CS 520: Assignment 1 - Path Planning and Search Algorithms

Haoyang Zhang, Han Wu, Shengjie Li, Zhichao Xu

October 7, 2018

1 Introduction, group members and division of workload

In this group project, apart from implementing DFS, BFS, A^* with Euclidean Distance Heuristic and A^* with Manhattan Distance Heuristic, we also did some modification to these algorithms for different performance out of our personal interests.

Name RUID	Workload
Haoyang Zhang 188008687	Implemented DFS, Iterative Deepening DFS, BFS, Bidirectional BFS, Bidirectional A^*, Simulated-Annealing-Based Beam Search and the visualization of maze. Modified DFS to make it able to return optimal path. Added Last-in First-out feature to A^*. Managed to combine Simulated-Annealing-Based Beam Search with Genetic Algorithm. Ran tests for DFS and BFS in question 10. Finished half of the writing of report for part 2.
Han Wu 189008460	Wrote python scripts for testing the performance of algorithms. Combine the data and generated figures for question 1, 2, 4 and 5. Finished the writing of report for question 1, 2, 4 and 5.
Shengjie Li 188008047	Implemented A^* with Euclidean Distance Heuristic, A^* with Manhattan Distance Heuristic and Genetic Algorithm. Ran tests for A^* with Euclidean Distance Heuristic and Manhattan Distance inquestion 10. Finished the format design of whole report using LaTeX.
Zhichao Xu 188008912	Wrote python scripts for testing the performance of algorithms. Combine the data and generated figures for question 3, 6 and 7. Finished the writing of report for question 3, 6 and 7. Suggested an improvement of A^* using Chebyshev Distance.

2 Part 1: Path Planning

1 For each of the implemented algorithms, how large the maps can be (in terms of dim) for the algorithm to return an answer in a reasonable amount of time (less than a minute) for a range of possible p values? Select a size so that running the algorithms multiple times will not be a huge time commitment, but that the maps are large enough to be interesting.

In order to find a reasonable size of the maze, we tested the running time of the algorithm. For each size, we generated 10 mazes and run different algorithms on these 10 mazes. We recorded the average time each algorithm needs to return an answer. The 10 mazes in each test ccould be either solvable or unsolvable. We set p equals to 0.3 and executed the experiment.
The results are shown below as Table 1:

		Size					
		100	200	400	800	1600	3200
Time(s)	DFS	0.01942	0.07087	0.28435	0.93919	4.5064	10.83366
	BFS[2]	0.07342	0.33946	1.73646	6.40967	25.69253	91.73234
	A^* Euclidean	0.07811	0.41706	1.62604	5.97804	25.27724	100.37974
	A^* Manhattan	0.06093	0.29222	0.88752	3.63068	11.147	63.8882
	BFS[1]	254.36093 (size=30)					

Table 1

DFS was the default setting. BFS1 was the default setting. However, it took too much time. When the size was 30, it took more than 250 seconds to return an answer. So, we changed the settings a little to make BFS acceptable. Here came BFS2. In BFS2, checkFringe=True. The others were also False. A* Euclidean means that A* algorithm used Euclidean Distance as the heuristic function. A* Euclidean means that A Star algorithm used Manhattan Distance as the heuristic function.

In the table, we could see that when size becomes large, the average time of returning an answer (whether solvable or unsolvable) increases. BFS2 and A* Euclidean are often the most time-consuming algorithms. We need to run the algorithm repeatedly for the purpose of validations. In order to make it faster in the following test, we chose 200 as the default size of our maze.

2 Find a random map with $p \approx 0.2$ that has a path from corner to corner. Show the paths returned for each algorithm. (Showing maps as ASCII printouts with paths indicated is sufficient; however 20 bonus points are available for coding good visualizations.)

In this question, lots of algorithms have been applied to solve the same maze, because BFS, DFS and A^* have several variants. We have implemented them and compared differences between all the variants.

Starting with a small maze whose size is 16×16 as Figure 1 shows.

The most common DFS opens 43 blocks and its maximum fringe size is also 43. Coincidentally, it returns a path whose length is still 43. Apparently, its path is not optimal. See Figure 2.

Then we have tried Iterative Deepening DFS which opens 2753 blocks (calculated accumulatively). Its maximum fringe size is 34 and path length is 33. Notice that IDDFS do not return an optimal path(explained in "Modify DFS to Return an Optimal Path.html")! See Figure 3.

In order to return an optimal path, IDDFS and checkFringe should be set True. CheckFringe means keeping that blocks in the fringe always have the shortest path seen so far, but if we find a new path to this block, which is shorter, update no matter whether this block has been added to closed set or not. In this case, 8453 blocks have been opened, and the maximum fringe size is 29. The optimal path length is 31. See Figure 4.

Also, there is another way to find an optimal path. Set keepSearch is True. In this case, when DFS find a path, save it rather than return. Keep searching for a shorter path. However, it takes too much time to return (A small comparison is recorded in the table.).

See Figure 5.

Hence, we can use checkFringe to accelerate keepSearch. By doing so, it opens only 1118 blocks (compared with 8453 of IDDFS and checkFringe), and its fringe size is at most 35. Its path is also optimal. See Figure 6.

There are many other ways to improve DFS, and one of them is quickGoal, which means return the path the time trying to add the Goal into fringe, instead of the time popping the Goal out of fringe. In this way, DFS opens 43 blocks, but the maximum fringe size is 42 compared with 43 of common DFS. The path is the same as the one common DFS returned. It is interesting to find quickGoal do not improve DFS much. However, it is because DFS's fringe is a LIFO queue. In this maze runner, it will not make any difference greater than 3. See Figure 7.

Another method is randomWalk. In common DFS, the priority of 4 directions is Left > Down > Right > Up. But randomWalk will randomly pick up a direction between left and down and between right and up. Notice that left and down is still preferable to right and up. Sometimes, it works. For example, in this case, the fringe size is 35, and block count is 33. The path is also shorter than common DFS, which is 33-block length. See Figure 8.

Yet it will not be a good idea to totally randomly pick up a direction. Here is the case. Set randomWalkPlue true. It opens 70 blocks. Fringe size goes up to 44. And the path is 53-block length. The reason is that totally random will cause agents go around in a small region, which is contradictory to the idea of DFS. Also, since the Start and the Goal locate at the ends of diagonal, the whole path direction should be southeast.

Besides, chechFringe itself can also make the path a little bit shorter. Setting checkFringe true returns a path whose length is 39 shorter than common DFS, though it is not the optimal. See Figure 9.

Except checkFringe, we can also use distinctFringe to stop duplicated block being pushed into fringe. But once this block is closed, it will never re-open even if we might find a shorter path. In this case, the fringe size, block count and path length is the same as common DFS, but we can see the difference later. See Figure 10.

Certainly, we can combine those options, here is an example. Set quickGoal, randomWalk and distinctFringe true. The block count, path length and fringe size are 34, 33 and 31. See Figure 11.

The most common BFS opens 113708 blocks, and its fringe size is 27898, because there are lots of duplicated blocks in its fringe. But it returns an optimal path. See Figure 12.

To fix this issue, we can use checkFringe. Now it opens only 210 blocks and its fringe size decreases to 16. The path is absolutely optimal.

Also, Bi-Directional BFS can be helpful, whose fringe size is 2468 compared to 27898 of common BFS. Furthermore, it only opens 3528 blocks. Since it is a variant of BFS, the path must be optimal. When there is no path, Bi-Directional search usually returns faster than common ones because an empty fringe of either side, the Start or the Goal, indicates there is no path. See Figure 13.

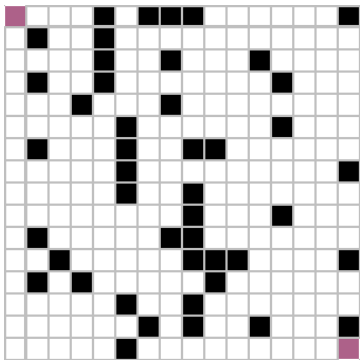

Figure 1

distinctFringe 43 43 43.png
distinctFringe 43 43 43.bb

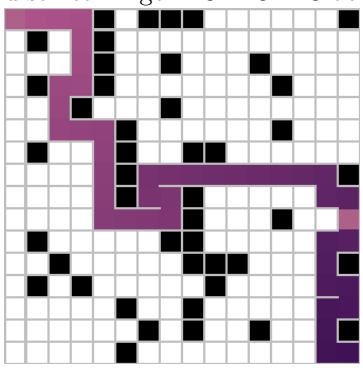

Figure 2

IDDFS 2573 33 34.png
IDDFS 2573 33 34.bb

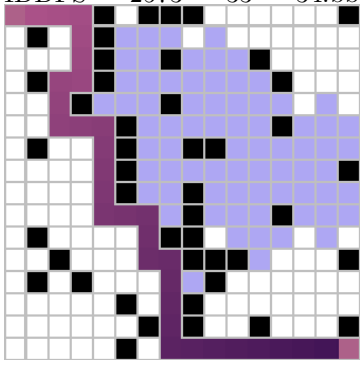

Figure 3

Now let us examine the performance of quickGoal of BFS. 93752 blocks have been opened, which is much fewer than common BFS. The fringe size also dramatically decreases to 19956. Because the fringe of BFS is a FIFO queue, quickGoal will return the path much earlier. See Figure 14.
. However, randomWalk and randomWalkPlus will not work regarding of BFS. Due to the fact that BFS is searching blocks level by level. The block count, path length, and fringe size of randomWalk is 113708, 31, and 27898. 113707, 31 and 27898 is the same data of randomWalkPlus. See Figure 15 and Figure 16.

Again, we can use many options at the same time. Here is an example. Set BDBFS, quickGoal and randomWalk true. The block count, path length and fringe size becomes 151 31 and 25. See Figure 17.

Speaking of A*, there are mainly 3 dimensions to improve it. The heuristic function, Bi-Directional A* and another kind of priority queue. See Figure 18.

To begin with, we have chosen 3 distance function to be compared: Euclidean, Manhattan, and Chebyshev. All of them return an optimal path. But the preformation of Manhattan $>$ Euclidean $>$ Chebyshev. The reason is Manhattan is more closed to the real cost than Euclidean and Chebyshev.

In addition, we can also implement Bi-Directional A*. However, BDA* may not return an optimal path. Because the meet point has shortest path to the Start and Goal, yet the shortest path from the Start to the Goal may not go through the meet point.

Speaking of priority queue. 2 elements with the same priority can be LIFO or FIFO. The FIFO ones are more common, but we also have tried a LIFO one. LIFO ones preforms much like DFS, especially the maze is nearly empty (no walls).

All the comparison is recorded in the table. Notice that BDA* LIFO with Manhattan is extraordinary except for non-optimal.

