CS 520: Assignment 1 - Search

Karthikeyan Sethuraman (ks1400)
Diksha Prakash (dp978) Arun Sinhmar (as2985)

September 30, 2019

1 Environments and Algorithms

1.1 Generating Environment and Path Planning

Throughout the report and the code, the following conventions have been used:

1. 'Dimension X Dimension' is the size of the maze, referred to as an array
2. 'P' refers to the probability of each block to be blocked i.e. occupied
3. 'Source' refers to the starting point i.e. first block of the maze
4. 'Destination' is the goal i.e. the last block of the maze

The image below shows a sample maze generated by our code. Its representation can be explained as under.

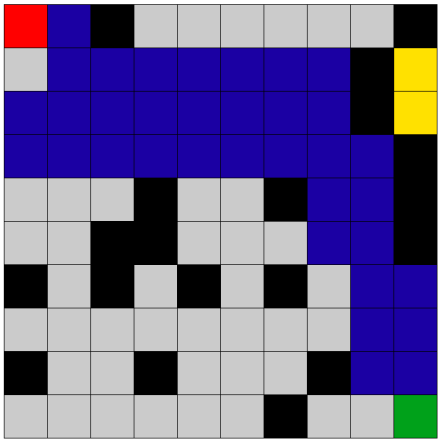

Figure 1: Sample Maze (dimension = 10, p = 0.2)

The interpretation of the same is as under:

1. The start cell has been represented with 'Red' colour (#ff0000)
2. The end/destination/goal cell is represented with 'Green' colour (#228b22)
3. Obstacles or occupied nodes are represented with 'Black' colour (#000000)
4. Visited nodes while the algorithm is executed are represented in 'Silver' colour (#C0C0C0)
5. The unvisited nodes while the algorithm is executed are represented in 'Gold' colour (#FFD700)
6. The final path chosen by the algorithm is represented in 'Blue' colour (#00008B)

2 Analysis and Comparison

2.1 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?

P-value of the Maze Dimension	0.1	0.2	0.3
5	0.199461	0.105143	0.161629
10	0.314976	0.520719	0.422699
15	2.591455	0.607398	0.728766
20	2.838433	0.877254	4.061350
25	1.436684	1.164291	1.078645
30	4.050970	1.927628	5.978942
35	5.196672	2.987280	4.668951
40	4.991331	5.980422	3.710487
45	5.294657	7.045395	3.904780
50	9.232664	8.795992	4.654513
55	8.152723	7.422164	6.525365
60	10.798891	10.327760	7.682490
65	17.173147	10.629892	10.099500
70	16.817784	13.581582	11.749251

P-value of the Maze Dimension	0.1	0.2	0.3
5	0.299239	0.266527	0.376344
10	0.303209	0.439856	0.676274
15	0.844324	0.364152	0.622928
20	1.291621	1.176161	0.796852
25	3.988054	1.638328	0.960815
30	4.460686	4.400870	2.696967
35	3.545713	3.094580	2.335868
40	4.213479	4.457204	3.684950
45	5.821366	5.611300	4.416466
50	8.411106	9.144979	10.592699
55	10.374510	7.692015	6.740987
60	13.367105	9.780986	11.374760
65	13.884366	13.892800	14.361405
70	16.952682	13.984186	16.032017

Figure 2: Output of DFS (L) vs BFS (R)

Figure 2 is the tabulation of the time taken (in ms) for a range of 'p' and 'd' values, for DFS and BFS respectively. Since for p¿0.4, most mazes become unsolvable, we re restricting ourselves to p-values¡0.4. Factors to consider for Picking a Dim:

1. Our Dim needs to be large enough that, there should be more than one solution(paths) that enables us to visualise the difference between results given by various algorithms and compare them.

2. Our Dim needs to be large enough that the expected time taken (for few randomly generated mazes) the solutions significantly differs across algorithms

3. Dim needs to be small enough that we don't run out of memory. Based on the tabulation above, we can estimate approximately the time taken and accordingly decide.

2.2 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, but good visualizations are a bonus.

