CS 440
Spring 2021

**This Maze is on Fire**
**Author:** Alden Lu, Haoran Wen.

## 1 Preface

Alden Lu and Haoran Wen worked collaboratively on all parts of this project. The solutions for each question was solved and coded together. Due to each person's time constrains, Alden did a little more work on the code implementations while Haoran did a little more work on the data collection and the report.

"Work on this project is my own" Haoran Wen
"Work on this project is my own" Alden Lu

## 2 Problem 1

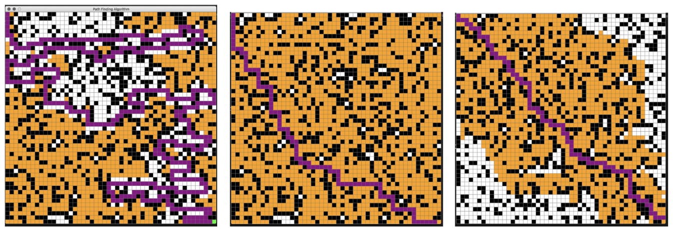

Figure 1: DFS, BFS, A* | 50 and p = 0.3

The program prompt the user for a dimension and object density probability p and it the program generates a maze of dimension x dimension with a the object density of p.

The program uses pygame to generate the maze.

To run the program use 'python Maze.py [dim] [p] [q]' Once the program has successfully be launched. Use the following keyboard commands.

| Command | Action |
|---|---|
| 24em b | BFS |
| d | DFS |
| a | A* |
| SPACEBAR | New Maze |
| Enter/Return | Reset Maze |
| f | Strategy 1 |
| g | Strategy 2 |
| h | Strategy 3 |

## 3 Problem 2

When trying to find out if a path exist in a maze from S to G, it is better to use a BFS algorithm instead of a BFS algorithm. This is because DFS explores each path to the end and returns the first path it finds. BFS on the other hand would explore all paths in paths in the maze and return the most optimal path. This that $O(DFS) < O(BFS)$ as DFS explores less nodes in a graph compared to BFS, unless there is no path from S to G. In this problem we are only interested in determining if a path exist and does not involve finding an optimal path thus DFS will always be more efficient then BFS when solving this problem.

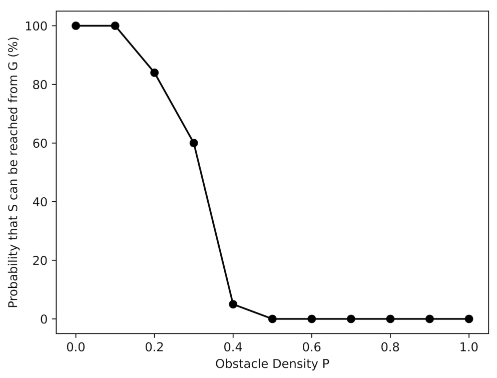

Figure 2: Dim is 50

In figure 2 we plotted the 'probability that S can be reached from G' vs 'object density p'. The probability of that S can be reached from G was calculated by running DFS or BFS on a 50x50 maze 25 times and dividing the the number of times a path exist over total times DFS or BFS is ran (25): $[(\text{paths found})/(25)] * 100$

As shown by the graph, the probability that a path exist from S to G decreases significantly is p is greater than 0.3

## 4 Problem 3

When comparing the number of nodes explored for BFS and A*, if a path exist then A* will usually explore less number of nodes than BFS. If no path exists than A* and BFS will explore the same number of nodes.

As shown by figure 3, the trend lines of A* and BFS shows that the number of nodes explored by A is less than the number of nodes explored by BFS. After a p value of 3, the number of nodes explored by A* and BFS are the same. This aligns with the conclusion made in question 1 and figure 2, as after p value of 3 the probability of a path existing decreases significantly. And when there is no path, A* and BFS will explore the same number of nodes.

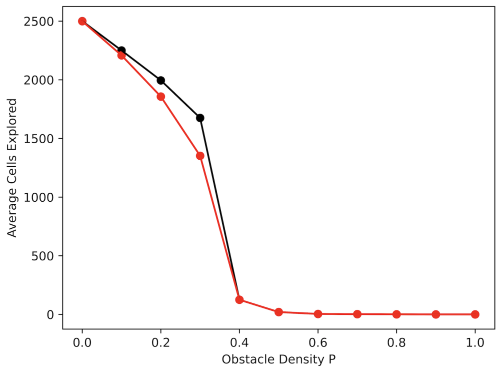

Figure 3: Dim = 50, Black = BFS, Red = A*

## 5 Problem 4

The largest maze that takes i minute to run DFS, BFS, and A* is included below.

From the graph, it shows that BFS and DFS has nearly the same complexity, as they both take 1 minute to run their respective algorithms on a very similarly sized maze. There was a slight variation as BFS is able to run on a slightly larger maze in 1 minute. However this could be due to how we implemented BFS and DFS however the run time of DFS and BFS are the same of O(V+E).
The size of the maze that take 1 minutes to run A* is smaller then both DFS and BFS which lead to the idea that A* in this case has a worst complexity compared to DFS and BFS. One possible explanation for this could be that A* utilizes priority queues while DFS and BFS uses stacks and queues. And since priority queues has greater Big O compared to stack and queues. It is reasonable to conclude that A* has a worst Big O than DFS and BFS depending on the state of the maze.

