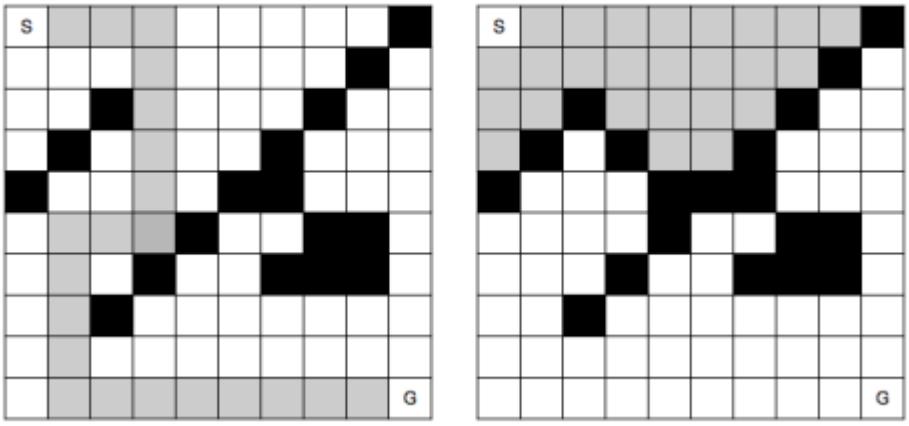
Maze Runner Analysis

CS 440



Muhammad Arif, Cesar Herrera, Jae Weon Kim

Spring 2020

List of Tables

List of Figures

Introduction

This project is intended as an exploration of various search algorithms, both in the traditional application of path planning, and more abstractly in the construction and design of complex objects.

Questions

1. Find a map size (**dim**) that is large enough to produce maps that require some work to solve, but small enough that you can run each algorithm multiple times for a range of possible p values. *How did you pick a* **dim***?"*

2. For $p = 0.2$ generate a solvable map, and show the paths returned for each algorithm. *Do the results make sense? ASCII printouts are fine, gut good visualizations are a bonus*

3. Given **dim** how does maze-solvability depend on p? For a range of p values, estimate the probability that a maze will be solvable by generating multiple mazes and checking them for solvability. What is the best algorithm to use here? Plot *density vs solvability,* and try to identify as accurately as you can the threshold p_0 where for $p < p_0$, most mazes are solvable , but $p > p_0$, most mazes are not solvable

4. For p in $[\,0, p_0]$ as above, estimate the average or expected length of the shortest path from start to goal. You may discard unsolvable maps. Plot *density vs expected shortest path length* what algorithm is most useful here?

5. Is one heuristic uniformly better than the other for running A*? How can they be compared? Plot the relevant data and justify your conclusions

6. Do these algorithms behave as they should?

7. For DFS, can you improve the performance of the algorithm by choosing what order to load the neighboring rooms into the fringe? What neighbors are 'worth' looking at before others? Be thorough and justify yourself.

8. On the same map, are there ever nodes that BD-BFS expands that A* doesn't? Why or why not? Give an example, and justify.

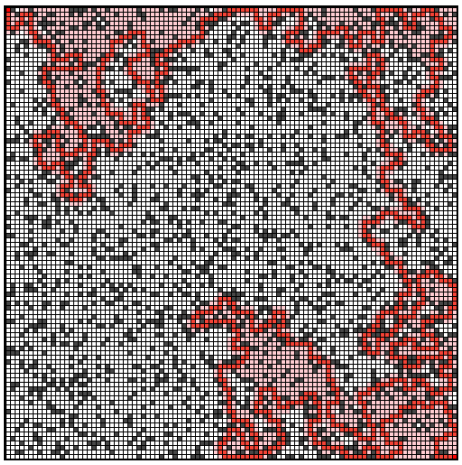
Analysis and Comparison

1) After testing through various map sizes we ultimately decided on a dimension of ***100***. To come to this conclusion we ran our algorithms 1000 iterations on increasing dimension sizes. Through these tests we noticed that solvability remained around the same with increasing map sizes, but the time to solve the mazes increased as map size increased. The results are displayed on Table 1 below.