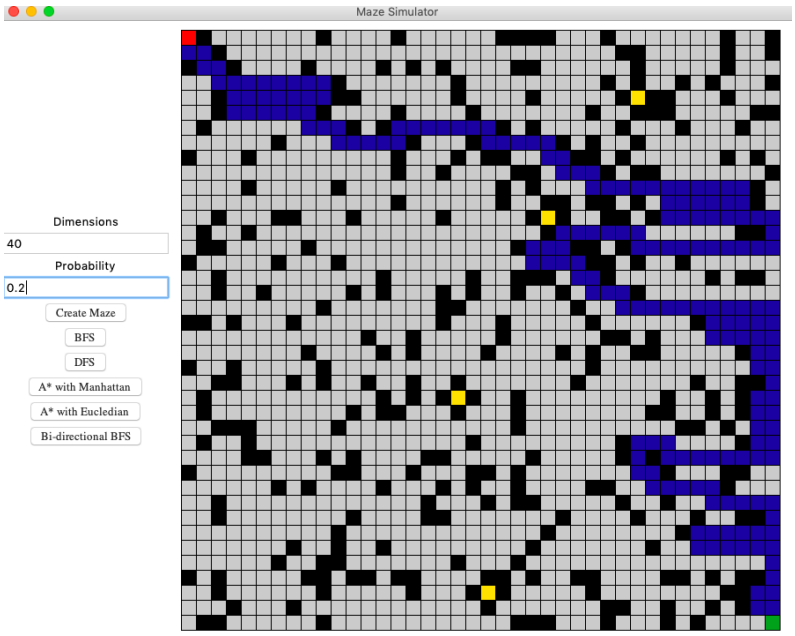

Figure 3: Output of DFS (dimension = 40, probability = 0.2)

The following can be observed from the results generated post the implementation of the algorithms:

1. BFS always returns the optimal shortest path, by exploring more nodes as compared to DFS.

2. DFS returns the non-optimal path. DFS doesn't necessarily give the shortest path, especially as the dimension of the maze increases.

3. Both A* Manhattan and A* Euclidean explore the nodes layer wise by exploiting the heuristics. For this particular maze, we can see that A-star

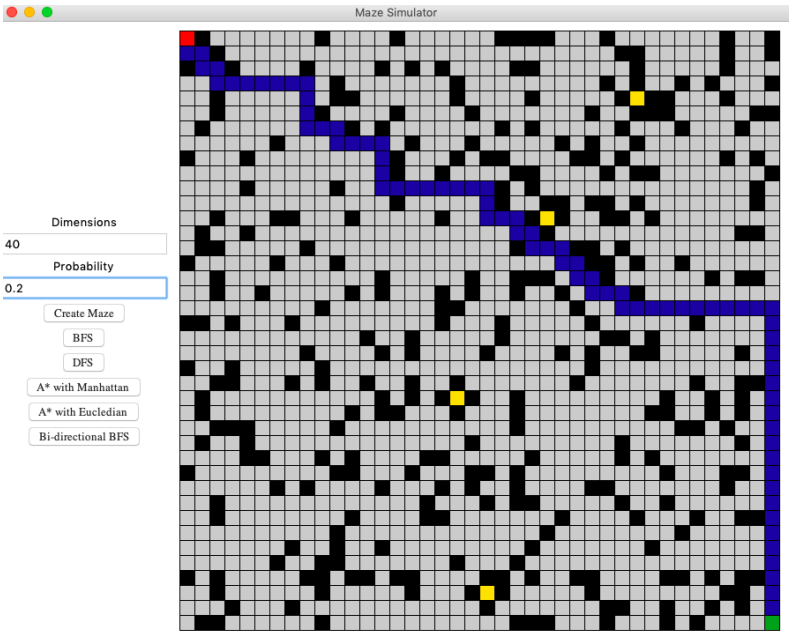

Figure 4: Output of BFS (dimension = 40, probability = 0.2)

Euclidean reaches the solution with lesser number of nodes, although not guaranteeing the shortest path.

4. We can see that Bi-directional BFS is faster than the BFS. We can observe that there are more unvisited nodes in both the directions which enables it to reach the solution faster. In Bidirectional BFS, there are more unvisited nodes, in both directions, as compared to BFS.