3 For a fixed value of dim as determined in Question (1), for each $p = 0.1, 0.2, 0.3, ..., 0.9$, generate a number of maps and try to find a path from start to goal - estimate the probability that a random map has a complete path from start to goal, for each value of p. Plot your data. Note that for p close to 0, the map is nearly empty and the path is clear; for

IDDFS checkFringe 8453 31 29.png IDDFS checkFringe 8453 31 29.bb

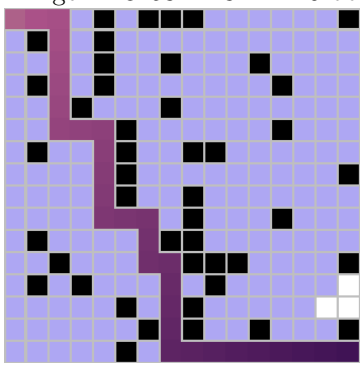

Figure 4

keepSearch checkFringe 1118 31 35.png keepSearch checkFringe 1118 31 35.bb

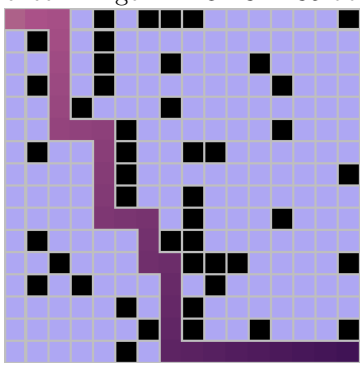

Figure 5

quickGoal 43 43 42.png quickGoal 43 43 42.bb

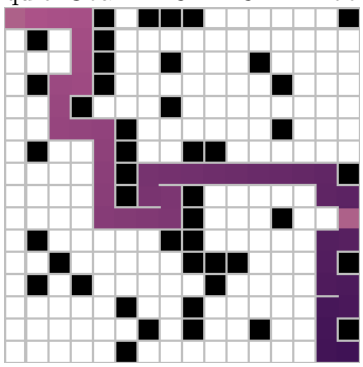

Figure 6

randomWalk 33 33 35.png randomWalk 33 33 35.bb

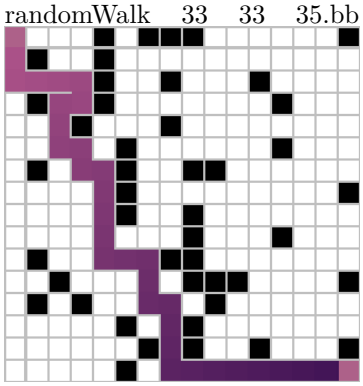

Figure 7

randomWalk Plus 70 53 44.png randomWalk Plus 70 53 44.bb

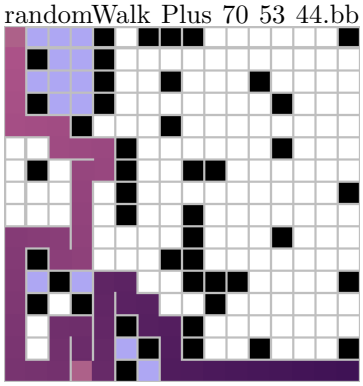

Figure 8

checkFringe 41 39 35.png checkFringe 41 39 35.bb

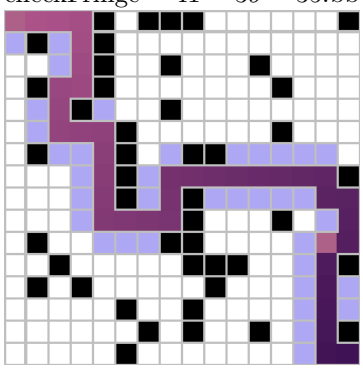

Figure 9

distinctFringe 43 43 43.png distinctFringe 43 43 43.bb

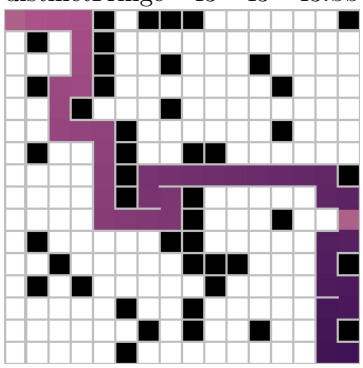

Figure 10

quickGoal randomWalk distinctFringe 34 33 31.png quickGoal randomWalk distinctFringe 34 33 31.bb

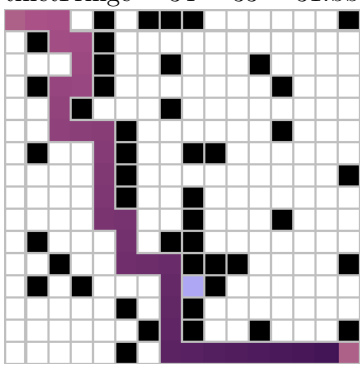

Figure 11

randomWalkPlus 113706 31 27898.png randomWalkPlus 113706 31 27898.bb

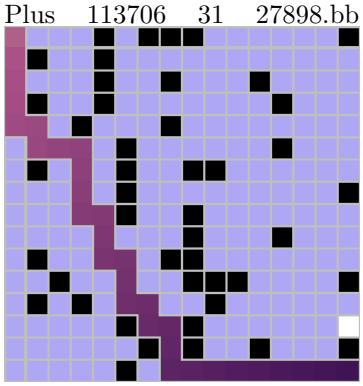

Figure 12

BDBFS 3528 31 2468.png
BDBFS 3528 31 2468.bb

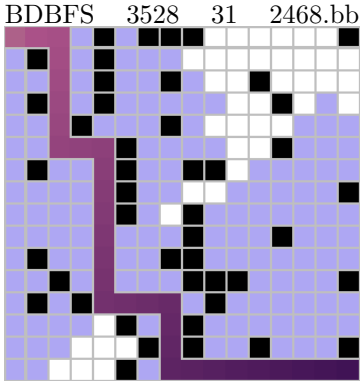

Figure 13

quickGoal 93752 31 19956.png
quickGoal 93752 31 19956.bb

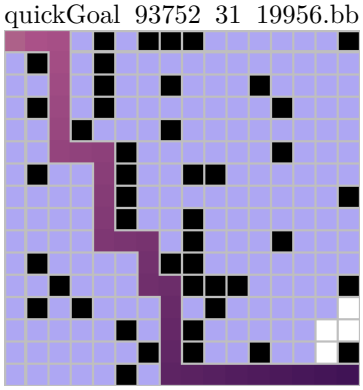

Figure 14

113708 31 27898.png
113708 31 27898.bb

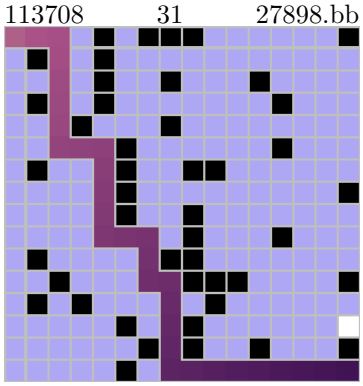

Figure 15

randomWalk 113708 31 27898.png
randomWalk 113708 31 27898.bb

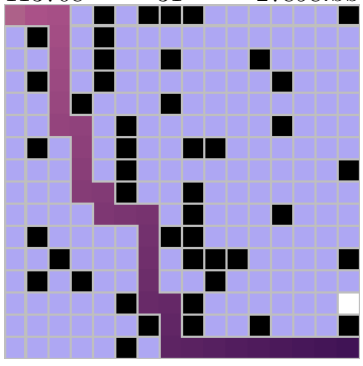

Figure 16

BDBFS quickGoal randomWalk checkFringe 151 31 23.png
BDBFS quickGoal randomWalk checkFringe 151 31 23.bb

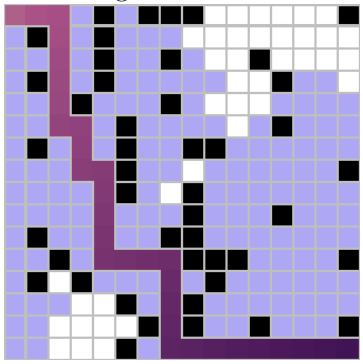

Figure 17

159 31 24.png 159 31 24.bb

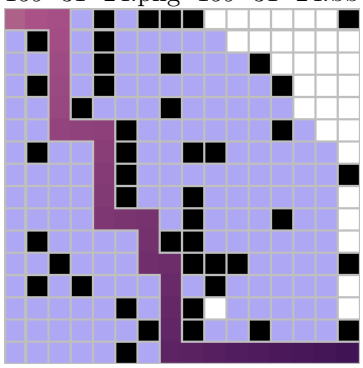

Figure 18

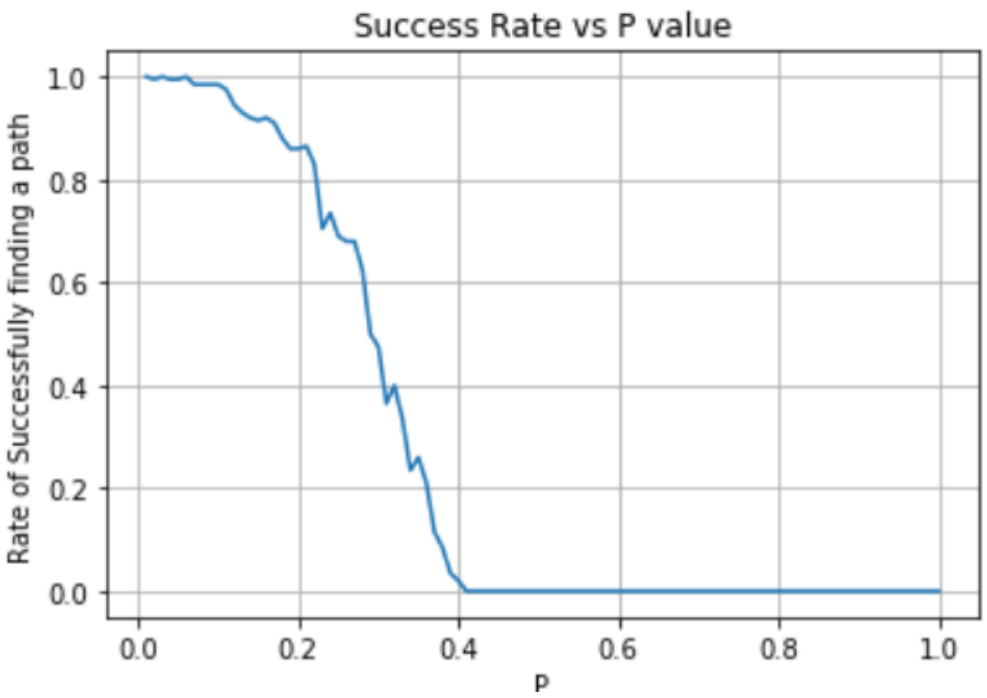

Figure 19: rate of successfully finding a path vs the p-value.

p close to 1, the map is mostly filled and there is no clear path. There is some threshhold value p_0 so that for $p < p_0$, there is usually a clear path, and $p > p_0$ there is no path. Estimate p_0 . Which path finding algorithm is most useful here, and why?

In this question, we tried to use a grid-search method first to narrow down the scope of parameters. We used 200 as the mazeSize, and we generated 800 different mazes for the validation of the parameter value.

After getting the result, we plotted the rate of successfully finding a path vs the p-value. See Figure 19.

Then we tried to decrease the step size, and selected the p-value from 0.16 to 0.40 as the x-label to generate a new plot for better analyzing the threshold value here. See Figure 20.