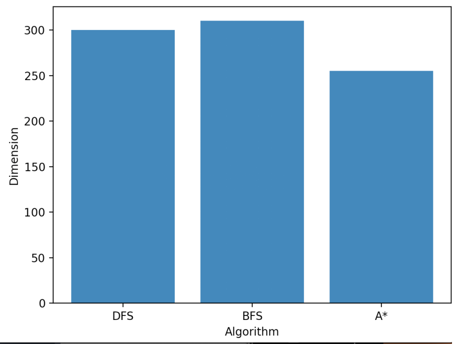

Figure 4: Maximum Maze size that takes 1min to run the respective algorithm

## 6 Problem 5

Generations-in-the-future Maze Estimation A* (GMEA*)

GMEA* is composed of three steps. The first step is measure how dangerous the each particular cell was. In order to accomplish this, we would simulate the fire in a copy of the maze. The maze with 1 fire cell will be passed in and the a function called fire_simulation() will simulate the fire spread a given constant number of times (in this case we made it 60) for dim number of time steps. for each iteration, we collect data on which cells had caught on fire for that iteration. As we iterate, we will sum up the amount of times a particular cell had caught on fire generating what is called a danger_matrix(). The second step of this algorithm is to take this danger value matrix and apply it to a search algorithm. We used A* for this part by applying the danger values to the heuristic in the basic A* algorithm. Thus the heuristic is composed of [danger value + Euclidean distance] By doing this, the A* algorithm will try to stray away from cells that have high danger values and go towards lower danger values while still attempting to compute a short path. The third step is execution. The Agent receives a path and blindly follows it no matter where the fire is currently. This concludes the algorithm. The "predicting future" portion of this algorithm would be the accumulation of the danger values during the fire simulation. The fire simulation simulates the fire in the future, to the A* algorithm an idea of how the fire is going to spread and compute a path based on that.

## 7 Problem 6

To compare strategy 1, 2, and 3, we ran all 3 strategies on a maze with dim value of 50 and p value of 0.3 and calculated their success rate for each flammability value of 0 to 1 at 0.1 increments. The success rate is calculated by: (agent exited maze safely)/(agent died) ∗100. We generated a new maze for every trial and only considered solvable mazes (mazes that has a path from S to G if q is 0). The results for the three strategies are plotted below.

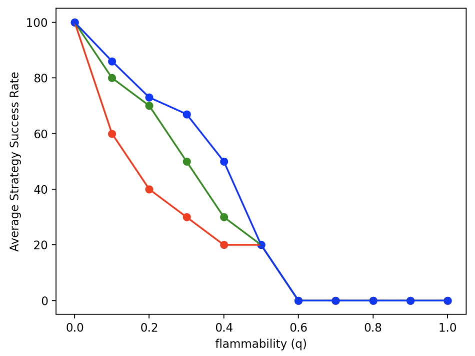

Figure 5: Strategy 1 = Red, Strategy 2 = Green, Strategy 3 = Blue | dim = 50 | p = 0.3

As shown by the graph, for q  4 all 3 strategy has the same probability of success. However for when $0 < q < 5$ Strategy 3 had the highest chance of success while strategy 1 had the lowest probability of success. This is in line with our expectation, as Strategy 1 the agent tries to get to the exist as fast as possible by taking the shortest path from S to G without any consideration for the fire. With the agent running toward to the exit carelessly, it stand to reason it will die the most often using this strategy as the agent can run straight into the fire. Strategy 2 had a better success rate than strategy 1. This is because strategy 2 takes the state of the fire into consideration while trying to reach to exit. At every step the agent is looking for the shortest path to the exit that is not on currently on fire.

While strategy 2 does consider the state of fire when planning a path, it however does not take in consideration how the fire might spread. And since it is not predicting the future state of the fire and thus the agent could take a path that has a high probability to be set on fire.

Strategy 3 is the most successful strategy because it takes the spreading pattern and the fire into consideration and tries to find a path that would avoid the fire by attempting to predict the future state of the fire.

## 8 Problem 7

The obvious improvement to strategy 3 given unlimited computational resources would increase the amount of simulations and the amount of time steps by an infeasible amount. By doing this, we are able to predict how the fire may move in terms of direction and how likely a cell is going to catch on fire based on the simulations. By doing infeasible amounts of simulations, the prediction made based on the probabilities from the simulations should be more and more accurate. To improve upon this, we could also recompute the shortest path like we had done in strategy 2 for every step we take and prioritize those cells over other cells. The algorithm should first compute survivability then compute optimization.

## 9 Problem 8

If the agent has 10 seconds between each step, there are many ways to utilize this resource to improve our strategy. However it would depend on the machine. For the method describe we are assuming that the machine is capable of running in within the given time limit.
This method would be to use this time to simulate the future state of the fire given the current state of the maze and the fire by running multiple simulations on the current maze and fire state and then determine which next step the agent would take that will be the safest and towards the exit. By actively running simulations of the current state of the fire maze we would get a better prediction of how the fire might spread as the agent is building and fine tuning the predictive model at every step.

If the there is resource left other, additionally we could be try to create a predictive model of the danger matrix of how the fire spreads for each 'q' value. At each step, the we can analyze the nature the fire is spreading at and estimate what the q value the fire is. Based on the estimation we could utilize. the appropriate danger matrix for the estimated flammability.

Another way to use the left over resource would be to keep track and calculate how far the fire is from the agent and from the exit and use this as an additional heuristic for the path finding algorithm. Using this information along side the "danger matrix" from GMEA*, When the fire is far away form the agent and the exit, the agent can be more aggressive. If the fire is close to the agent and far away from the exit, the agent can can play it safe and follow a more conservative path. And if the fire is close to the exit, the Agent can choose to be more aggressive and attempt to race the fire to the exit.