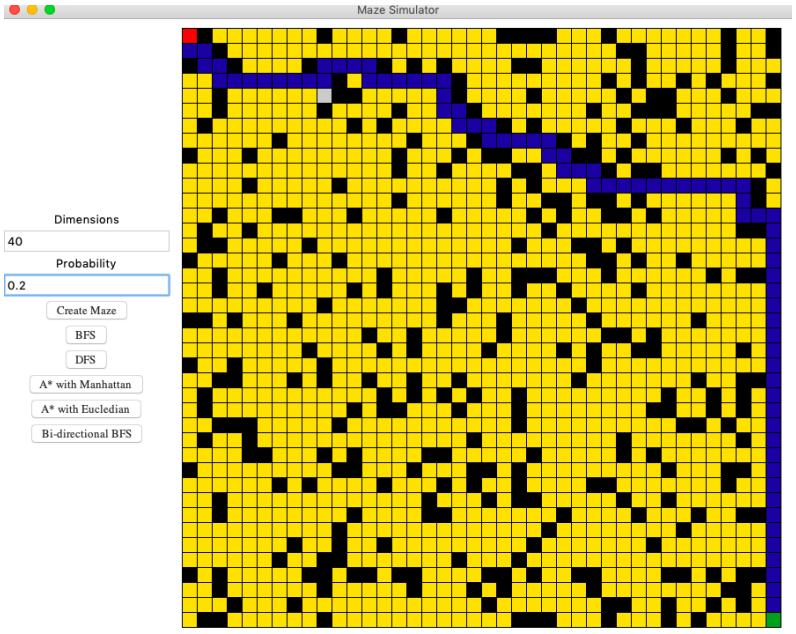

Figure 5: Output of A* Euclidean (dimension = 40, probability = 0.2)

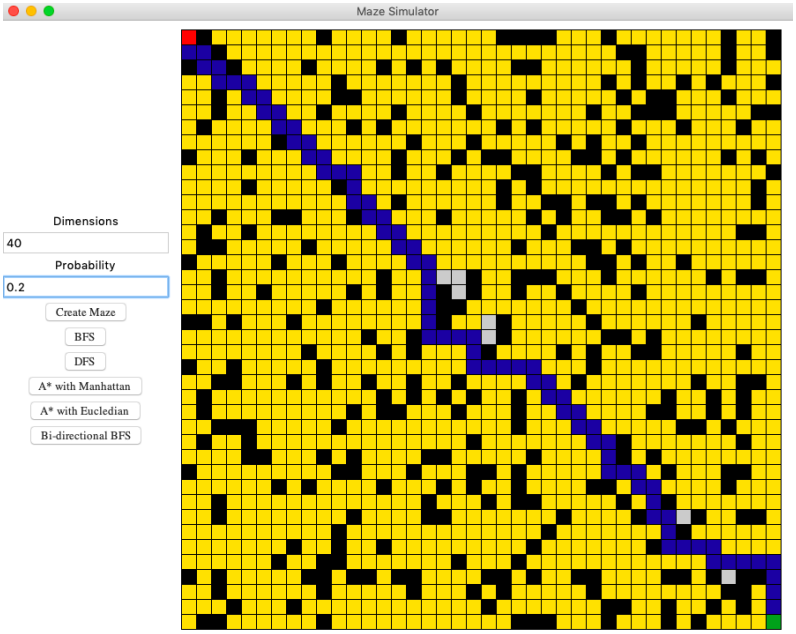

Figure 6: Output of A* Manhattan (dimension = 40, probability = 0.2)