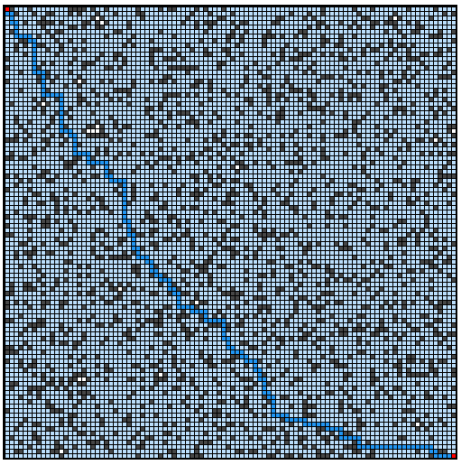
Dimension	**Time:(seconds)**	**Solvable:(out of 1000)**	**Solvable:(percentage)**
10	1.417	832/1000	83.2%
25	6.091	841/1000	84.1%
50	21.914	840/1000	84.0%
100	76.892	861/1000	86.1%
200	313.148	846/1000	84.6%

Table 1: **Increasing Dimension Size Impact**
p *value used:* **0.2**

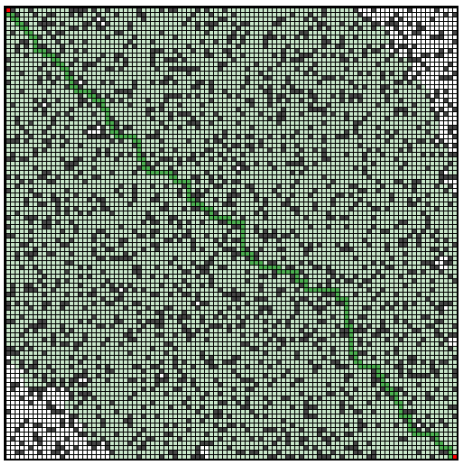
2) We generated a **100 x 100** maze with ***p=0.2*** and ran our different algorithms on it. See Figure:1 Paths below



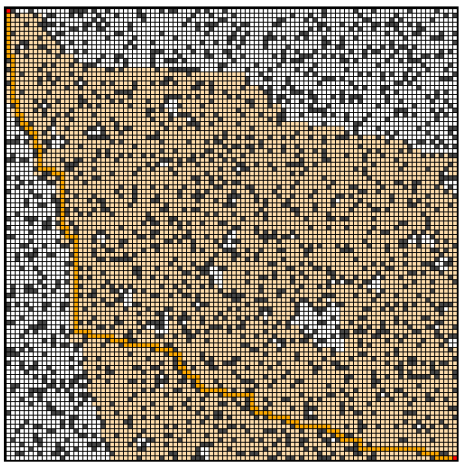
(a) DFS



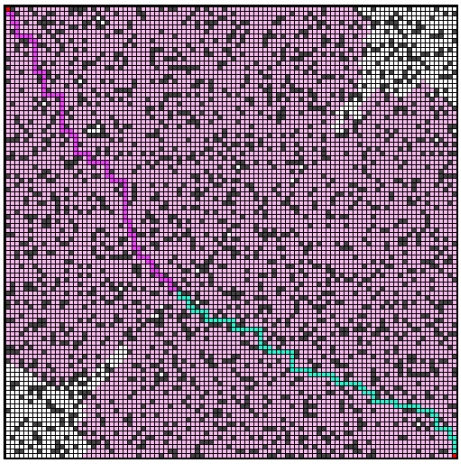
(b) BFS



(c) A* Euclidean



(d) A* Manhattan



(e) BI-BFS

Figure 1: Paths

3) For any dimension size maze solvability remained relatively the same for a specific $\boldsymbol{p}$ value. Thus, to test out the impact of different $\boldsymbol{p}$ values we iterated 1000 times for $\boldsymbol{p}$ values in range of 0 to 1. Our results are graphed below. See Figure: 2 below. Out of the five algorithms we used DFS as it was the fastest performing algorithm . From the data we can conclude that the threshold $\boldsymbol{p_0}$ is around 0.3, because when $\boldsymbol{p} < 0.3$ most mazes are solvable and when $\boldsymbol{p} > 0.3$ most if not all are unsolvable.

