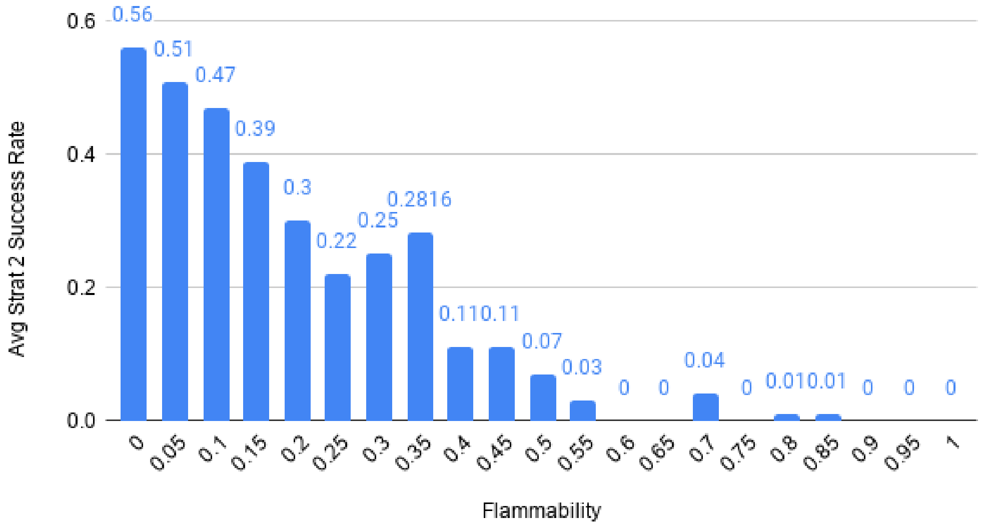
3. If two tiles have the same utility values we will break the tie by using the euclidean distance heuristic

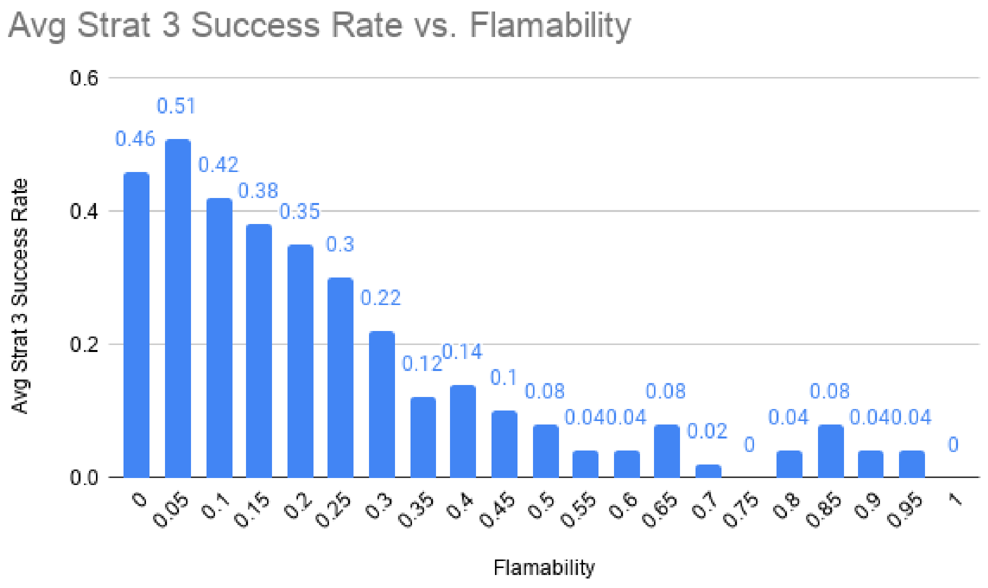
6. Strategy 1, strategy 2, and strategy 3 are essentially the same when q = 0, which results in no fire spreading across the maze. Since all three strategies will use the euclidean heuristic to find a path to the goal. The only aspect that differentiates strategy 1 and strategy 2 is that strategy 2 accounts for the current placement of the fire by recalculating a path after every move, whereas strategy 1 only accounts for the initial placement of the fire. Strategy 3 is quite different from strategy 1 and strategy 2 since it accounts for future iterations of the fire in order to preemptively avoid the path of the fire.

Avg Strat 1 Success Rate vs Flammability



Avg Strat 2 Success Rate vs Flammability





7. If we had unlimited computational resources, then we would simply run our simulation almost an infinite amount of times to insure the accuracy of the assigned utility value.

8. If we are limited on computational resources, then considering that in strategy 3 most of the overhead comes from simulating multiple possible iterations of the fire we should essentially remove this computational overhead. As a compromise we should instead implement a restricted area around the current position of the fire where the utility value is essentially zero. Tiles that are not in the restricted area will simply have the same utility values but are greater than zero thus they have to rely heavily on the astar euclidean heuristic to find an optimal path to the goal. This makes it so that the chosen steps will be an optimal path to the goal while having a sort of a "buffer

zone" between the chosen step and the fire.

For the writeup, we both contributed by doing different parts and we meet up to review it together. While reviewing it, we both discussed and added any final ideas.

Documentation

How to run the project:

Please enter the arguments in this order and omit the prob. of fire spread argument if the algorithm does not need fire.

Dimension must be greater than or equal to two

- java FireMaze [type of algo to run] [block density] [maze dimension] [prob. of fire spread]

- type of algo to run: dfs, bfs, astar, strat1, strat2, strat3