RUTGERS UNIVERSITY - DEPARTMENT OF COMPUTER SCIENCE

THIS MAZE IS ON FIRE

Khanh Nguyen
February 20, 2021

1 ABSTRACT

In this report, we explore various search algorithms such as Depth First Search, Breadth First Search, and A* with Euclid Distance in both simple and complex way. In section 1, we present how to generate the maze with blocks and path. As part of section 2, results from DFS and how probability of blocks can impact final result of DFS. In section 3, results from analysis and comparison between BFS and A* are seen. Lastly, the largest dimension which each algorithms can withstand based on the probability.

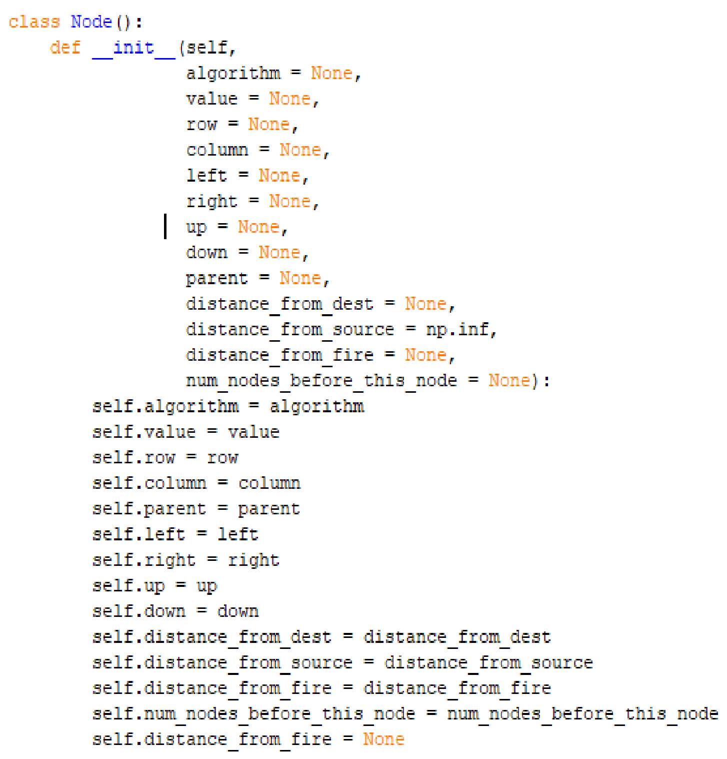
2 MAZE

By using 3 classes, each class represent a different functionality but sharing the same goals of creating a Maze: Node, Graph, and Maze class.

2.1 INITIALIZATION

Node

- Value: Hold agent, blocks, etc values.
- Up,down, left, right: Keep track of current node neighbors
- Distance: Keep track of the distance between the current node with respective node by using heuristic function.

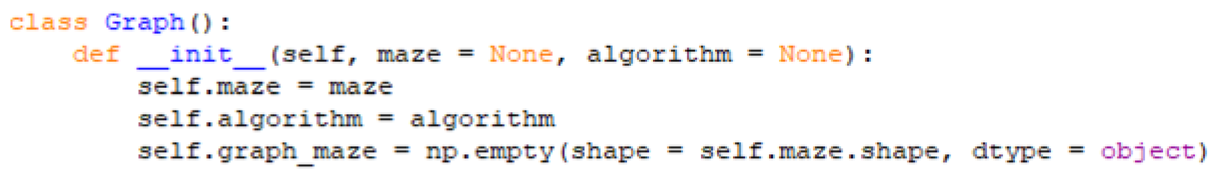


```python
class Node():
    def __init__(self,
                 algorithm = None,
                 value = None,
                 row = None,
                 column = None,
                 left = None,
                 right = None,
                 up = None,
                 down = None,
                 parent = None,
                 distance_from_dest = None,
                 distance_from_source = np.inf,
                 distance_from_fire = None,
                 num_nodes_before_this_node = None):
        self.algorithm = algorithm
        self.value = value
        self.row = row
        self.column = column
        self.parent = parent
        self.left = left
        self.right = right
        self.up = up
        self.down = down
        self.distance_from_dest = distance_from_dest
        self.distance_from_source = distance_from_source
        self.distance_from_fire = distance_from_fire
        self.num_nodes_before_this_node = num_nodes_before_this_node
        self.distance_from_fire = None
```

Figure 1: Class Node.