After analyzing the result, we notice that the when p is at 0.40, the success rate stays at 0. Thus we came to conclusion that the threshold is 0.40 for p here. And to gain a 50% of success rate, the p-value should be set to 0.30.

In this problem, we used the Bidirectional A^* method, and the optimal distance function is Manhattan Distance. After the calculation, we found that BDA^* consumes less time than other algorithms. In this specific problem, because we simplified the settings, we could only move to 4 directions. Manhattan Distance better simulates such scenarios thus it has the best performance here.

4 For a range of p values (up to p_0), generate a number of maps and estimate the average or expected length of the shortest path from start to goal. You may discard all maps where

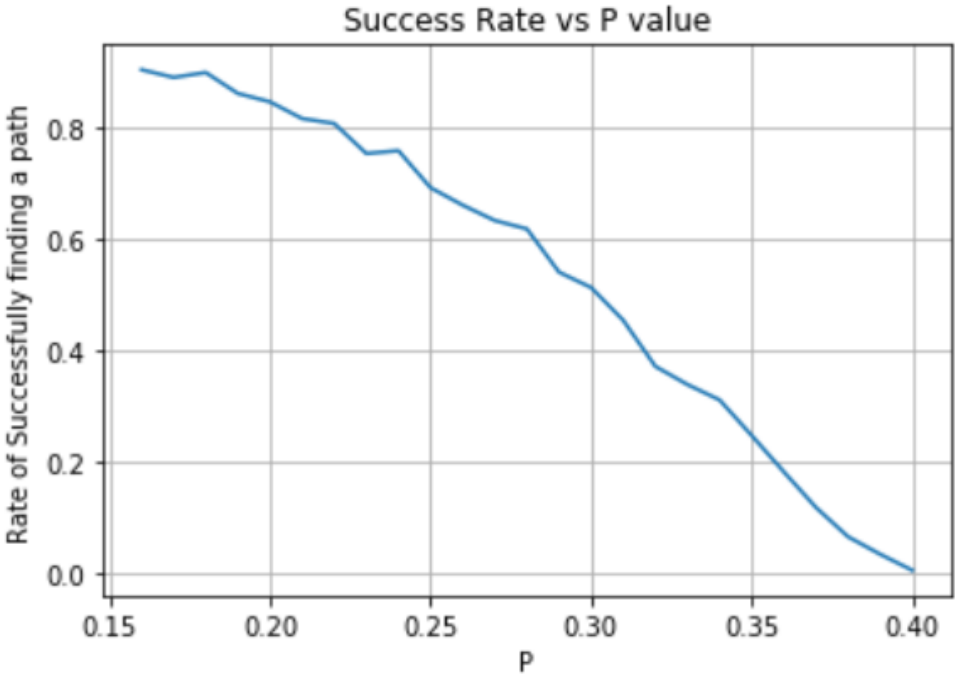

Figure 20: rate of successfully finding a path vs the p-value.

no path exists. Plot your data. What path finding algorithm is most useful here?

We set p from 0.05 to 0.35 and the step size was 0.01. For every p, we generated 100 mazes and let the algorithm solve the maze. At this time, all mazes were solvable. We used A^* with Manhattan Distance Heuristic to solve the problem because A^* algorithm could return the shortest path. For every p, we took the average of 100 shortest path, and we plotted its relation with the probability p. See Figure 21:

From Figure 21, we can see that when p is less than 0.2, the average shortest is the same. Actually, it equals to the length of the shortest path from the upper left to the lower down. When p becomes greater, the average length rises fast. When p is beyond p_0, the average rises fast.

5 For a range of p values (up to p_0), estimate the average length of the path generated by A^* from start to goal (for either heuristic). Similarly, for the same p values, estimate the average length of the path generated by DFS from start to goal. How do they compare? Plot your data.

We generated 100 mazes under different p and used 3 algorithms – A^* with Euclidean Distance Heuristic, A^* with Manhattan Distance Heuristic, and DFS(quickGoal = True, randomWalk = True) to solve the mazes. All mazes are solvable. We calculated the average path of each algorithm and plot them together. The result is shown in Figure 22:

From Figure 22, we can see that A^* with Euclidean Distance Heuristic and A^* with Manhattan Distance Heuristic have the same average shortest path. It is easy to understand this result because A^* returns the shortest path no matter what heuristic function is used. The average length of DFS is larger than A^*. DFS is fast, but it can't return the shortest

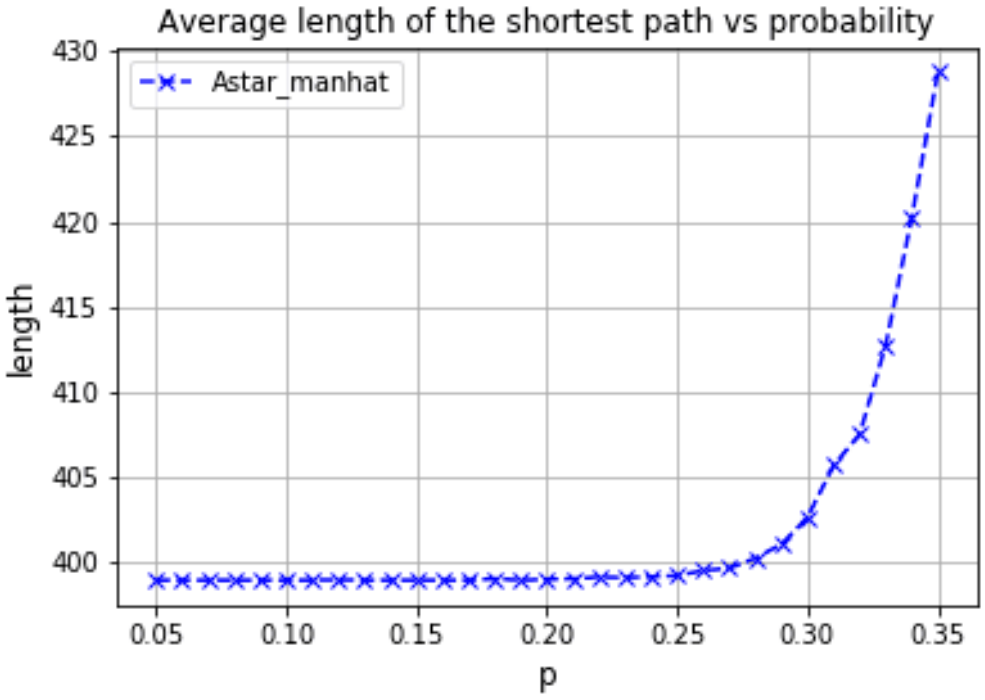

Figure 21: rate of successfully finding a path vs the p-value.

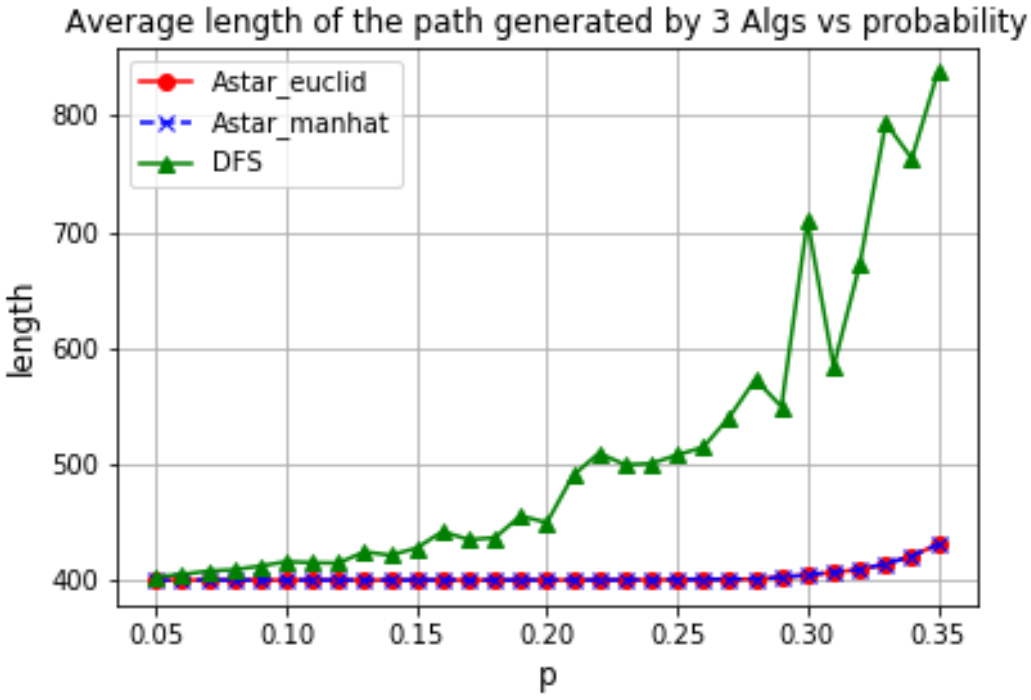

Figure 22: rate of successfully finding a path vs the p-value.

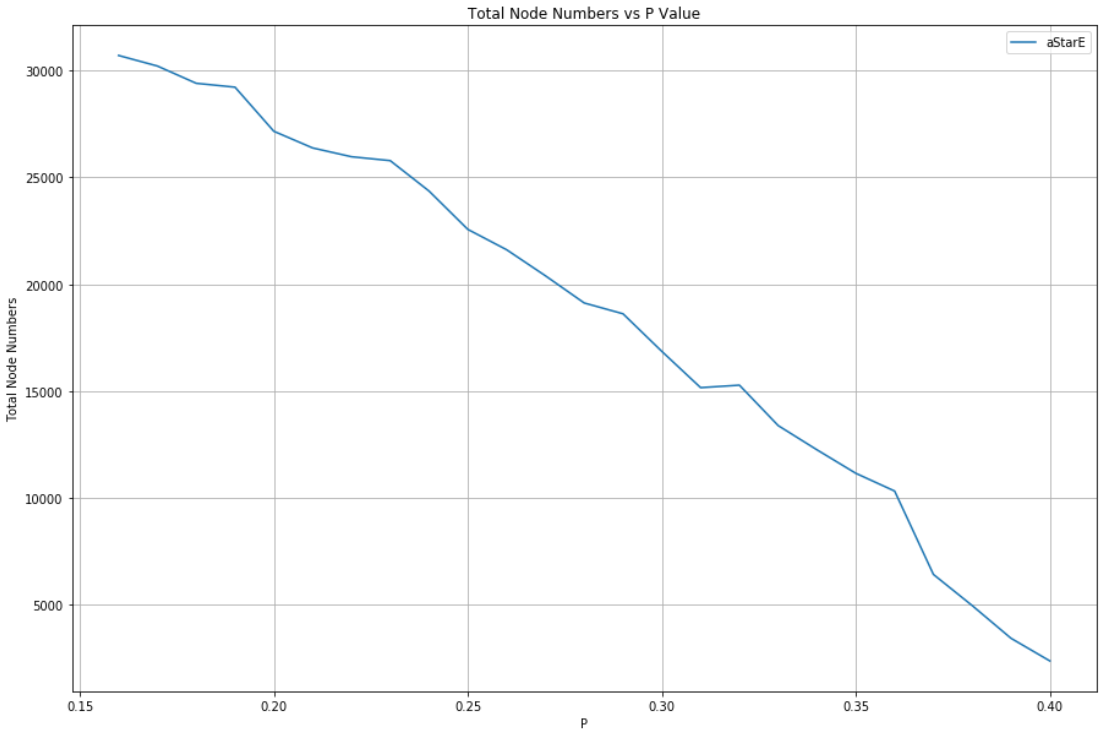

Figure 23: A^* with Euclidean Distance Heuristic.

path when it is not optimized. When p is greater than 0.2, the difference increases quickly. When p is large, there exists oscillation in the average length of DFS. However, we could come to conclusion that the overall trend is rising.

