CS 440: This Maze is on Fire

Steven Nguyen & Kyra Kennedy

19 February 2021

0.1 Abstract

In the first part of the project, we demonstrate generating 2D mazes on command line and demonstrate navigating through the mazes using DFS, BFS, and A*.
In the second part of the project, we demonstrate again navigating, however, there will be a special barrier randomly placed in the maze that expands randomly through neighboring cells.

0.2 Problem 1

Please see the attached file, *maze.py*, for both generating the maze structure and for filling the structure given parameters.

0.3 Problem 2

Please see the attached file, *dfs.py*, for the implementation of DFS.
For finding if a point is reachable from the other, DFS is better than BFS in this case because while both searches find the path in at most the same amount of steps and at least the same amount of steps, DFS uses a much less amount of memory while searching.

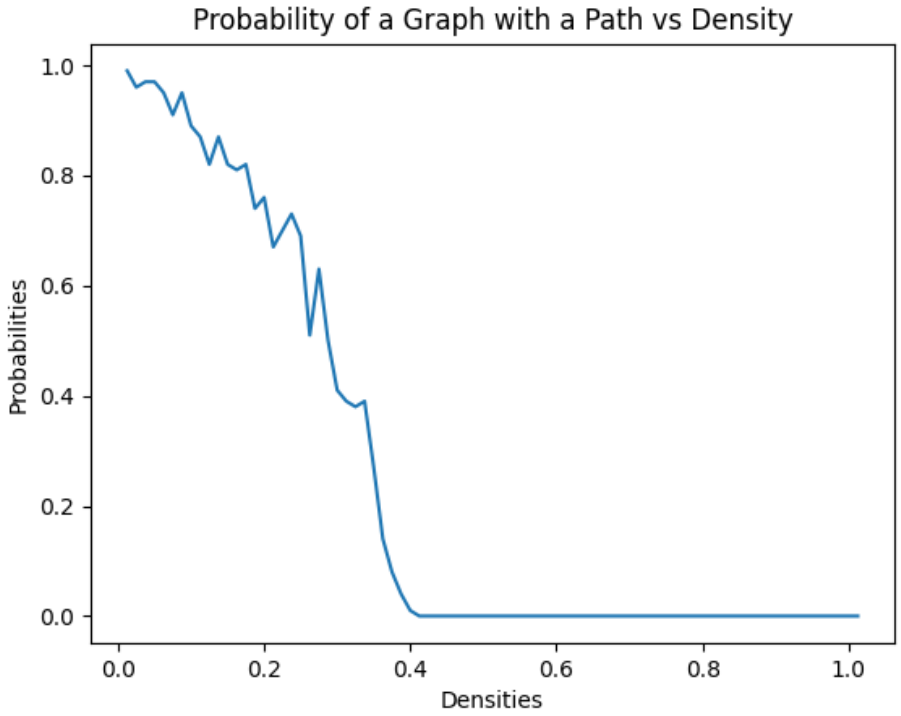

Figure 1: Compares the probability that a graph has at least one valid path from start to finish using Manhattan Distance in random 100 by 100 mazes vs density

We determined that as density increases, the probability that a maze has at least one valid path is lowered.

0.4 Problem 3

Please see the attached files, *bfs.py* and *a.py*, for the implementations of BFS and A*.
In our project, when plotting *the number of nodes explored by BFS* and *the number of nodes explored by A** vs *the density of the generated maze.*

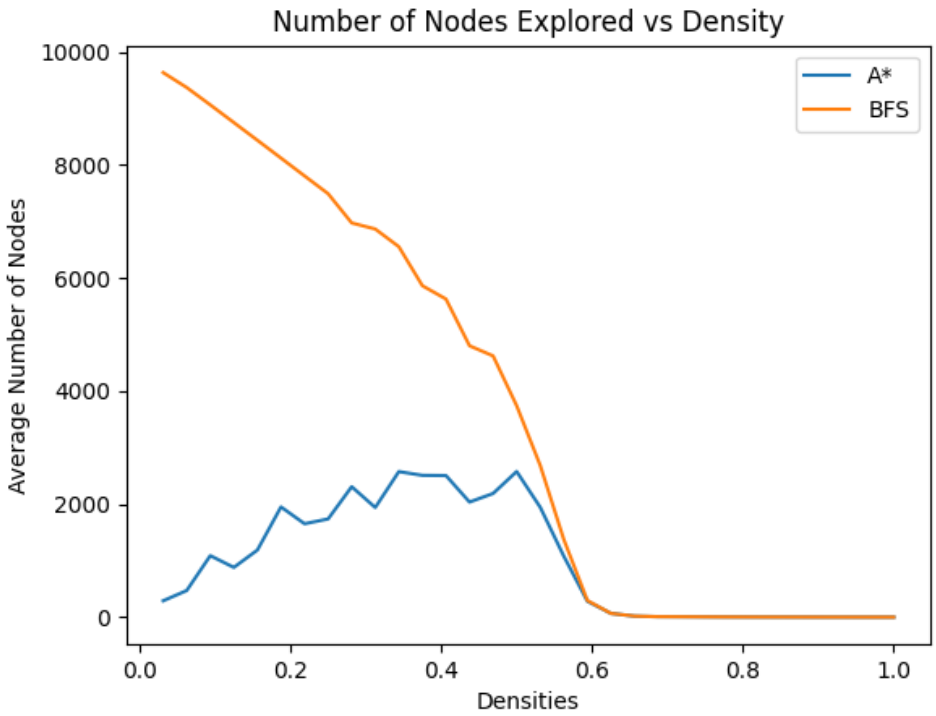

Figure 2: Compares the average number of nodes explored by Euclidean Distance in random 100 by 100 mazes vs density for both A* and BFS

Up until around .5 density, the average number of explored nodes in A* increases because longer paths must be generated with more barrier. Then after around .5, there are significantly more mazes that do not have a valid path from start to finish, so less nodes are explored because both there are less explorable nodes and less paths.
For BFS, because the mazes are designed so the start is on the top left and end is on the bottom right, BFS will search almost all cells in the maze and

will end up searching much more cells than A* when the density is low.