Graph

- Maze: the initial maze which will be converted into a tree.

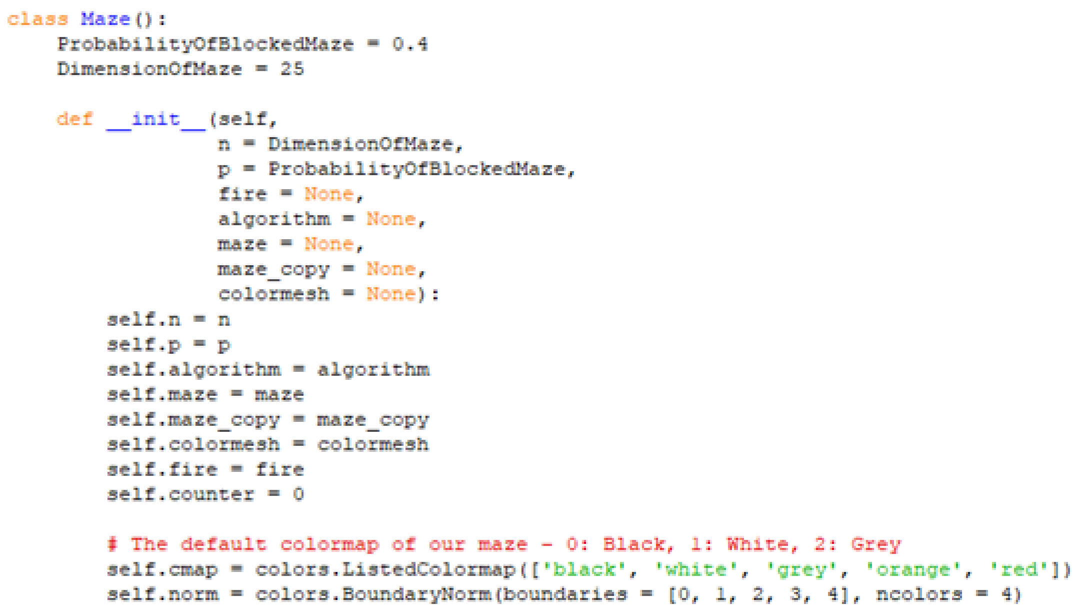


```python
class Graph():
    def __init__(self, maze = None, algorithm = None):
        self.maze = maze
        self.algorithm = algorithm
        self.graph_maze = np.empty(shape = self.maze.shape, dtype = object)
```

Figure 2: Class Graph.

Maze

- P: probability of generating block
- N: Maze dimension's
- Fire: Whether the user wants to integrate the fire into the maze or not.
- Algorithm: To choose the algorithms.



```python
class Maze():
    ProbabilityOfBlockedMaze = 0.4
    DimensionOfMaze = 25

    def __init__(self,
                 n = DimensionOfMaze,
                 p = ProbabilityOfBlockedMaze,
                 fire = None,
                 algorithm = None,
                 maze = None,
                 maze_copy = None,
                 colormesh = None):
        self.n = n
        self.p = p
        self.algorithm = algorithm
        self.maze = maze
        self.maze_copy = maze_copy
        self.colormesh = colormesh
        self.fire = fire
        self.counter = 0

        # The default colormap of our maze - 0: Black, 1: White, 2: Grey
        self.cmap = colors.ListedColormap(['black', 'white', 'grey', 'orange', 'red'])
        self.norm = colors.BoundaryNorm(boundaries = [0, 1, 2, 3, 4], ncolors = 4)
```

Figure 3: Maze Class

2.2 Functionality

Node: Represent the state of the block, unvisited children, visited children, fire, agent(goal) with respective color: black(0), white(1), grey(2), orange(3), red(4) with exception of green as a current path.