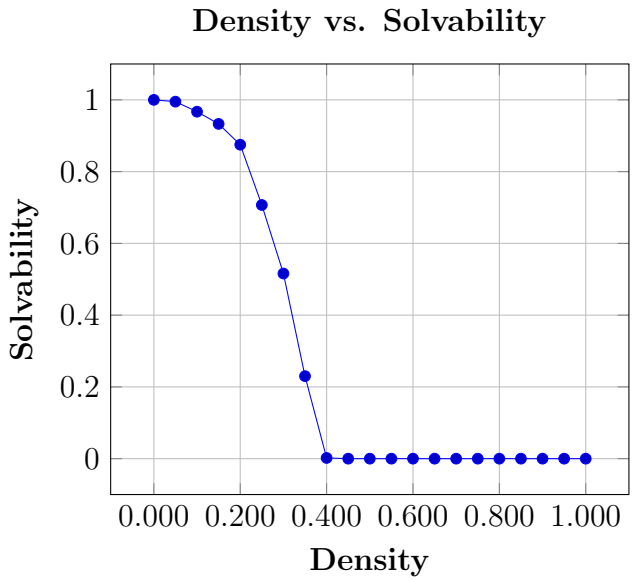


Figure 2: Density vs. Solvability

4) Intuitively, to estimate the average or expected length of the shortest path from start to goal we went for BFS as the algorithm is intended to find the shortest path to the goal node. Thus, our results are shown in Figure: 3 below. ***(Iterated 1000 times.)***

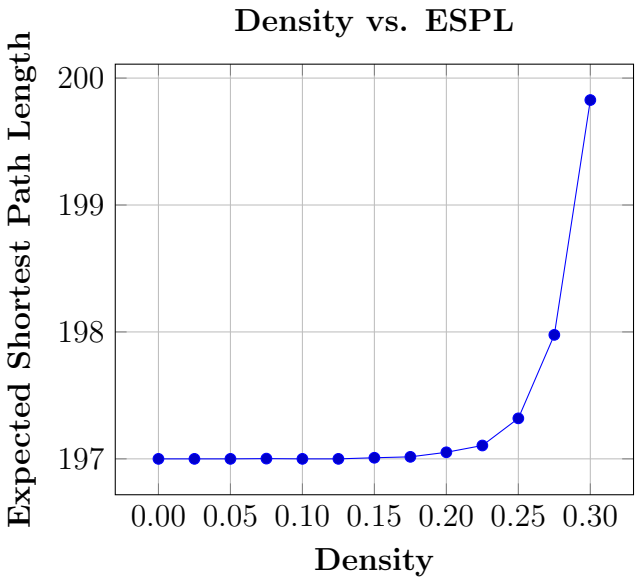


Figure 3: Density vs. Expected Shortest Path Length: BFS

5) Depending on what you define as ***"better"***, one heuristic can be called better than the other. One way to define ***"better"*** can be that we want the least number of cells visited in order to find the path from the starting cell to goal cell. Thus, this is a comparison on which one is ***"faster"***, both computationally and time wise because it theoretically visits less cells if the algorithm is implemented correctly. in Figure:4 below.

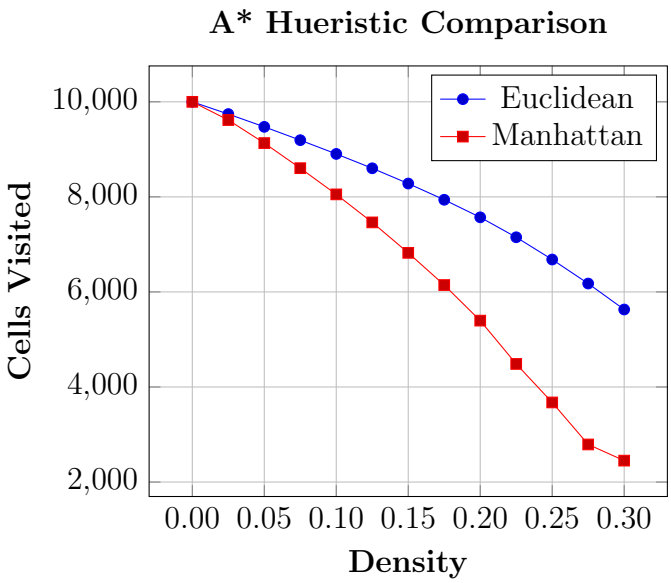


Figure 4: A* Hueristic Comparison: Euclidean vs. Manhattan

6) In analyzing our results, the algorithms implemented do behave as they should. ***See Figure:1 Paths above for visualization.***

7)For DFS, you can drastically improve the difference of the algorithm by choosing what order to load the neighboring rooms into the fringe. For example, our first implemenatation of DFS our algorithm prioritized going up and to the right, however in the second improved DFS, prioritization of the down and right directions actually resulted in shorter path lengths and less number of visited nodes. Thus, in order to quantify this we graphed the average path length for a given density for 1000 iterations. See Figure 5 below.

