Maze On Fire

VRINDA JAIN, ADITYA ANIKODE, AZIM KHAN

February 23, 2021

Abstract

Mazes are a great way to incorporate the complexities of path finding algorithms. Similarly in this report, we will showcase different strategies such as Breadth First Search, Depth First Search, and A star. We compared the time and space complexities of these algorithms to determine which is effective for different scenarios. We also discovered strategies using a combination of those algorithms to surpass a Maze on Fire.

1 Maze with different block densities

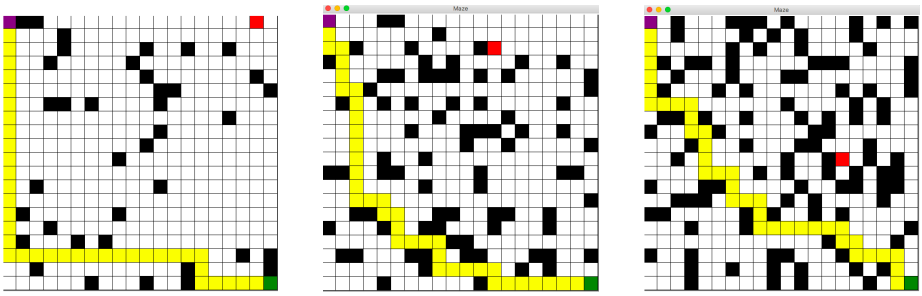

Figure 1: Block Density at 0.1,0.2,0.3

2 Why is DFS a better choice than BFS when determining if path exists?

The problem asks us to determine whether there is a path between the starting point and ending point– not to find the shortest path. Since we simply need a yes and no answer, we do not need the path to be the most optimal. Therefore, we only require DFS to solve this question because it will take much less space. BFS expands more nodes due to this searching technique finding the optimal/shortest

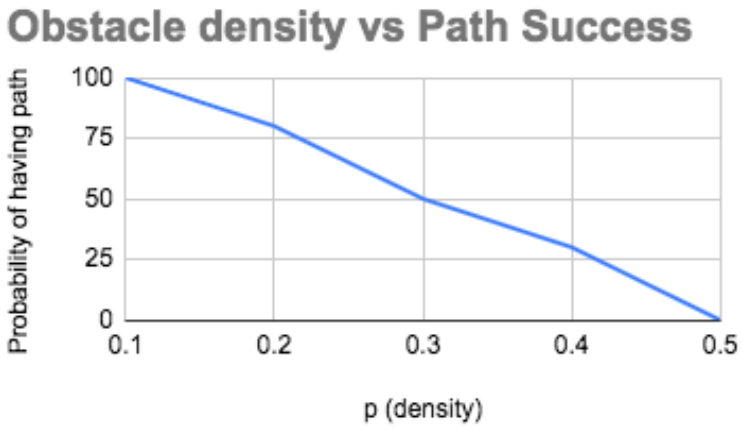

Figure 2: Block Density vs Path Success

path to the goal. DFS thus has the least amount of nodes expanded, conserving space complexity as it just finds A path to the goal. Even though the path BFS will be the shortest path, it will still explore other paths which will take up more space because it is not guaranteed that the first path will be correct. Since we are working with a maze, the goal will most likely be far away from the starting point, so it is better to use DFS because it will use the least nodes to reach the target.

We implemented DFS using recursion which slowed down our code. We simply let it expand on one node recursively until it reached the end.

[**?**]

2.1 Why is BFS a better approach when finding the shortest path?

Why is BFS a better approach when finding the shortest path? BFS is a better approach because it has the ability to explore all possible paths that lead to the goal and will choose the short path. DFS has a greater maximum fringe size, however BFS has more nodes expanded and it is vital that we discover more nodes because they may lead to an optimal path compared to one long fringe. The DFS limits the path options, diminishing the chances of finding the most optimal path.

3 Comparison between BFS and A*

Our BFS Algorithm used queues to traverse the graph to explore the neighbors until we reached the goals. Depending on the path we took, we returned that solution. While making the A* algorithm, we explored different strategies by

utilizing different heuristics such Manhattan, Euclidean, and Diagonal Distance. In the end we learned through testing that Euclidean distance provided the optimal solution.