0.5 Problem 4

Max Size Maze for Search Algorithms	
Search Algorithm	Max Size
DFS	1300 by 1300
BFS	3300 by 3300
A*	1500 by 1500

Figure 3: Within 60.00 seconds, at 0.3 Density, DFS can search a 1300 by 1300 maze, BFS can search a 3300 by 3300 maze, and A* can search a 1500 by 1500 maze.

A* is slow because it is spending too many operations finding which of its neighbors is closer to the end whenever scanning each cell. DFS is slow probably because the start and end cells are not randomly placed and DFS, in our case, scans neighbors starting from the top in a clockwise direction. So, DFS spreads in the downward-left direction while the goal is always in the downward-right direction.

0.6 Problem 5

For strategy 3, aptly named Too H.O.T. to Handel, where H.O.T stands for Hop Out The way. For this strategy, at each step the agent checks its immediate neighbors to its north, south, east, and west. If one of them is on fire, a new path is computed, otherwise it continues on the last path it computed. By doing so, a new path doesn't have to be created at every turn, such as in strategy 2, but still allows for re-computation, unlike strategy 1.
This accounts for the unknown future in that when fire is nearby, it acknowledges that it poses a threat in the near future and computes a new path to now avoid it. In this way, the strategy is more aware of where the fire is and how it should move around it.

0.7 Problem 6

Please see the attached files, *strategy_1.py*, *strategy_2.py*, and *strategy_3.py* for the implementations of strategies 1, 2, and 3.
Strategy 1 does BFS at the beginning of the pathfinding and continues using that path.
Strategy 2 does BFS at each step recalculating a new path to accompany the changing fires.
Strategy 3 does BFS when the current cell is neighbored by a fire.

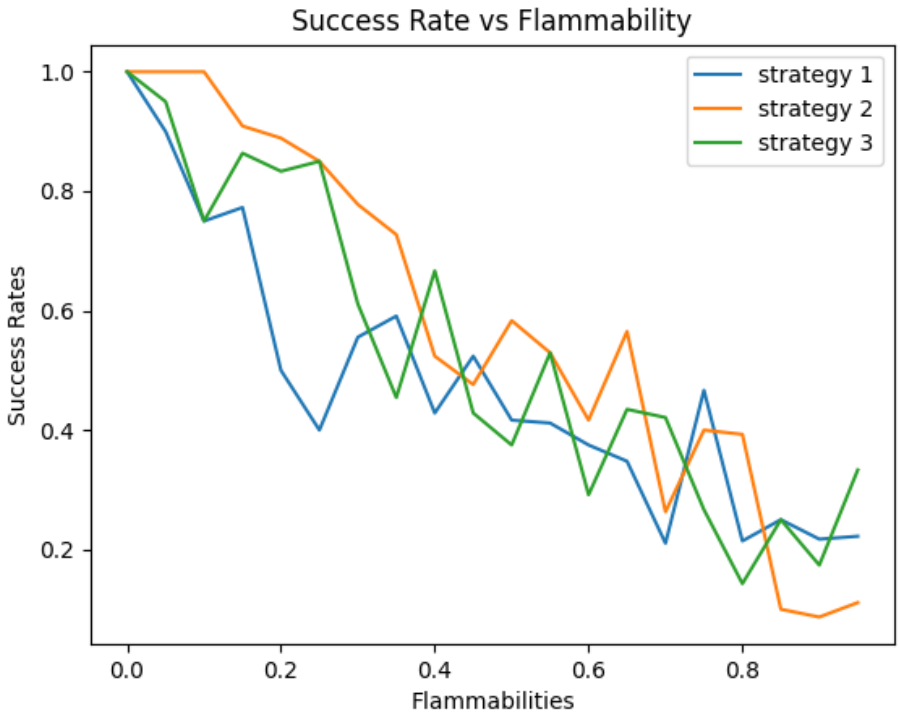

Figure 4: Compares the success rates of the 3 strategies given different fire flammabilites.

Strategy 1 is the worse overall in pathfinding with fire with no accompanyment of the changing fire. Strategy 2 is the best by changing the path each step. Strategy 3 is similar to strategy 2, but though it doesn't change the path often enough to completely avoid being caught by fire, it runs much

faster by not running BFS as often.
The three strategies run the same while the fire is minimal because they only differ in fire detection.

0.8 Problem 7

With unlimited computational resources, strategy 3 could be improved with a more Stochastic model. This would be able to calculate an indepth game tree for any size maze and allow the agent to find the most effective path even for future fire spread. Since the code for this was gone over in recitation, it wasn't the chosen strategy, but its inclusion would improve the code's ability to plan for the future spread by analyzing all possible paths of the fire.

0.9 Problem 8

With a limited time between moves, we would reduce the computation by reducing the length of the path used in the prediction e.g., pathfind to just a bit after the fire rather than all the way to the end; and by reducing how many paths are searched when predicting, such as paths that go in the opposite direction of the goal.