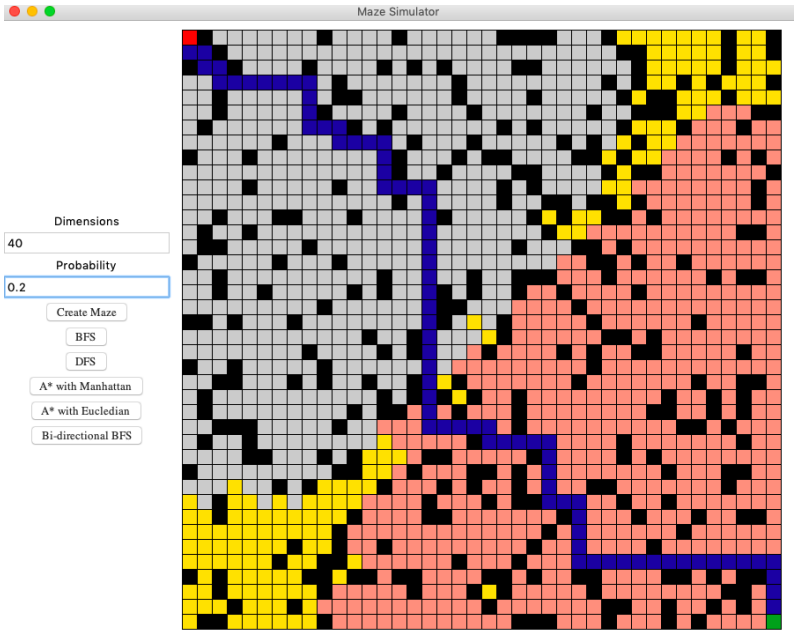

Figure 7: Output of Bidirectional BFS (dimension = 40, probability = 0.2)

2.3 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 p0 where for p ¡ p0, most mazes are solvable, but p ¿ p0, most mazes are not solvable.

Table 8 has the Solvability for a range of P-Values given a Dim value. In order to calculate the solvability randomly 100 mazes were generated and the ratio of solvable to total mazes is reported as solvability of the maze. Figure 9 is the

Dimension	5	10	15	20	25	30
Maze Probability						
0.1	0.99	0.95	0.99	0.92	0.99	0.96
0.2	0.89	0.82	0.81	0.83	0.88	0.81
0.3	0.62	0.59	0.50	0.45	0.58	0.48
0.4	0.36	0.12	0.09	0.06	0.04	0.06
0.5	0.15	0.02	0.00	0.00	0.00	0.00
0.6	0.03	0.00	0.00	0.00	0.00	0.00
0.7	0.00	0.00	0.00	0.00	0.00	0.00
0.8	0.00	0.00	0.00	0.00	0.00	0.00
0.9	0.00	0.00	0.00	0.00	0.00	0.00

Figure 8: Table: Maze P-value VS Solvability for a given dim value

Maze P-value VS Solvability for a given dim value, plotted for a range of dim values from 5 to 30. For mazes with dimensions higher than 5 , we can observe that most mazes are solvable only until a p-value of 0.4. At 0.4, the solvability of the mazes touches zero and remains the same as the P-value further increases.

Best Algorithm to use here: In order to check connectivity between two points in a graph/Tree, DFS is better than BFS in most of the cases. DFS keeps going until the bottom and terminates if there is a path, whereas BFS spends time exploring all the children at a level in order to find the most optimal path. So to just check for solvability, DFS would be a better approach.

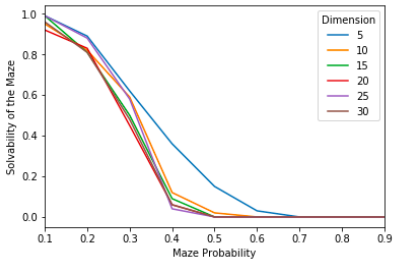

Figure 9: Graph: Maze P-value VS Solvability for a given dim value

2.4 For p in [0,p0] 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?

Below given is the tabulation of mean path length(for 20 randomly generated solvable mazes) across a range of P X D values.
NOTE - Please note that 0 in the below table means that the code couldn't generate a solvable maze in fixed number of iterations. Due to computational limitations, we had a stop point at 20 tries to generate a Solvable maze.
It can be observed that BFS always gives a shorter expected path for all Mazes

P-value of the Maze / Dimension	0.1	0.2	0.3	0.4
5	9.0	9.00	9.00	9.0
10	19.0	19.00	19.00	19.0
15	29.0	29.00	29.91	31.0
20	39.0	39.00	39.83	0.0
25	49.0	49.12	51.00	55.0
30	59.0	59.00	60.09	69.0
35	69.0	69.12	69.44	93.0
40	79.0	79.00	80.17	0.0
45	89.0	89.25	91.55	98.0
50	99.0	99.00	99.91	111.0
55	109.0	109.11	112.60	113.0