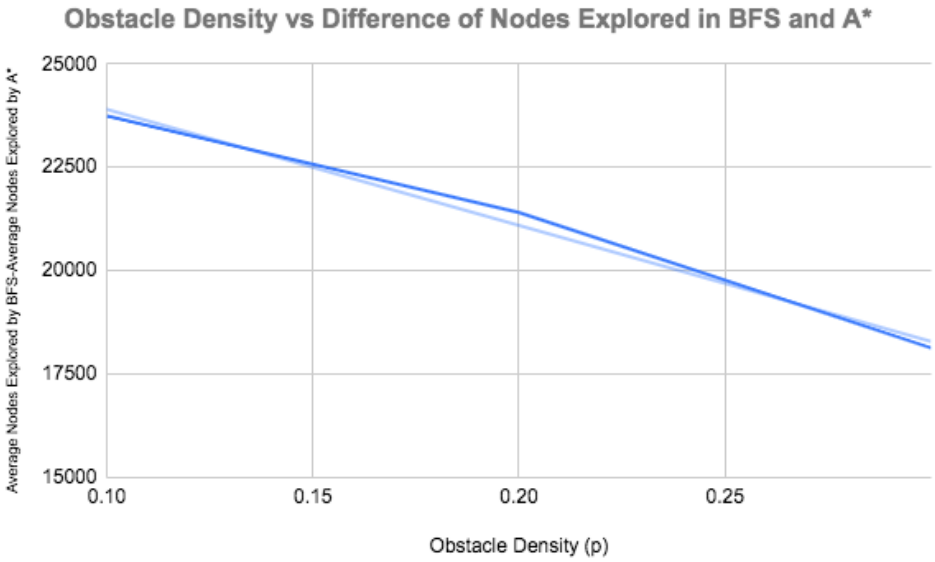

Figure 3: "Average of nodes explored by BFS - of nodes explored by A*" vs obstacle density p for largest dimension possible

3.1 What if there's no path between S and G?

When there is no path between the start and end point, then we have reached the worst case possible. The difference would be 0 because both methods will explore all of the nodes. A* runs until there are no more nodes left to explore, meaning there is no path from S to G. BFS will also explore all of the nodes because its purpose is to explore all possible paths, eventually hitting all the nodes when there is no path.

4 Largest Dimension for each search method

Note: DFS was implemented using recursion which slowed down the computation of solving the maze as the size of the maze grew

Largest dimension you can solve using DFS, BFS, and A* at p = 0.3. for time t less than 1 minute
BFS: 165
DFS: 55
A*: 2700

How big can and should your fringe be at any point during these algorithms?

The fringe for DFS at a given iteration will be much larger than the fringe size for BFS because you are only exploring all the children of one side of one node. However, it is not accounting for the unexplored children when finding a path, which leads to a arge a amount of fringes. For BFS, the number of fringes will be lower because it is exploring more paths, leading to less undiscovered children.

5 Strategy 3

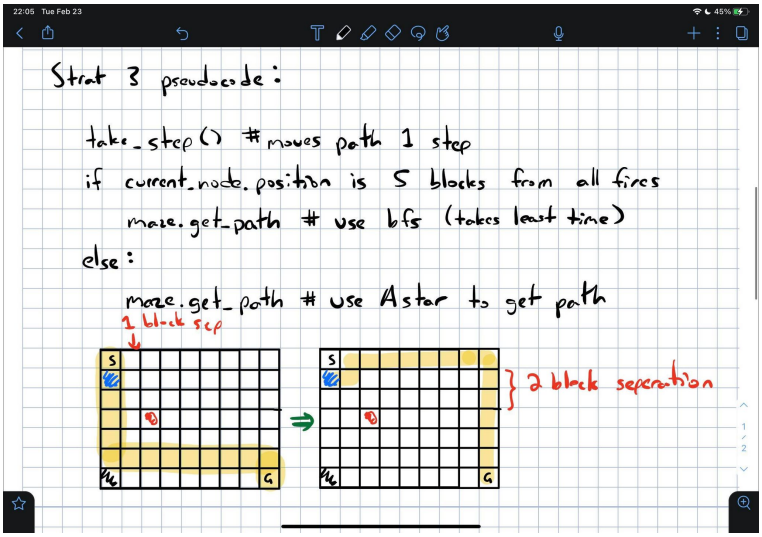

Figure 4: average successes vs flammability for each strategy with their largest dimension at p =0.3

Unfortunately, our team did not get to implement strategy 3 due to our lack of technical knowledge. However, we formulated a pseudo-code that can lead to a more optimal strategy. Essentially, this will work similarly to Strategy 2, but it will keep track of how far the fire is from the player. By looking at the

distance from current position to the closest fire. If the distance is less than 5 blocks, it will use Analyze the time costs of the new strategy compared to other 2, How can you apply the algorithms discussed?