6 For a range of p values (up to p_0), estimate the average number of nodes expanded in total for a random map, for A^* using the Euclidean Distance as the heuristic, and using the Manhattan Distance as the heuristic. Plot your data. Which heuristic typically expands fewer nodes? Why? What about for p values above p_0?

In this question, we generated 200 different mazes to calculate our result and the mazeSize we used is 200×200 here. We used the Euclidean distance as the distance function. The result is demonstrated below as Figure 23.

Note that LIFO is our special bool parameter. When LIFO=True, for nodes whose heuristic values are same, the last node entering the priority queue gets popped first.

When we used the Manhattan Distance, we conducted two tests. Figure 24 was the results of LIFO=True and Figure 25 was the results of LIFO=False.

In this scenario, Manhattan distance when LIFO=True clearly expands fewer nodes. This is because that Manhattan is a better simulation in this scenario. We simplified the situation such that the node could only explore four directions, which suits the way M distance works. As for LIFO, when p is small enough, the whole maze almost has no walls. In this case, almost every node has the same cost thus LIFO is similar to DFS, while FIFO could be regarded as BFS.

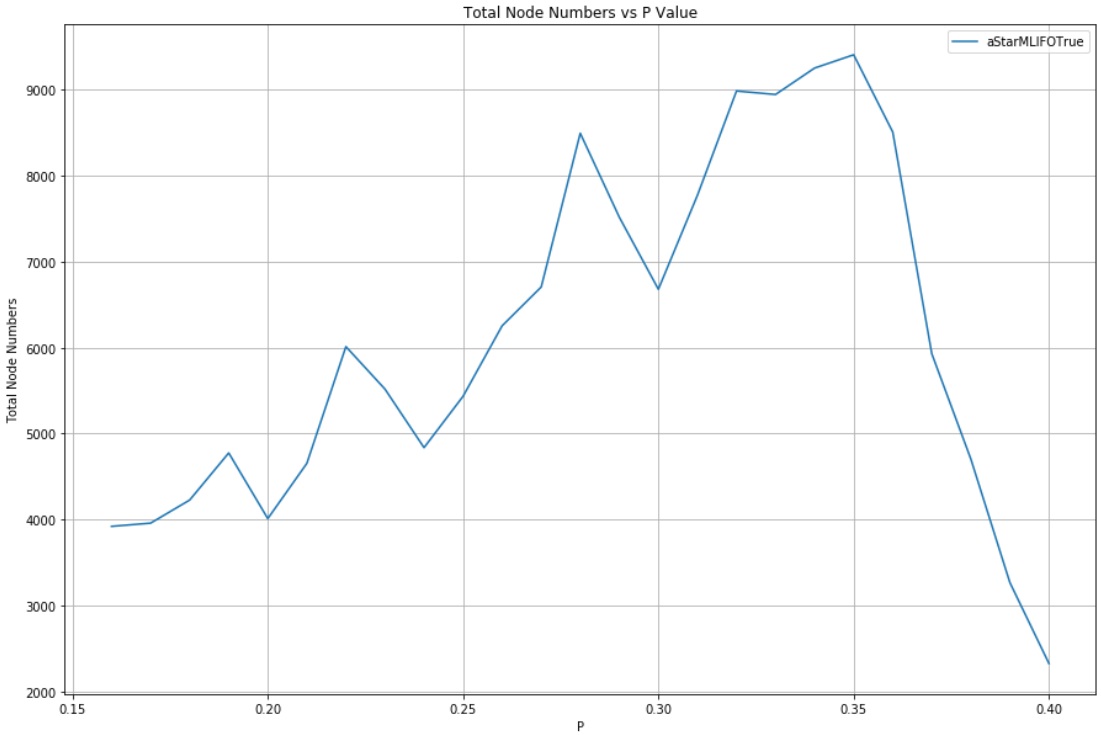

Figure 24: A^* with Manhattan Distance Heuristic when LIFO=True.

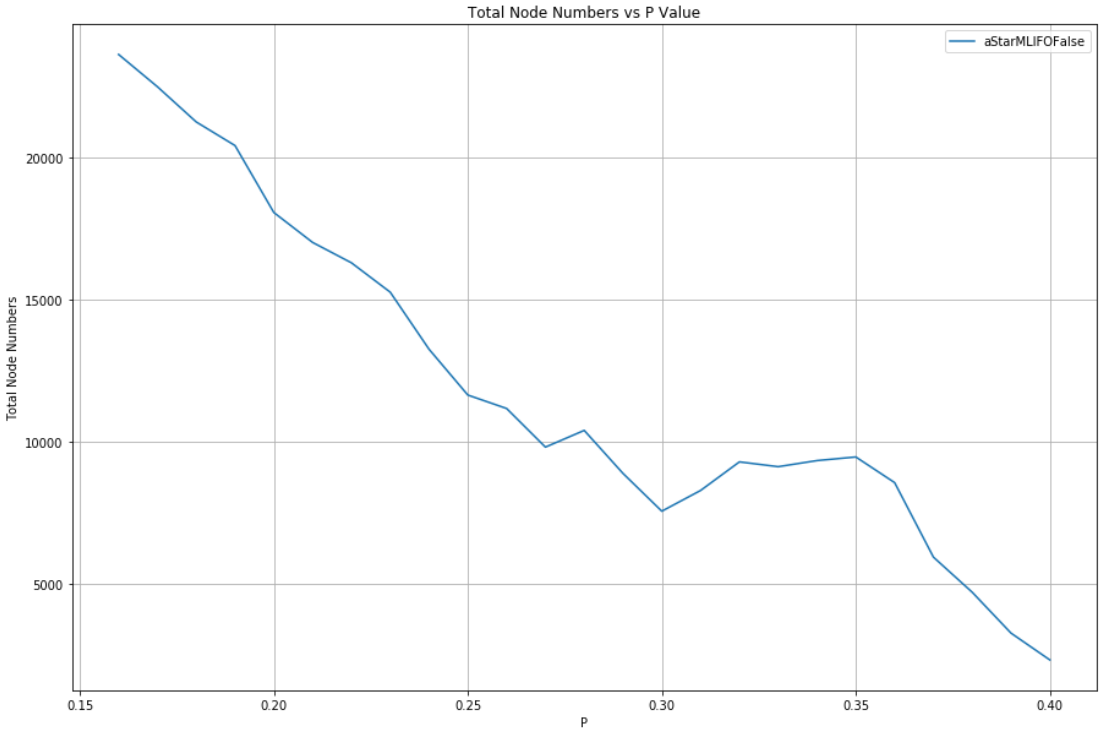

Figure 25: A^* with Manhattan Distance Heuristic when LIFO=False.

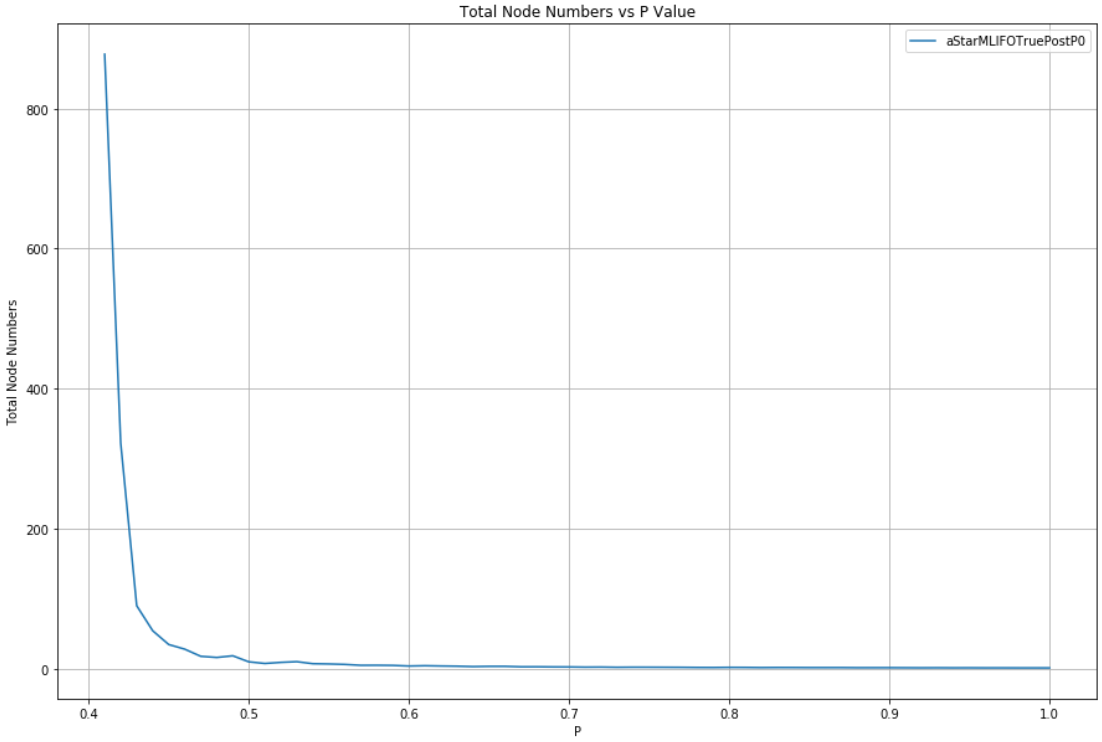

Figure 26: A^* with Manhattan Distance Heuristic when LIFO=True and $p >= p_0$.

For the Total Node Numbers after p_0, we have Figure 26, Figure 27 and Figure 28:

From these figures, we could tell when $p >= p_0$, their shape are similar.

7 For a range of p values (up to p_0), estimate the average number of nodes expanded in total for a random map by DFS and by BFS. Plot your data. Which algorithm typically expands fewer nodes? Why? How does either algorithm compare with A^* in Question (6)?

In this question, we generated 200 different mazes to calculate our result and the mazeSize we used is 200×200 here. The result for DFS is shown as Figure 29:

When we used the BFS, the result is shown as Figure 30:

When p is low, DFS does not need to trackback for a lot of times. As p increases, DFS needs to traceback more. Such that the nodes DFS explores increases as p-value grows. Also, DFS performs better than BFS. When DFS explores a path, it will go down the path until it come to a dead end. At the same time, BFS will explore all the nodes it is adjacent to. In the maze-exploration problem, the correct path often exists in the middle of the nodes, thus typically DFS overperforms BFS.