P-value of the Maze / Dimension	0.1	0.2	0.3	0.4
5	17.74	13.44	11.00	10.43
10	55.00	39.30	25.15	23.67
15	102.60	67.89	48.45	37.67
20	127.95	103.12	73.67	0.00
25	181.10	120.18	94.78	72.00
30	241.22	180.87	130.64	95.67
35	326.00	201.24	157.00	139.00
40	349.63	240.37	191.67	0.00
45	392.47	269.50	237.36	153.00
50	448.20	319.59	258.82	160.00
55	500.67	327.00	326.80	189.00

Figure 10: Path Length for BFS (L) vs. DFS (R)

within the threshold Po as compared to DFS.

P-value of the Maze / Dimension	0.1	0.2	0.3	0.4
5	9.0	9.0	9.0	9.0
10	19.0	19.0	19.0	0.0
15	29.0	29.0	0.0	0.0
20	39.0	39.0	0.0	0.0
25	49.0	49.0	0.0	0.0
30	59.0	59.0	0.0	0.0
35	69.0	69.0	0.0	0.0
40	79.0	79.5	79.0	0.0
45	89.0	89.0	89.0	0.0
50	99.0	99.0	0.0	0.0
55	109.0	110.0	0.0	0.0

P-value of the Maze / Dimension	0.1	0.2	0.3	0.4
5	9.0	9.0	9.0	9.0
10	19.0	19.0	19.0	0.0
15	29.0	29.0	0.0	0.0
20	39.0	39.0	0.0	0.0
25	49.0	49.0	0.0	0.0
30	59.0	59.0	0.0	0.0
35	69.0	69.0	0.0	0.0
40	79.0	79.5	79.0	0.0
45	89.0	89.0	89.0	0.0
50	99.0	99.0	0.0	0.0
55	109.0	110.0	0.0	0.0

Figure 11: Path Length for A-star Euclidean (L) vs. A-star Manhattan (R)

Since the Backend of Both A-star Euclidean Manhattan is BFS, (the heuristic is use only in deciding the precedence of neighbors to be visited), the expected path length would be the same as BFS.
As Bi-directional BFS is going to give the same path as BFS, we haven't tabulated it here. From Figure 12 for DFS We can infer that as the P-value increases the Optimal path produced becomes longer. This is much clearly visible for higher dimensions.

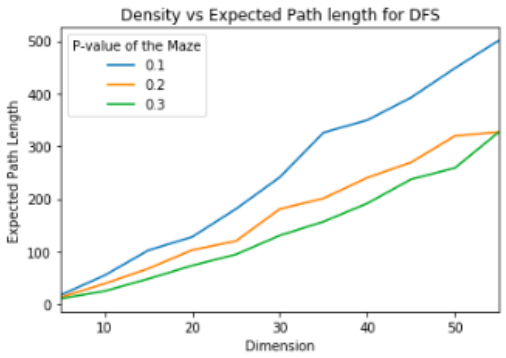

Figure 12: Density vs. Expected Path Length for DFS

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

Heuristics that give better estimate i.e. closest to actual path usually have a better performance as compared to others. According to us, A*- with Manhattan distance heuristic is better than A-star with Euclidean distance heuristic in terms of the number of nodes visited. As the nature of the two heuristics go, Euclidean heuristic is more suited for continuous space as contrasted to the Manhattan heuristic which is more suited to the nature of the maze.
To justify, an analysis is made using the numbers of nodes visited by each algorithm in figure 14. From these results, it can be clearly noticed that **A* Manhattan is exhaustive in node exploration as compared to Euclidean.**
The path length for both the algorithms is the same as indicated in Figure 13 thus, it can't be used as a metric for comparison of the two heuristic.
With respect to the time taken, we don't see a pattern as there's a variation across dimensionality and p-values.