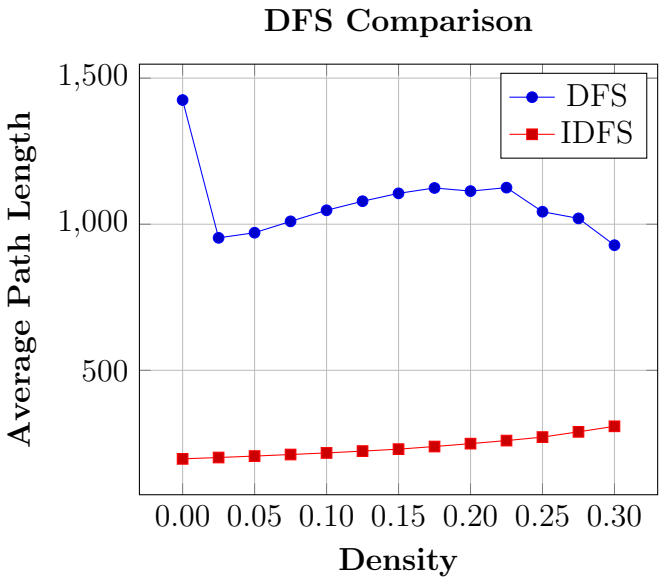
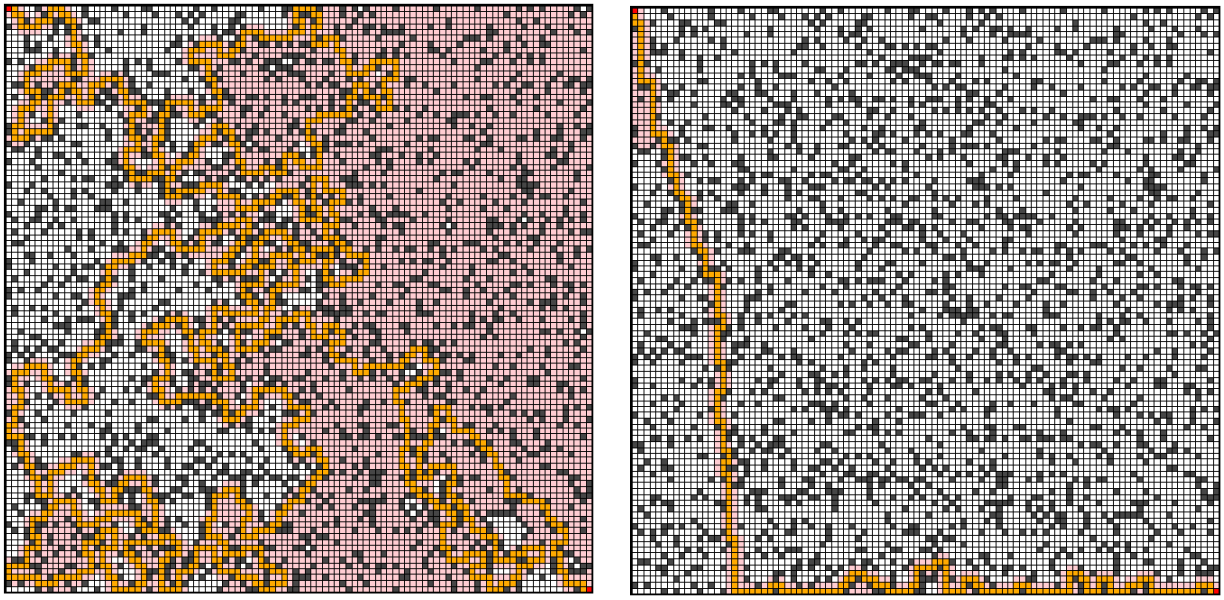


Figure 5: DFS vs IDFS(Improved DFS)

Furthermore, to visualize this difference you can take a look at the following images comparing DFS vs the improved DFS.