To draw the comparison, we have Figure 31:

Bonus 1 Why were you not asked to implement UFCS?
UFCS is equal to A^* algorithm with a heuristic function that always returns 0. In this mazerunner case, the cost of all steps are 1, so UFCS is exactly equal to BFS.

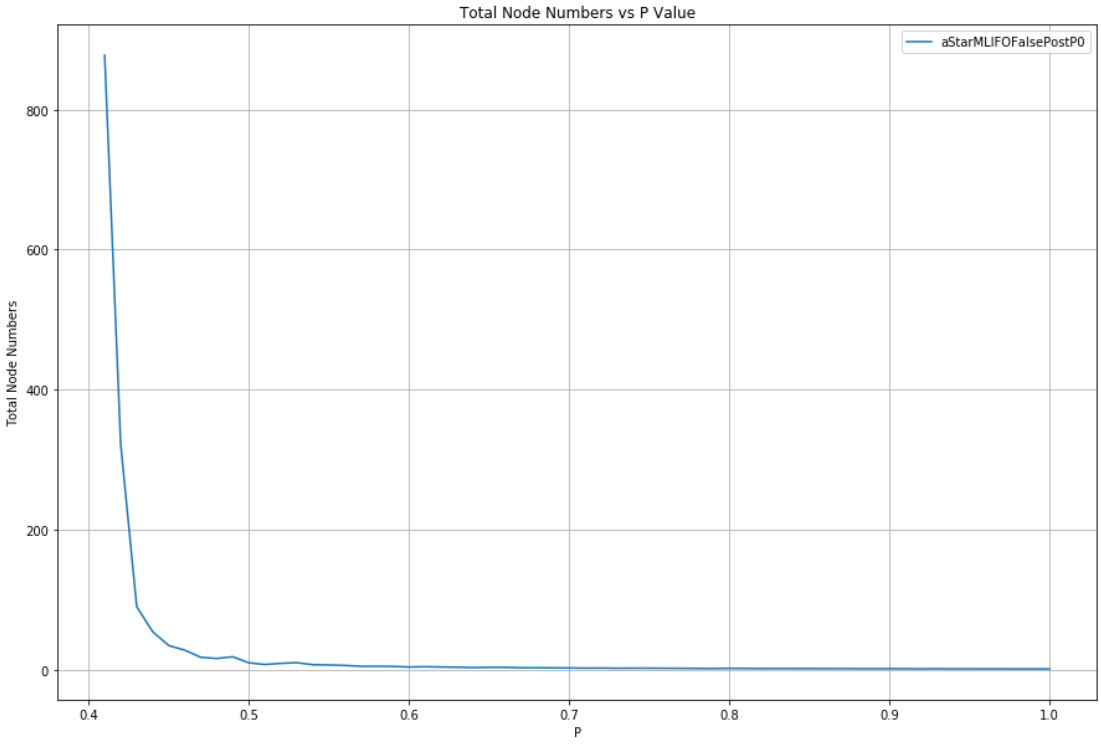

Figure 27: A^* with Manhattan Distance Heuristic when LIFO=False and $p >= p_0$.

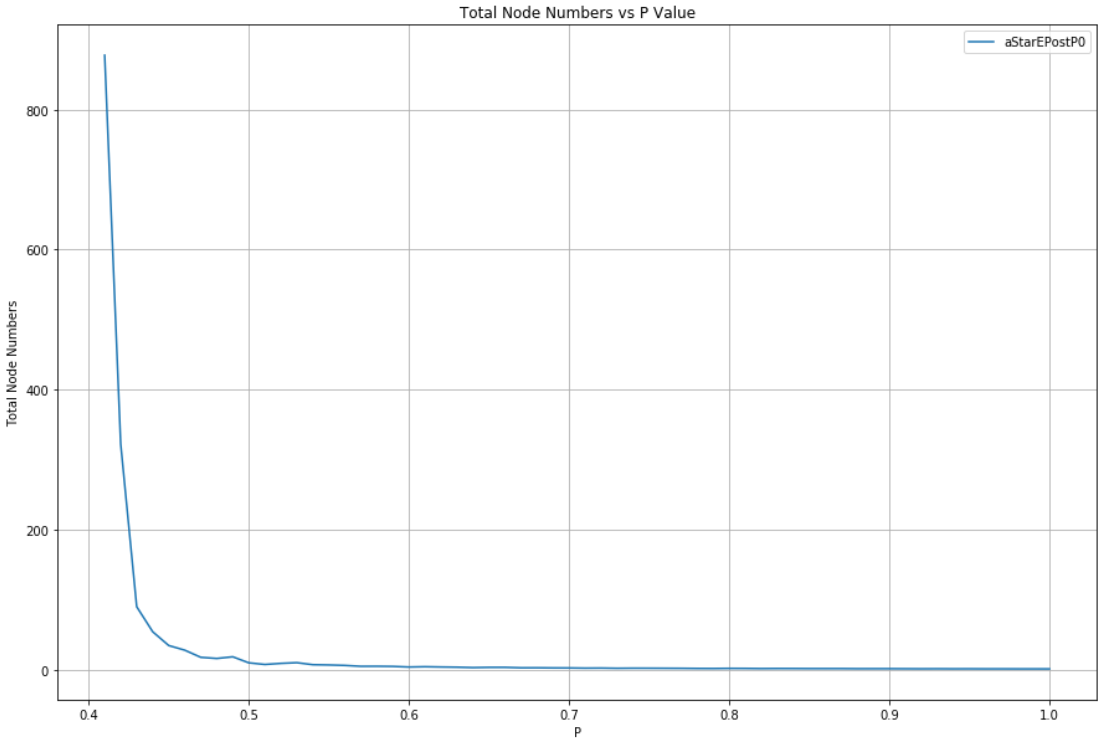

Figure 28: A^* with Euclidean Distance Heuristic when $p >= p_0$.

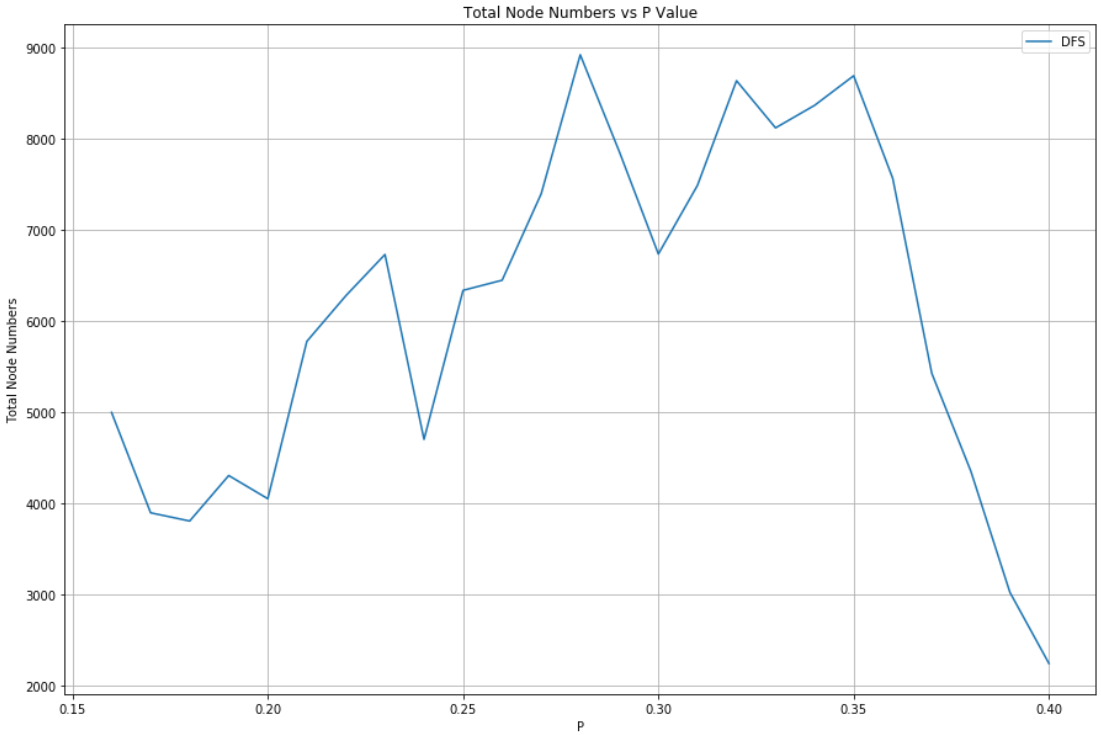

Figure 29: Average number of nodes expanded for different p using DFS.

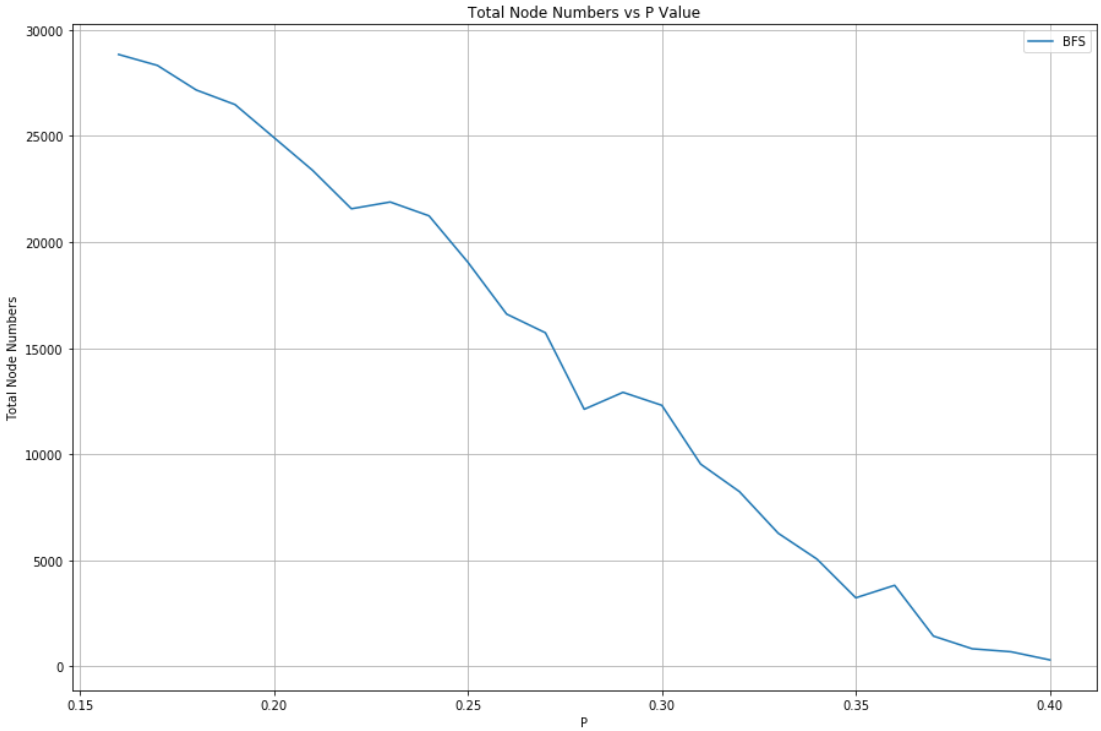

Figure 30: Average number of nodes expanded for different p using BFS.