P-value of the Maze Dimension	0.1	0.2	0.3	0.4
5	9.0	9.0	9.0	9.0
10	19.0	19.0	19.0	0.0
15	29.0	29.0	0.0	0.0
20	39.0	39.0	0.0	0.0
25	49.0	49.0	0.0	0.0
30	59.0	59.0	0.0	0.0
35	69.0	69.0	0.0	0.0
40	79.0	79.5	79.0	0.0
45	89.0	89.0	89.0	0.0
50	99.0	99.0	0.0	0.0
55	109.0	110.0	0.0	0.0

P-value of the Maze Dimension	0.1	0.2	0.3	0.4
5	9.0	9.0	9.0	9.0
10	19.0	19.0	19.0	0.0
15	29.0	29.0	0.0	0.0
20	39.0	39.0	0.0	0.0
25	49.0	49.0	0.0	0.0
30	59.0	59.0	0.0	0.0
35	69.0	69.0	0.0	0.0
40	79.0	79.5	79.0	0.0
45	89.0	89.0	89.0	0.0
50	99.0	99.0	0.0	0.0
55	109.0	110.0	0.0	0.0

Figure 13: Path Length in Euclidean (L) vs. Manhattan (R)

P-value of the Maze Dimension	0.1	0.2	0.3	0.4
5	8.00	8.00	8.4	8.0
10	18.11	18.25	0.0	0.0
15	28.22	29.25	36.0	0.0
20	38.40	47.00	58.0	0.0
25	48.33	64.67	0.0	0.0
30	69.80	64.00	67.0	0.0
35	69.25	71.00	123.0	0.0
40	89.86	84.00	0.0	0.0
45	105.33	175.00	0.0	0.0
50	128.00	106.50	0.0	0.0
55	109.50	129.50	0.0	0.0

P-value of the Maze Dimension	0.1	0.2	0.3	0.4
5	8.67	8.86	8.6	8.0
10	21.78	27.00	0.0	0.0
15	60.44	50.00	79.0	0.0
20	73.20	114.00	116.5	0.0
25	115.67	111.00	0.0	0.0
30	149.20	178.67	99.0	0.0
35	310.00	188.00	350.0	0.0
40	279.00	251.00	0.0	0.0
45	505.33	650.00	0.0	0.0
50	645.50	509.50	0.0	0.0
55	866.50	835.00	0.0	0.0

Figure 14: Number of nodes explored in Euclidean (L) vs. Manhattan (R)

P-value of the Maze Dimension	0.1	0.2	0.3	0.4
5	0.221676	4.219110	0.415742	11.232436
10	0.599353	0.569684	0.249207	9.235442
15	0.912142	0.809956	3.723145	0.000000
20	1.097417	1.345098	0.000000	0.000000
25	1.964390	1.969218	0.000000	0.000000
30	2.413034	17.053986	0.000000	0.000000
35	3.354918	3.953934	0.000000	0.000000
40	4.687095	4.924059	0.000000	0.000000
45	6.260633	1.662334	0.000000	0.000000
50	7.202493	1.745045	5.983829	0.000000
55	7.925987	0.000000	0.000000	0.000000

P-value of the Maze Dimension	0.1	0.2	0.3	0.4
5	0.262424	0.147070	0.820593	2.146006
10	0.333031	0.273314	0.165860	0.498414
15	0.580132	0.997615	0.484467	0.000000
20	3.919789	7.229567	0.000000	0.000000
25	2.154460	1.993656	1.994848	0.000000
30	1.890262	9.704399	0.000000	0.000000
35	4.548205	4.788446	0.000000	0.000000
40	5.327861	42.896986	0.000000	0.000000
45	2.258062	7.479906	0.000000	0.000000
50	7.612864	9.972572	0.000000	0.000000
55	9.727716	12.879054	0.000000	0.000000

Figure 15: Time Taken in Euclidean (L) vs. Manhattan (R)