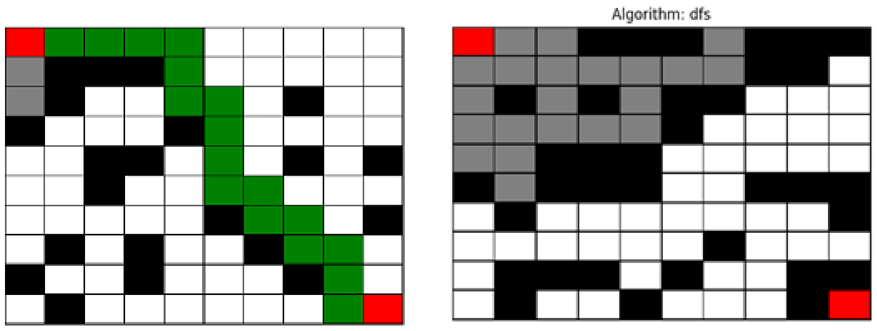
Graph: Convert numpy array to tree of nodes and attach each node 's neighbor (top, bottom, left, right)

Maze: Responsible for generating block based on binominal probability and visualizing the current status of agent

3 DFS Result

The DFS is the fastest among all the three; however, it does not guarantee to have the result as a global optimal path. A trade off between the time complexity and space complexity. To be more explicit, the figure 2 shows that in term of three metrics: number of nodes expanded, maximum fringe size and the final path length, the former metrics will be dominated by DFS and BFS does better in terms of the last two metrics. Moreover, the DFS algorithm can sometimes find the path in one go without backtracking in Fig-3 (a)

Path traversed in DFS algorithm:



(a)Path Found (b)No Path Found

Figure 4: Path traversed by DFS algorithm.

For DFS, the algorithm can be improved better by prioritizing the right direction while choosing the order of loading neighboring nodes into the fringe since the goal is to reach the destination at the bottom right corner. Therefore, it is reasonable to look into first the right and bottom neighbors rather than neighbors – left and up.

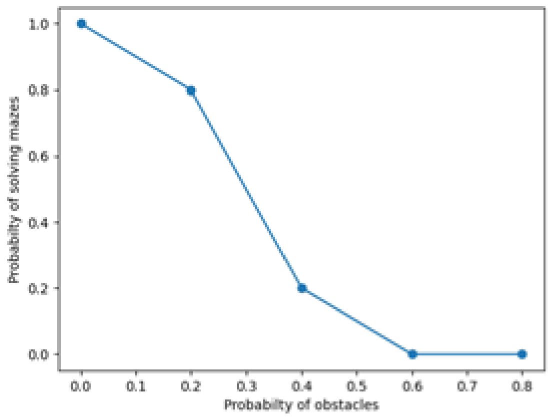


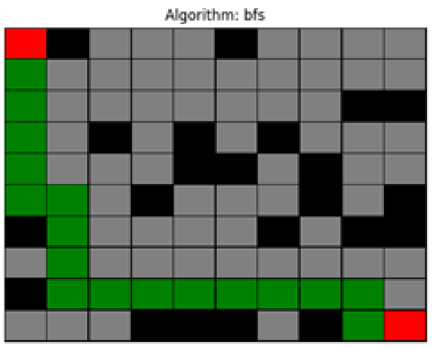
Figure 5: Probability of obstacles vs probability of solving maze

By taking the optimal maze dimension size of 25, we can see that the maze-solvability depends on the probability p. Note that the maximum number of solvable mazes is set as 100. Even though the maze is being solvable by DFS, it does not guarantee the best optimal path for this algorithm as mentioned before.

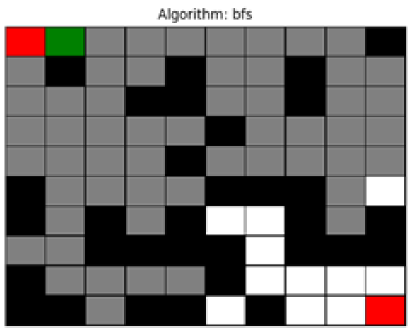
4 Comparison between A* and BFS

Even though the A* and BFS does not dominate in terms of time with DFS, their final result is ought to be the global optimal path. However, the A * algorithm overall has a slightly better than the BFS since BFS is for unweighted tree and A* is for estimating the distance between the destination with it current ‘s child node (weighted tree). Therefore, in terms of the three metrics: number of nodes expanded, maximum fringe size and the final path length, the two former metrics will be dominated by BFS and A* does better with the latter.