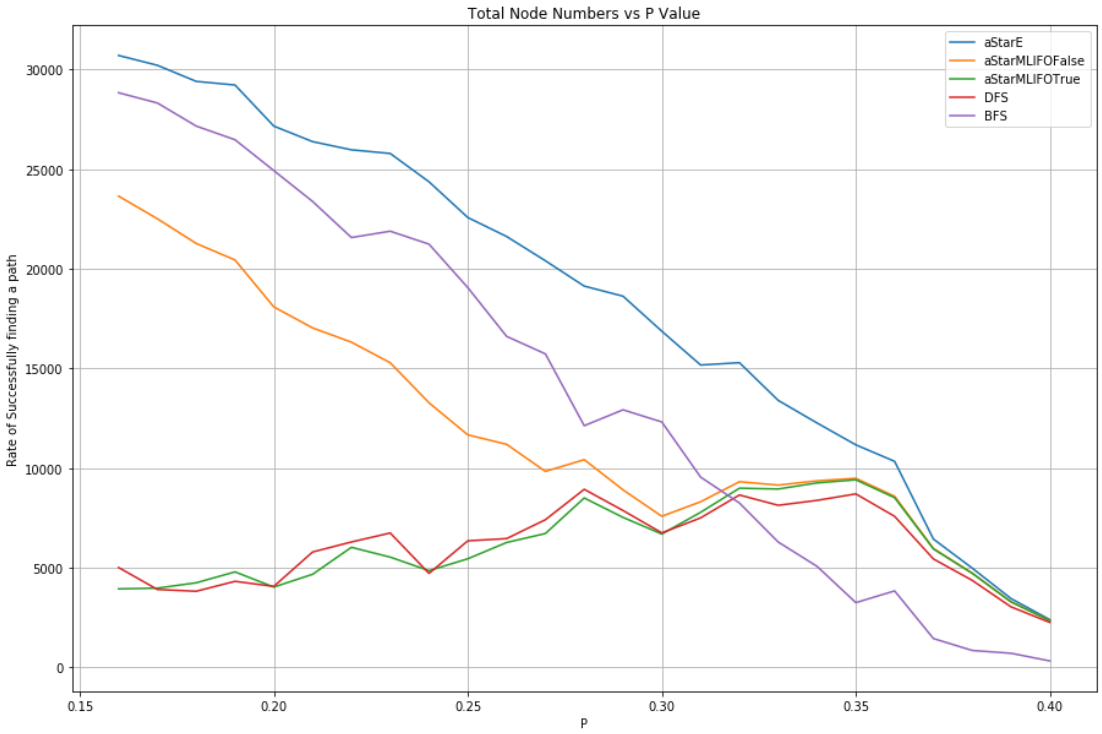

Figure 31: Comparison of average number of nodes expanded.

3 Part 2: Building Hard Mazes

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

   (a) What local search algorithm did you pick, and why?
   We combined Genetic Algorithm with Simulated-Annealing-Based Beam Search. We have several reasons:

      i. Genetic Algorithm works much better than Simulated-Annealing-Based Beam Search when the mutation rate is large.

      ii. Genetic Algorithm usually returns mazes with the "pattern"(or say "schema") that makes a maze hard. However, it is hard to finetune the "pattern" when its size is too large because the pattern is likely to be cut into 2 halves and unlikely to be pieced together again. For instance, (one of) the ideal pattern of the longest path maze should seem like a Hilbert curve, which is as large as the maze. A slight dislocation will make the maze unsolvable. Hence we need another algorithm to finetune.

      iii. Beam Search requires a brunch of initial states(mazes), which perfectly matches the population of final iteration from Genetic Algorithm. Also, Simulated Annealing can be helpful when it runs into a local maximum. Combine Beam Search and Simulated Annealing to both utilize most mazes that Genetic Algorithm returned and avoid converging at a local maximum.

   (b) How are you representing the maze/environment, to be able to utilize your chosen search algorithm? What design choices did you have to make to apply this search

algorithm to this problem?

Overall:

i. A maze can be represented by a $rows \times cols$ boolean matrix, where True means this block is a wall, and False means the agent can go through this block.

ii. The objective function value(fitness) of a maze is calculated by this formula:

$$ObjectiveFunction(maze) = \begin{bmatrix} BlockCount & PathLength & FringeSize \end{bmatrix} \times \begin{bmatrix} w_{block} \\ w_{path} \\ w_{fringe} \end{bmatrix}$$

If we want to find the hardest maze in terms of a single aspect, set one weight to 1, and set others to 0.

iii. Genetic Algorithm will return much more mazes than what Beam Annealing requires. Therefore, using the idea similar to Simulated Annealing to pick some mazes as seeds of Beam Annealing.

A. Pick a few best mazes as "perfect seeds".

B. Discard mazes whose fitness is lower than some percent, say 90%, of maximum fitness, if the number of remained mazes is larger than the size of Beam Search.

C. Randomly pick some "non-prefect seeds" with the probability:

$$P = e^{\frac{Weight}{PerfectRatio} + Bias}, PrefectRatio = \frac{ObjectiveFunction(otherMaze)}{ObjectiveFunction(bestMaze)}$$

D. Keep picking until the number of "seeds" equals the size of Beam Search.

iv. After Beam Annealing, compare the fine-tuned result with previous Genetic Algorithm's result. Return best several mazes of both results. (In this case, Beam Annealing failed because Genetic Algorithm presented a nearly perfect result. However, Annealing allows mazes to get a little easier, which is difficult to become harder again since it is hard to randomly get a perfect maze.

Genetic Algorithm:

TODO

Beam Annealing:

i. A maze's neighbors are all the mazes that can become this maze by changing several blocks. Therefore, we choose a very small probability to flip each block's value to generate some neighbors of this maze for Stochastic Beam Search.

ii. Each solution algorithm returns 3 values: the number of blocks it has explored, the length of the path found, and the maximum size of the fringe it has used.

iii. The probability to move down is calculated by this formula:

$$P = e^{\frac{Weight * \Delta E + Bias}{Temperature}}, \Delta E = ObjectiveFunction(newMaze) - ObjectiveFunction(maze)$$

A larger $Weight$ can decrease P without any other influence. A larger $Bias$ mainly decrease the probability to keep step forward and back in a "plateaux",

which works together with "impatient halt"(explained in Question 9).

iv. Decrease the temperature by multiply a const smaller than 1.0, called "cool rate", which makes the temperature go down fast at first and then getting slower and slower to let agent have more chance to climb up(explained in Question 9).

v. Speaking of Beam Annealing, the key point is to make an agent can "regress" to where it was, especially when an agent "teleports" to another agent's region. Hence, once an agent teleport, one of its neighbors is set to a maze in the previous region.

vi. In order to increase the performance, each neighbor maze will be check if it is solvable by using BDA*. If it is unsolvable, regenerate it.

2. Unlike the problem of solving a maze, for which the 'goal' is well-defined, it is difficult to know when we have constructed the 'hardest' maze. That being so, what kind of termination conditions can you apply to your search algorithm to generate 'hard' if not the 'hardest' mazes? What kind of shortcomings or advantages do you anticipate your approach having?

Genetic Algorithm:
TODO

Beam Annealing:
There are 3 different ways to halt the iteration:

(a) Maximum Iteration Limit: Count the time it iterates. Once it exceeds the limit, halt and return the result.
- It is the easiest way to terminate iteration, but actually, there is no reason to terminate it EXACTLY there, except that it has run too many times.

(b) System Cooled Down: As time goes(it iterates), the temperature of Annealing decreases. If it is below a minimum temperature, halt and return the result.
- If the temperature is too "cold", there is no need to keep Annealing. No easier mazes could be picked up. It is reasonable.
- It related to the cool rate. If the cool rate is too small, the temperature will decrease too fast, and there is not enough time for a maze getting harder after becoming easy. However, if the cool rate is too large, the temperature will keep a high level for a long time, which may totally destroy the work Genetic Algorithm has done.

(c) No Patience Left: If agents do not move for a really long time, it must converge at a local optimal. It is time to halt and return the result.
- Since we have found a local optimal, we should return it.
- In the beginning, it is less likely to converge. Hence we should iterate it for a little more times. But if it has been iterated for lots of times, it is unnecessary to keep iterations going.
- When an agent climbs on a "plateaux", it will get lost. Hence, we should limit its ability to move to an equal-difficult maze by using $Bias$ to calculate the probability to move.
- The issue is, sometimes, Genetic Algorithm returns a "nearly perfect" maze. In this case, it takes at least 30 iterations to halt, since we assume what Genetic Algorithm did is just pre-training.

3. For each of the following algorithms, do the following: Using your local search algorithm, for each of the following properties indicated, generate and present three mazes that attempt to maximize the indicated property. Do you see any patterns or trends? How can you account for them? What can you hypothesize about the 'hardest' maze, and how close do you think you got to it?

   (a) DFS

   The configuration of DFS is set to 'quickGoal' : True, 'distinctFringe' : True, and all others are False.

   i. Length of solution path returned: See Figure 32, Figure 33 and Figure 34.

   Patterns or trends:

   - There are 2 classical patterns of walls. (See Figure 35) In one case, walls lead agents to go down some rows, then there usually several nearly "empty"(no walls) rows to move horizontally. But at the end of these rows, diagonal walls lead agents to move up. This case looks like "I" and ">". The other case is "J-shape" walls.
   - "I" and ">" walls and "J-shape" walls appear alternately.

   Reasons:

   - The key point is when randomWalk == False, the priority of 4 directions is Right > Down > Left > Up.
   - "I" and ">" walls lead agent walk horizontally, and the path it has walked becomes "new wall" because those blocks have been added into the closed set. Now it becomes a "J-shape" walls in the mirror.
   - "J- shape" walls force agents search all blocks inside because agents prefer to go right or left than to move up.

   "The hardest" maze:

   - The ideal "J-shape" walls can be maze-sized. (See Fig 36) In this case, the path length can be $size \times size - WallCost = 16312$. Notice that if size is an odd number, not all the accessible blocks are in the path. Specifically, only the first 3 blocks in the 2nd row are in the path.
   - Compared with 16312, 5557 is not a big number. However, it is extremely impossible to randomly set all walls in a line.
   - However, compared with less than 1000 block length in a 200 by 200 maze(from Question 5), 5557 in a 128 by 128 maze can be regarded as a "harder" maze, though it is not the "hardest" maze.

   ii. Total number of nodes expanded: See Figure 37, Figure 38 and Figure 39.

   Patterns or trends:

   - The pathway near the Goal(10 * 10 blocks in the lower right corner) is extremely narrow(only 1 block width), opens to left, and begins with a "door sill" wall that forces the path to move up a block.
   - "I" and ">" walls and "J-shape" walls appear alternately.