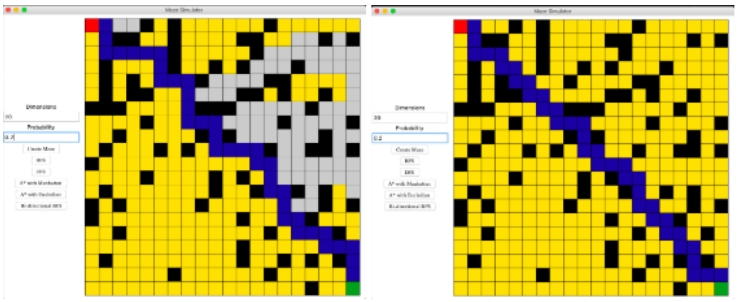

Figure 16: A* Euclidean (L) and A* Manhattan (R)

2.6 Do these algorithms behave as they should?

Yes, the algorithms behave as they should. As expected, DFS is taking a longer non-optimal path, giving a not necessarily shorter solution and takes less time.

BFS is giving the shortest path. BFS comparatively visits more number of nodes and takes more time.
The A-star is making use of Heuristic (both euclidean and Manhattan) for exploration and traversal, covering way lesser number of nodes as compared to BFS. The A*-Manhattan visits lesser number of nodes than Euclidean, and returns optimal short path while taking the least time. Whereas, A*-Euclidean visits more nodes and the returns optimal shortest path by consuming more time.
The Bi-directional BFS is essentially two BFS in action simultaneously from two opposite ends. It covers far nodes than A-star algorithms. It converges to a solution faster than BFS. Also, there are significantly quite a few nodes that remain unvisited in Bidirectional BFS as compared to the regular BFS.

2.7 For DFS, can you improve the performance of the algorithm by choosing what order to load the neighboring

The performance of DFS can be improved by choosing what order to load the neighbors into the fringe. DFS uses stack data structure for development of the fringe. Stack is a Last In First Out (LIFO) data structure thus, if we first push the actions that takes us away from the goal and then push the actions that takes us closer to the goal, the total cost of algorithm can be decreased. Thus, the order of neighbors visited DFS matters.

2.8 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.

Bi-directional BFS (BD-BFS) is essentially simultaneous BFS from two opposite ends. The inherent nature of the BFS makes it expand all the nodes reachable for each layer. Thus, BFS tends to visit each node that is reachable in a given map.
A-star Algorithm follows a heuristic based approach where only those nodes are expanded in which the 'f' function is admissible and optimistic. The whole idea is to find the shortest path in the least amount of time by the guidance of heuristics i.e. it won't expand the nodes for which the heuristic isn't convincing enough. Thus, the objective of A-star isn't to visit all the reachable nodes, unlike BFS.
Figures 17 and 18 describe the number of nodes visited during path finding for both BD-BFS and A-star, independently. Here, the visited nodes are indicated by 'silver' and 'pink' (in BD-BFS) colours whereas the non-visited nodes are indicated in 'yellow' colour.
The span of silver and pink colours is clearly dominant over the span of silver colour in the images of A* algorithm implementation.

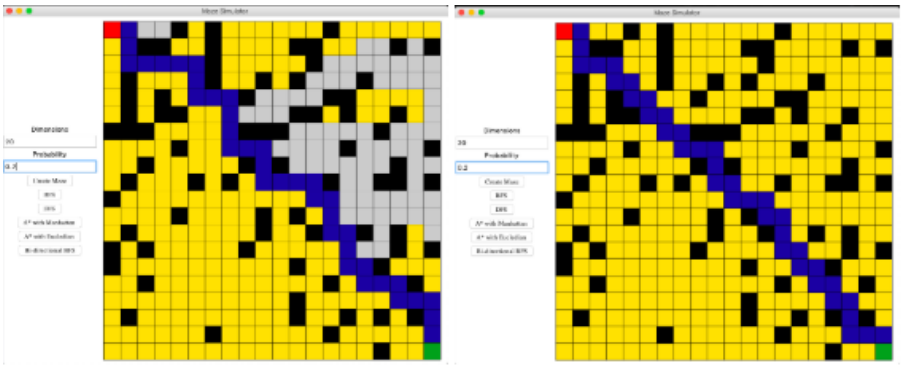

Figure 17: A* Euclidean (L) and A* Manhattan (R)