Path traversed in A* and BFS algorithm:

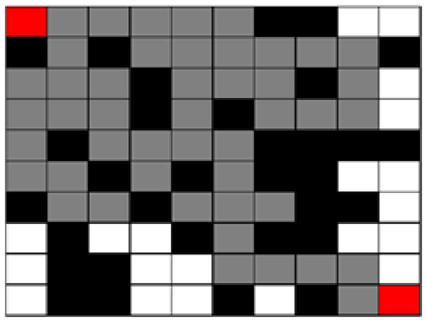


(a)Path Found

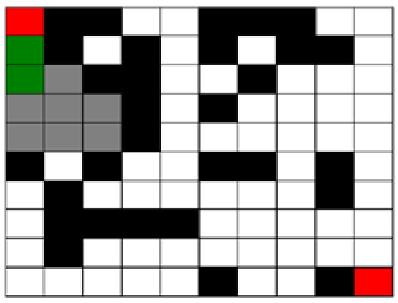


(b)No Path Found

Figure 6: Path traversed by BFS algorithm.



(a)Path Found



(b)No Path Found

Figure 7: Path traversed by A* algorithm.

In the circumstances there is no path from S to G, we can consider A* as an BFS algorithm but with extra cost in calculating the distance between the distance between child, destination, source. The F-7(b) show that if BFS and A* in the same circumstances of not be able to find the path, the three metrics: number of nodes expanded, maximum fringe size and the final path length will be shared equally for both algorithms. However, in the circumstance that they are able to find the path, not only F-7(a) show the different between the fringe and final path between A* and BFS, but also the following figure will show the different between the number of nodes expanded metrics with the predefined settings: 10 dimension, 10 trials / each probability.

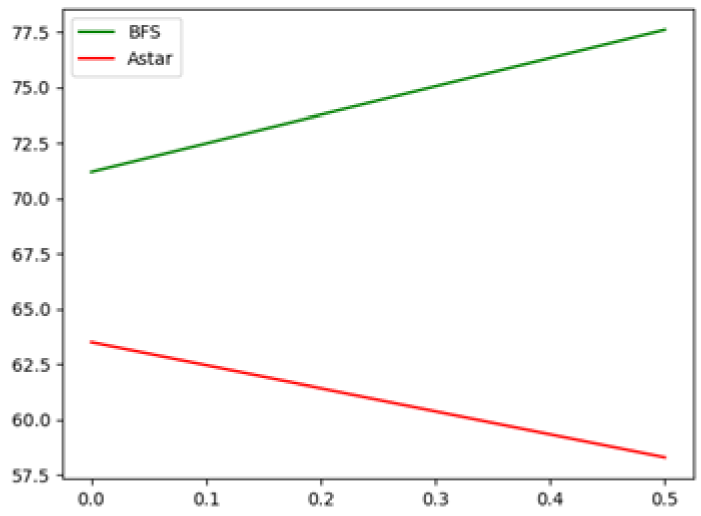
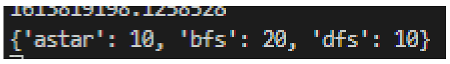


Figure 8: Average expanded nodes between BFS and A*

5 DFS RESULT

By increasing the probability of block sizes, we are increasing our uncertainty which leads to the result of early termination for each algorithm (since each algorithm can stuck in the block before reaching the goals). To be more fairly, it would be recommended for the probability to be 0 for the algorithms to utilize both time and space complexity under the assumption of having early termination. However, we will still develop the algorithm under the assumption that when the algorithm meets unfindable path, it will be terminated without considering any exceptions. Note that the path is considered to be unfindable when the destination node does not have any parent which will be returned when the fringe is out of nodes or when the time reaches 60s. In case we are considering multiple trials, it will maintain the integrity and balance between the algorithm; furthermore, reduce uncertainty.