6 Comparison between strategy 1 and 2

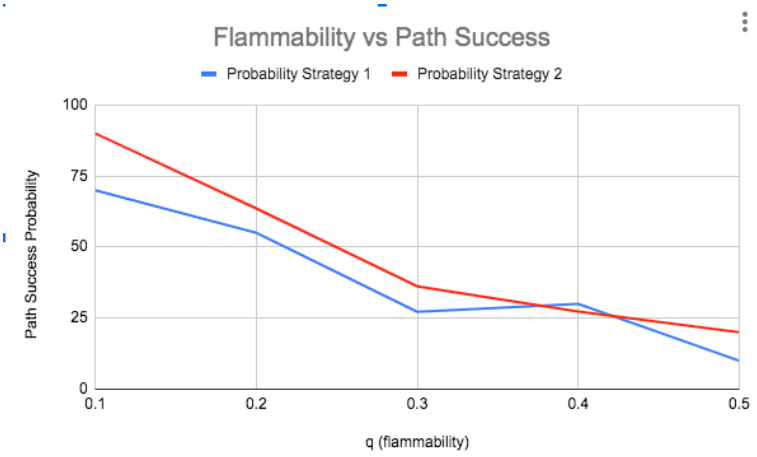

Figure 5: average successes vs flammability for each strategy with their largest dimension at p =0.3

6.1 Which of the two is the superior strategy?

Strategy two will be the superior strategy in finding success for solving the maze. This is because the maze is being updated for every step that the fire takes, recalculating the path every time the grid updates. However, strategy one entails that regardless of the fire, the shortest path is taken, which may lead to the agent dying faster due to encounters with the fire. Even though strategy 1 has access to all the locations of the fire, it can not change its initial course of path, so it never uses that advantage of having access to the potential future states of the fire.

6.2 Where do the different strategies perform the same? Where do they perform differently?Why?

They start to perform the same at 0.4 flammability but for the most part, strategy 2 has a higher chance of finding a path compared to strategy 1. This is because strategy 2 finds a new path whenever the fire interrupts its original path.

7 Unlimited Computational Resources

Given unlimited computational resources, we would calculate the probability of any cell catching on fire depending on the current cells on fire. It would update as the agent moves, allowing the agent to choose a path based on the lowest probability of being set on fire. Storing heuristics and exploring multiple fringes on different paths takes a great amount of storage, so with unlimited storage, we would be able to bypass that problem and the strategy would give the agent most amount of security.

8 Potential Strategy 4

A star with Manhattan because it has space complexity closer to DFS but also has an optimal path due to its capability of having a higher maximum fringe size compared to the other methods. Looks at more nodes at once without backtracking. If we want to reach the goal faster, the Manhattan heuristic is the best option. When we look at the number of nodes expanded because it expands less nodes compared to A* Euclidean.

Conclusion

Working on this project, we have learned to use DFS, BFS, and A* algorithms for path finding strategies. We learned the importance of space complexity and when it is best to choose which algorithm. When simply looking for a yes and no answer for if a solution exists, DFS is the ideal choice. However, when searching for the shortest route, BFS is the perfect choice. When looking for a solution that saves time and space, given a larger input, A* would be the best. A* will maintain a linear number of fringes, while BFS and DFS would increase almost exponentially as the data sets get larger since A* will explore less nodes on average compared to the other search methods. While making the A* algorithm, we explored different strategies by utilizing different heuristics such Manhattan, Euclidean, and Diagonal Distance. In the end we learned through testing that Euclidean distance provided the optimal solution. When coming up with different strategies to surpass a Maze on Fire, we explored different solutions based on the three search algorithms and learned that using BFS was the best approach. Strategy 1 was a primitive solution to the Maze on Fire problem while strategy incorporated the dynamic aspect of the search algorithms where it would recalculate the agent's path as the fire spread, thus, providing a better success rate than strategy 1. Even though we were not able to implement strategy 3 in our code, through our pseudo-code we predict that it is an improvement over the earlier strategies since we not only took different path based on the fire but we also kept a safe distance between the agent and the fire to ensure maximum number of successful runs. With our lack of coding experience, this project was tougher than expected, however it allowed us to explore different paths and learn as we go similar to an AI. While we were not

satisfied with the incomplete final product, we learned a great deal through our struggles which we will make sure to implement in future projects.

Academic Integrity honor statement: On my honor, I have neither received nor given any unauthorized assistance on this examination (assignment).

References

Amato, Christopher. "Graph Search." Ccs.neu.edu, Northeastern University, www.ccs.neu.edu/home/camato/5100/graph$_search.pdf?fbclid = IwAR3IYNjoPhRpQbQrW9VMtY - PKEJrqej8CymrDEwQiZZfymcM7Z0kD_ucl88$.