   Reasons:

   - Notice that when randomWalk == False, the priority of 4 directions is Right > Down > Left > Up.
   - The pathway opens to left rather than up makes DFS cannot first run right to the last column and then find the pathway. It takes time for DFS to search all upper right part(above the path) and then realize "Oops, there is no way to the goal. I have to backtrack."
   - After DFS reached the "door sill", it prefers to go to any directions but up. Hence, it takes more time for DFS to search all lower left part(below the path) and then realize "Alright, I will go up to see if there is a path."

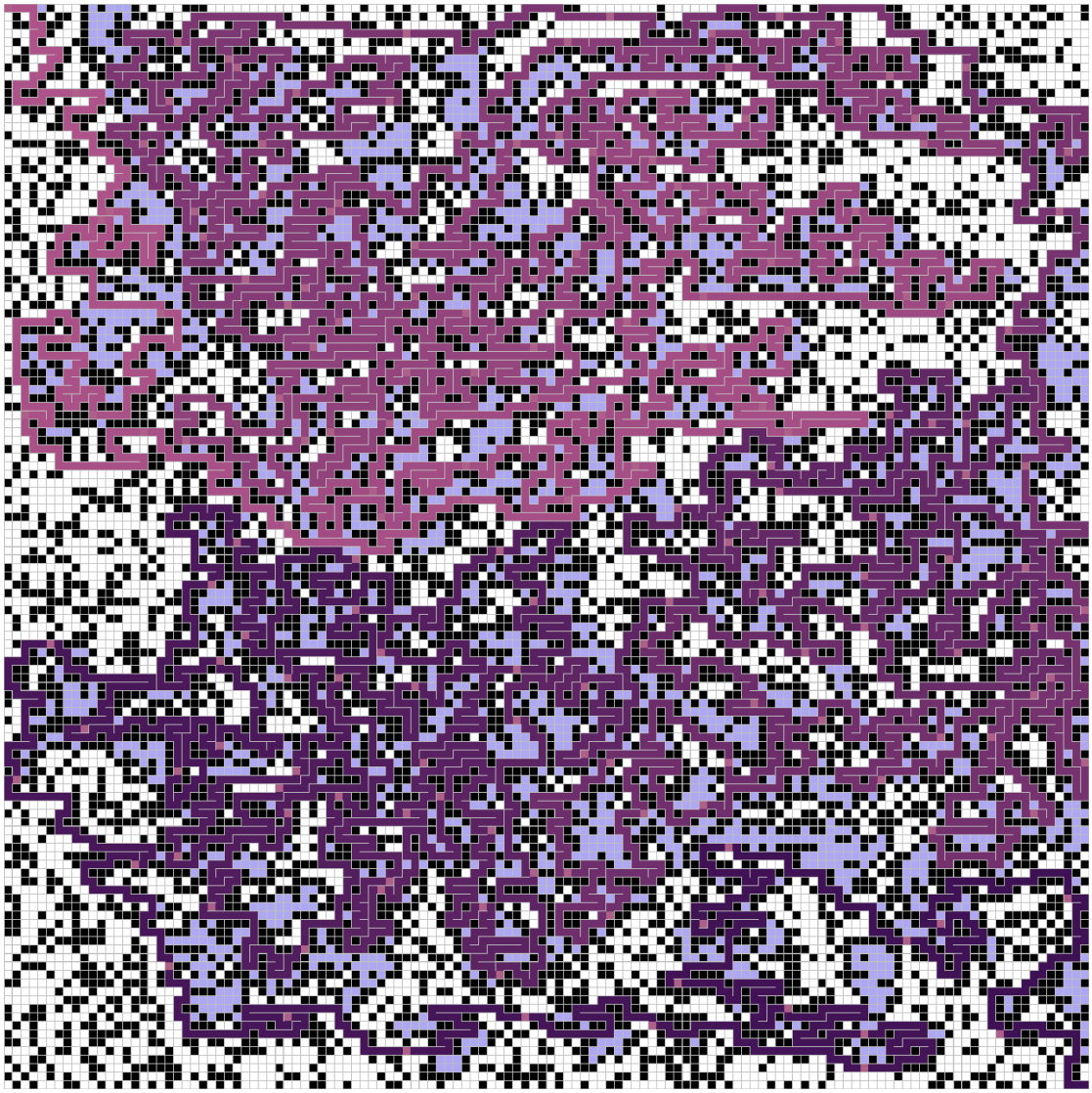

Figure 32: maze that is hard for DFS in terms of path length, which is 5557.

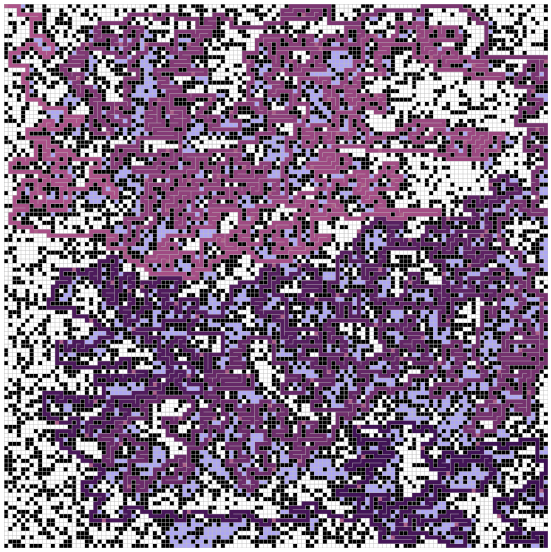

Figure 33: maze that is hard for DFS in terms of path length, which is 5503.

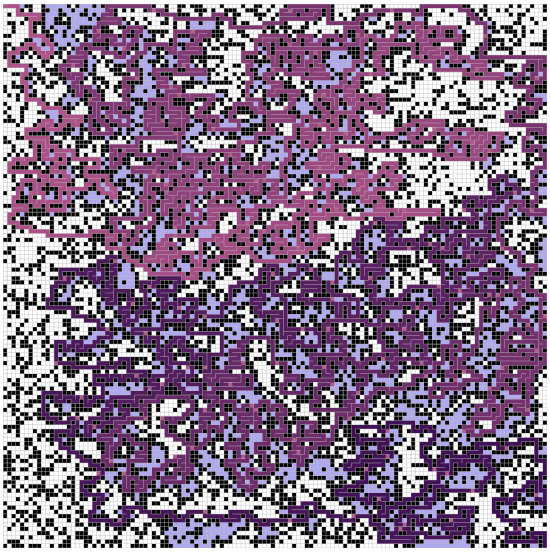

Figure 34: maze that is hard for DFS in terms of path length, which is 5495.

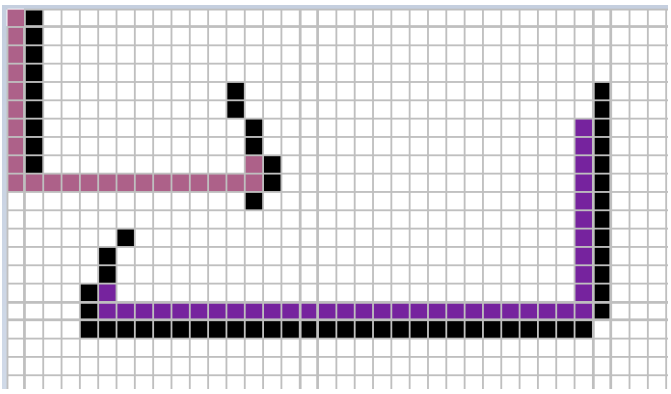

Figure 35: Two classical patterns of walls: "I" and ">" walls and "J-shape" walls.

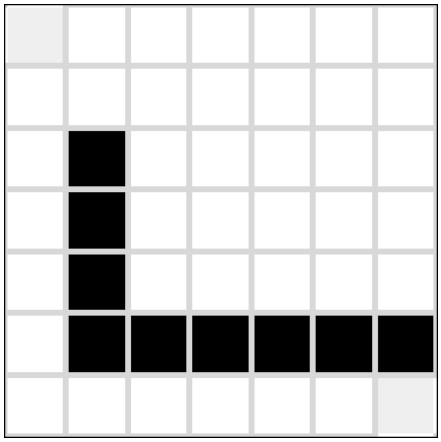

Figure 36

- Therefore, DFS nearly searched all the blocks to reach the Goal.
- Notice that if randomWalk == True, this pattern will not always work because DFS may go down first rather than right.

"The hardest" maze:

- A maze is "empty"(no walls) except the lower right corner. The lower right corner should be a pathway like this pattern. The smallest pattern can be done in 4 * 4.(See Fig 40). Notice there must be at least 2 block height "door sill" for DFS to turn back to search lower left part.
- Compare to $size \times size - WallCost = 16377$, 12993 can still be improved. However, it is really difficult to cancel out those outside walls by random without destroying the pattern. Therefore, it has been a nearly "hardest" maze.

iii. Maximum size of fringe during runtime: See Figure 41, Figure 42 and Figure 43.

Patterns or trends:

- The same pattern as Question 10.b.ii, but there should be a clear path along the first row and upper half of the last column.

Reasons:

- The time complexity of DFS related to the maximum depth it has searched.
- Hence, a "door sill" pattern can force DFS to search lower left part rather than immediately reach the Goal, which increased the maximum depth of DFS.
- A larger lower left part increases the depth, so the upper right part should be as small as possible.

"The hardest" maze:

- The same maze as Question 10.b.ii.
- The maximum fringe size is theoretically $(size - 1)^2 - WallCost = 16113$, compared with 10032. Noticed distinctFringe actually do not promise all elements in the fringe are distinct because it takes too much time to search in a stack frequently. (Further information is available in description.md)
- Again, it is really difficult to cancel out those outside walls by random without destroying the pattern. Therefore, it has been a nearly "hardest" maze.

(b) BFS

The configuration of BFS is set to 'quickGoal' : True, 'checkFringe' : True, and all others are False.