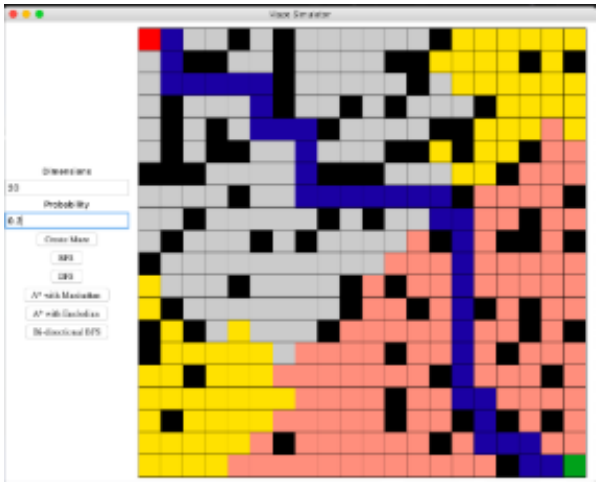

Figure 18: Bi-directional BFS

Thus, it can be concluded that BD-BFS expands the nodes that A-star doesn't. This can also be justified using the graph below which is a plot between the number of nodes visited in the two algorithms.

2.9 Bonus: How does the threshold probability p0 depend on dim? Be as precise as you can.

The threshold probability 'p0' relates to dimension via 'solvability'. Solvability is defined as the ratio of the number of solvable mazes to the total number of mazes. In the 3rd sub-part of Question 2, we have plotted density vs. solvability graph to obtain 'p0'. We found the threshold probability to be 0.4 for n=100. Let's assume that we generate only 10 mazes with p=0.4 and if all of them are

solvable, it changes 'p0',clearly indicating the relationship between the two. Thus, 'p0' relates to dimension via solvability.

3 Generating Hard Mazes

3.1 What local search algorithm did you pick, and why? How are you representing the maze/environment to be able to utilize this search algorithm? What design choices did you have to make to make to apply this search algorithm to this problem

Out of the many local search algorithms available for use, we chose 'Simulated Annealing' as our algorithm. This algorithm performs significantly better than the 'Hill Climbing' algorithm which has the flaw of getting stuck in a local maxima, while not discovering the global maxima. This problem is avoided by introducing randomness in the algorithm.

3.2 Unlike the problem of solving the maze, for which the 'goal' is well-defined, it is difficult to know if you have constructed the 'hardest' maze. What kind of termination conditions can you apply here to generate hard if not the hardest maze? What kind of shortcomings or advantages do you anticipate from your approach?

Our planned algorithm terminates on 2 conditions:

1. Temperature falls below the threshold value
2. Inability to generate solvable mazes even after 5000 iterations

The advantages associated with this method is that on decreasing the temperature slowly enough, the optimal state is reached. The nearness to the global optimum increases as the number of iterations performed increases. However, this is also a potential disadvantage as the algorithm has to be run for a long time to generate the global optima.
In order to make sure we generate a harder maze we can do a BFS and look at the length of the optimal path generated and compare it against the expected path length for that combination of probability-dimension mazes (which we have already tabulated in the previous question).

4 What If The Maze Were On Fire?

This problem describes a dynamic maze scenario. Till now, in the previous problems, we had a map for the maze and the algorithm spent some time computing best possible path.
Here, the idea is to design a solution to incorporate the dynamic nature of the maze as the fire spreads to the tiles at every turn.
Thus, the preliminary change is the introduction of three states as opposed to the initial two state ideology. Thus, the first step is to visualize the fire spreading by incorporating the probability factor for the spread of the fire. This visualization helps us to infer that the fire spreads more rapidly than the movement of the pointer in the maze to reach the goal from the start point.
As discussed, we have three factors to take into account, our aim is to stay at a safer spot at given instant of time. A safer spot would correspond to nearness from the goal and the maximum distance from the fire. Thus, the heuristic is based on distance to the solution and distance from the fire. At the same time since the fire is spreading rapidly, we need to find the most optimal and shortest path. Thus, the proposed method is a mixture of DFS combined with a heuristic.
After analysis we can infer that the algorithm is greedy in its nature and tends to be caught up in fire at times instead of just arriving at the goal.