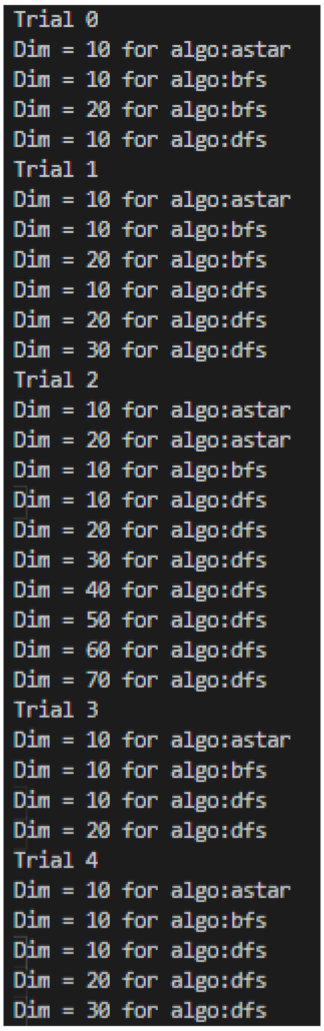
- For one trial



```
1615819198.1258528
{'astar': 10, 'bfs': 20, 'dfs': 10}
```

Figure 9: Finding max dimension among algorithms in 1 trial

- For multiple trial (in this case we set to 5):



```
Trial 0
Dim = 10 for algo:astar
Dim = 10 for algo:bfs
Dim = 20 for algo:bfs
Dim = 10 for algo:dfs
Trial 1
Dim = 10 for algo:astar
Dim = 10 for algo:bfs
Dim = 20 for algo:bfs
Dim = 10 for algo:dfs
Dim = 20 for algo:dfs
Dim = 30 for algo:dfs
Trial 2
Dim = 10 for algo:astar
Dim = 20 for algo:astar
Dim = 10 for algo:bfs
Dim = 10 for algo:dfs
Dim = 20 for algo:dfs
Dim = 30 for algo:dfs
Dim = 40 for algo:dfs
Dim = 50 for algo:dfs
Dim = 60 for algo:dfs
Dim = 70 for algo:dfs
Trial 3
Dim = 10 for algo:astar
Dim = 10 for algo:bfs
Dim = 10 for algo:dfs
Dim = 20 for algo:dfs
Trial 4
Dim = 10 for algo:astar
Dim = 10 for algo:bfs
Dim = 10 for algo:dfs
Dim = 20 for algo:dfs
Dim = 30 for algo:dfs
```

Figure 10: Finding max dimension among algorithms in 5 trial

6 SOLVE FIREBASE

The question asks us to create a structure and solution of a maze which is in essence “dynamic”. Till now the mazes we have been running our algorithms on were static, which meant that the algorithm had the map for the maze, it spent some time computing the best possible path to take and then took that path to reach to the goal, if possible. In this question, we have to come up with a solution that will incorporate the dynamic nature of the maze (fire spreading to the tiles at every turn) and try to find the solution before fire blocks the possible solutions. With the advent of fire, there are some situations which should be kept in mind while making an algorithm. Now, any cell can have three states: ‘open’, ‘blocked’, or ‘on fire’.

Before starting with the algorithm, we made changes to our visualization code by incorporating the probability factor of fire spreading. By applying these rules, we were able to visually see how the fire is spreading through the maze. This helped us see that unlike our pointer (which can only move once per step), fire can spread to at most 2 tiles with probability between 0.5 to 1. Also, this makes the task even more difficult because the fire is spreading more quickly.