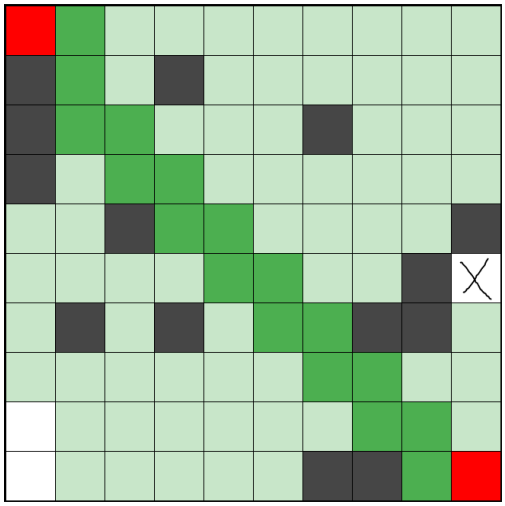


(a) DFS (b) DFS

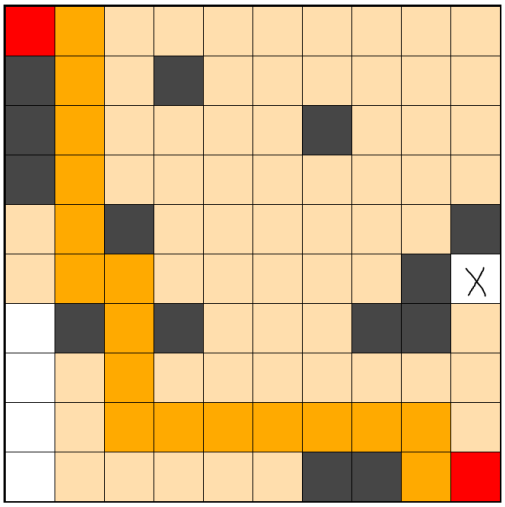
Figure 6: DFS vs IDFS
***Dimension**:100 x 100 **p**:0.20*

7) On the same map there are times that Bidirectional Breath First Search (BD-BFS) expands that A* doesn't. To make the visualization easier, instead of using a dimension of 100 by 100 like in our previous questions for this we can examine a 10 by 10 maze. See Figure: 7 below for the paths given by the three algorithms.

(a) A* Euclidean



(b) A* Manhattan



(c) BD-DFS

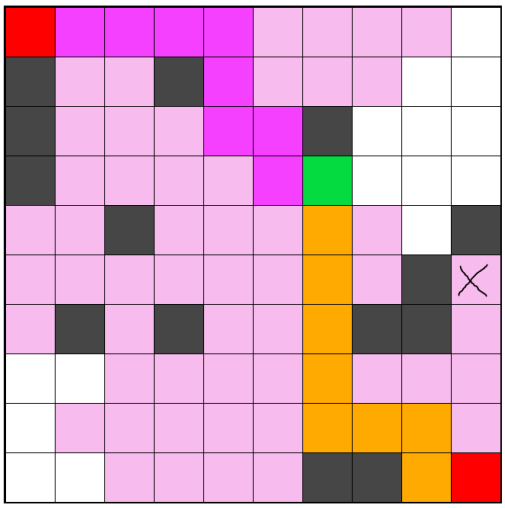


Figure 7: A* vs BD-DFS
***Dimension**:10 x 10 **p**:0.20*

As you can see we have marked an x on the cell in which both the ***euclidean*** and ***manhattan A**** did not visit, while ***BD-BFS*** did. The reason for this is because of the nature of breath-first search. Since, breath-first search works on the premise of discovering cells layer by layer, it "expands" outwards, giving it more of a chance to explore cells that ***A**** doesn't. Given that it also starts from both the start and goal cells, it will only come to an end once it finds the intersecting cells. On the other hand, the way that the ***A**** algorithm uses it's heuristics in combination with the priority queue by adding the distance to the current node and the estimated distance to the destination from said current node, it gives higer preference to the directions in which the estimated distance left is less. Hence, that cell explored by ***BD-BFS*** and not the ***A**** algorithm was due to the estimated cost of exploring it being higher than that of the ones that were explored.