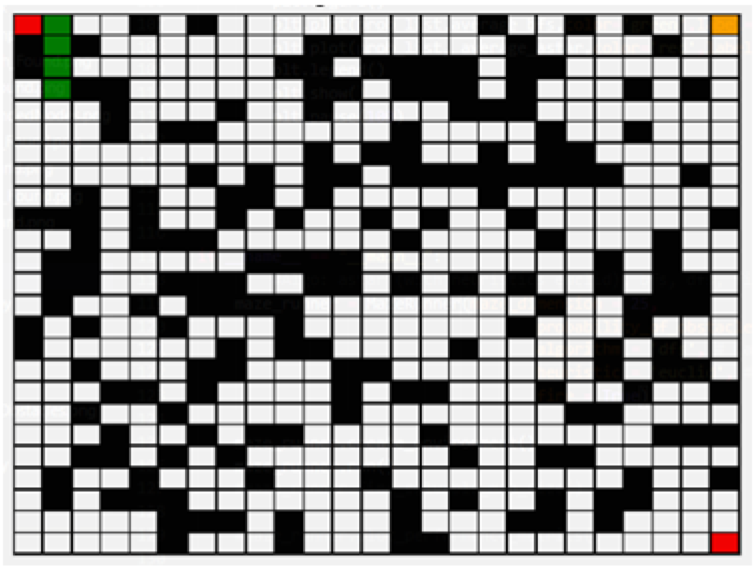


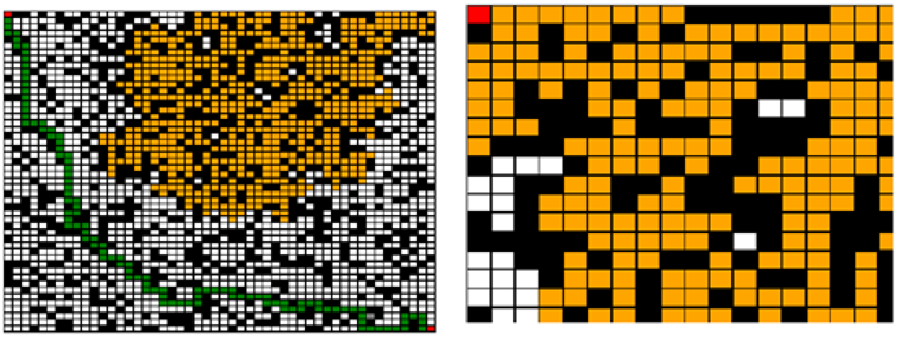
Figure 11: Firemaze (normal status)

With the codes already deployed, we started thinking along the lines of making a custom heuristic to solve this problem. Since A-Star already uses euclidean distance as a heuristic for finding paths, we took the same idea and modified our heuristic to include the distance from all the blocks on fire. Calculating distance from all fire blocks does not solve the problem, so we took the fire block which has minimum distance from our current block. We removed the distance from source from our final heuristic because in this scenario of fire, with fire increasing rapidly, it does not make sense to go back a lot of steps - if you go back 2 steps, the fire has spread to at least 2 tiles. Now, we have two distances with us, distance from closest fire, and distance to destination. We need to come up with a function that includes both to be represented as the final heuristic. The way we finalized the heuristic is as follows:Maze

- Parameters
  - Distance from closest fire: Maximize this distance (to feel safer)
  - Distance to destination: Minimize this distance (to feel closer to solution)
  - Alpha - Value given to change the importance of fire distance
- Cost function
  - Minimize (-Alpha*Distance from fire + Distance to destination)

  - We are able to minimize this function since we are adding the negative of distance from fire to destination.
  - Alpha - Value given to change the importance of fire distance

Now that we have our heuristic, we implemented a DFS + Heuristic approach, in which the next child to be explored will be the child having the best heuristic possible at that step. The priority queue prioritizes children with heuristic values. This allows us to expand the block which presents a safer and closer distance to the solution. In some sense this is a greedy algorithm as we only look at neighbors of our current cell and choose the best neighbor greedily. After running through various iterations, we realized a shortcoming of our algorithm. Since, it chooses its next child greedily (greedy algorithm), there are times when it gets stuck due to a closed block.



(a)Path Found (b)No Path Found

Figure 12: DFS + Heuristic traverse through firemaze

However, this solution is unaccountable for fire's future state. A solution comes from me which neglects the future state of fire would be creating an additional dimension for the agent to move. In this way, the agent only needs to care about the surface which fire are on. (In progress of developing the algorithm)