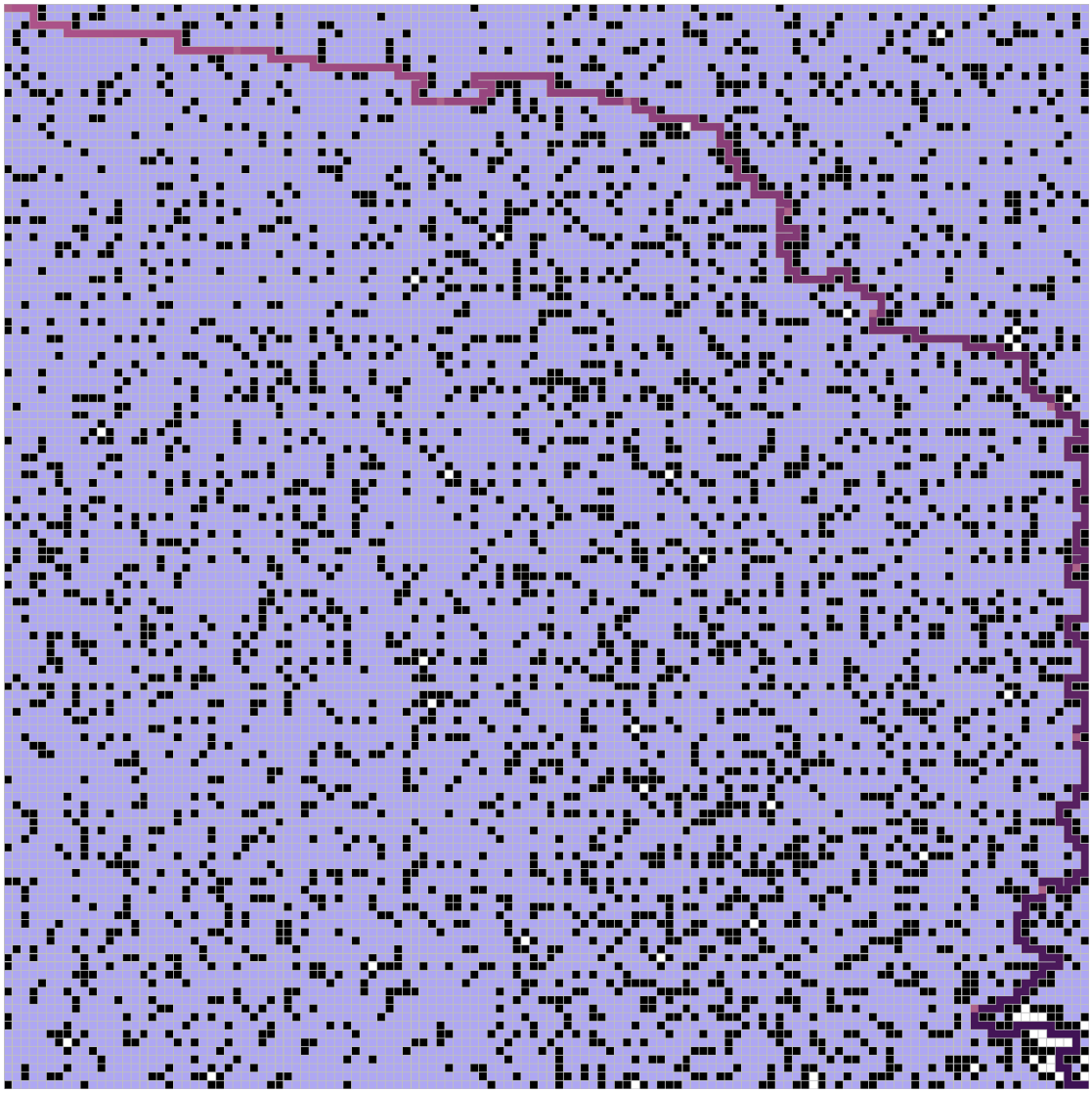

Figure 37: A maze that is hard for DFS in terms of time complexity, which has opend 12993 blocks.

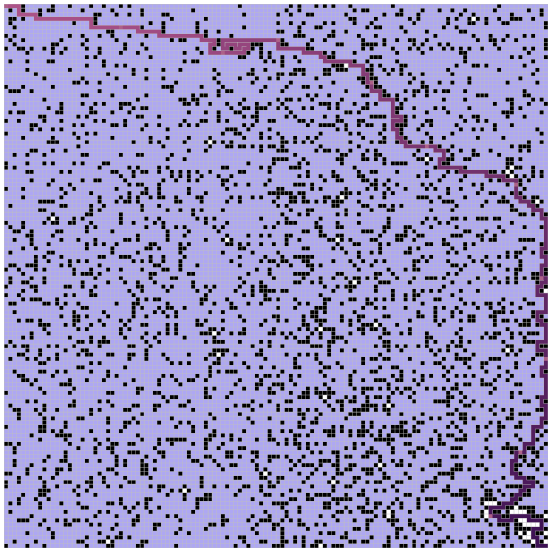

Figure 38: A maze that is hard for DFS in terms of time complexity, which has opend 12968 blocks.

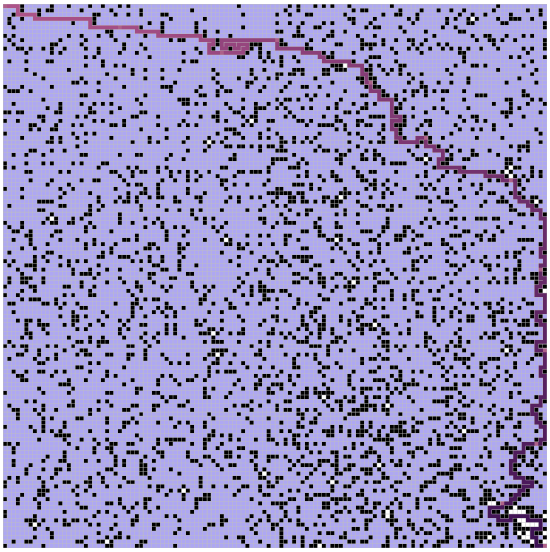

Figure 39: A maze that is hard for DFS in terms of time complexity, which has opend 12967 blocks.

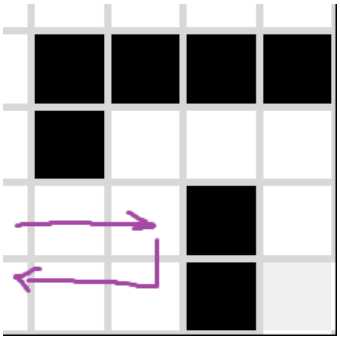

Figure 40: The smallest pathway pattern.

i. Length of solution path returned: See Figure 44, Figure 45 and Figure 46.
Patterns or trends:
- The path looks like a Hilbert curve, and as time goes, the path gets more and more sinuous.
- Walls are closely next to the path.

Reasons:
- It is true that the path can also look like a "Gluttonous Snake", which goes down to the bottom, walk right for 2 blocks, goes up to the top, walk right for 2 blocks, and repeat. However, it is hard to generate walls lined up just by random, so a Hilbert-curve-like maze is more probable.
- Hilbert curve can theoretically reach every point of a plane, i.e., its length can be the maximum.

"The hardest" maze:
- There are 2 kinds of the "hardest" maze: "Gluttonous-Snake-like" mazes and Hilbert-curve-like mazes. Both of them can let the path length reached $\frac{size*size}{2} = 8192$, where the 2 is the cost of building walls to limit the path.
- Compared with 8192, 2117 is not "hard" enough. It is also presented in Fig 2.10.5 that there are still many blocks totally unused. However, due to the time limitation, we cannot run the iterations more times. Yet, 2117 compared with 500 in Question 4(Notice that the size of mazes in Question 4 is 200, which is larger than 128.) can be regarded as a "harder" maze.
- Also, it is interesting to take the maximum fringe size into account. The ideal "hardest" maze's fringe size can be always 1 because there is only one path whose width is also 1 to reach the Goal. Hence, the maximum fringe size can also measure how close we get to the "hardest" maze. But notice that not all mazes whose max fringe size is 1 are "hardest" maze, due to the fact that the path can directly go diagonally. Table 3(b)i shows the fringe size and path length of top 5 "harder" mazes we have generated.

Block	Path	Fringe
7013	2117	14
7467	2093	18
7293	2091	18
7382	2069	27
7433	2069	26

Table 2

ii. Total number of nodes expanded: See Figure 47, Figure 48 and Figure 49.

Patterns or trends:

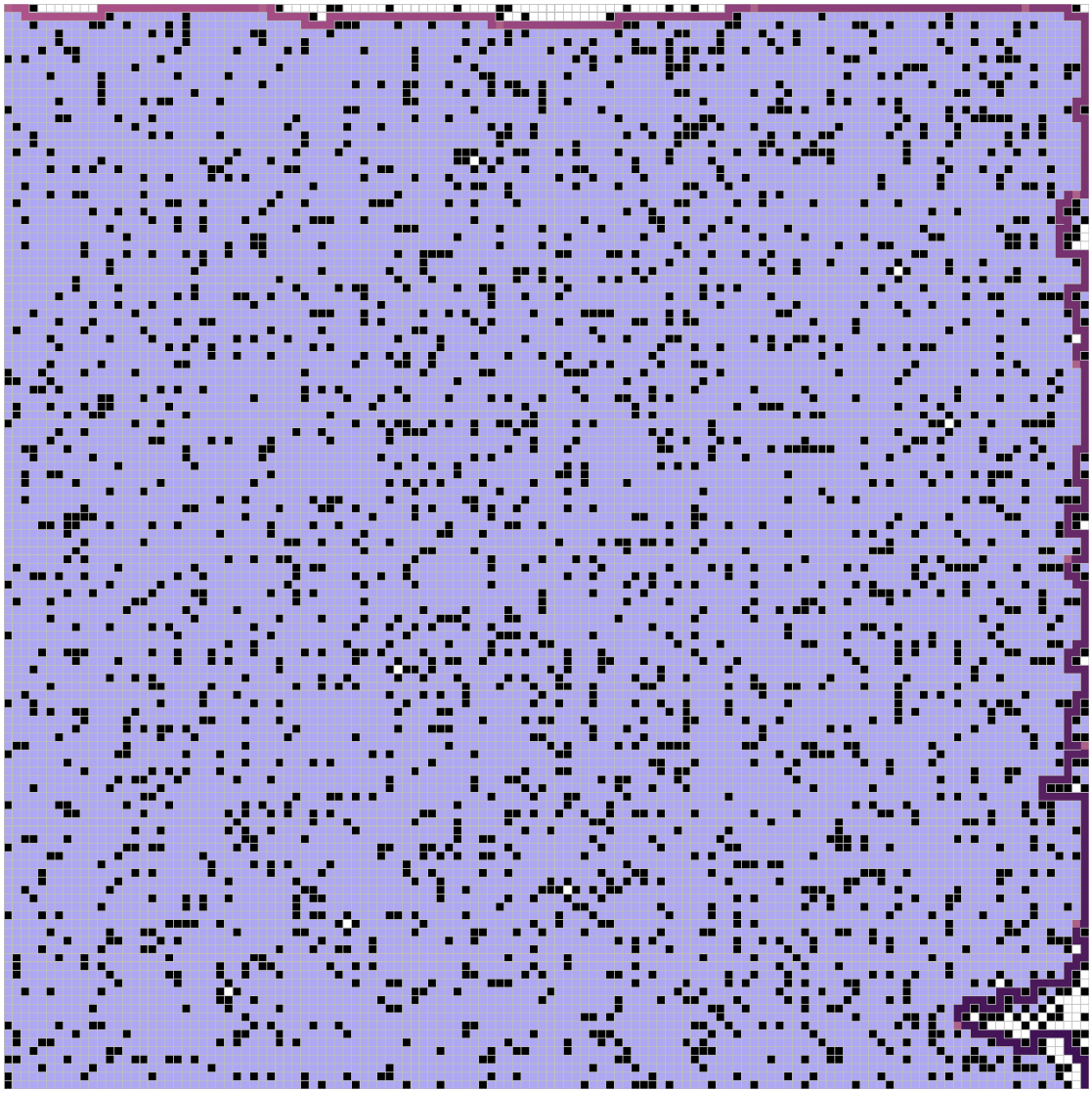

Figure 41: A maze that is hard for DFS in terms of space complexity, whose maximum fringe size is 10032.

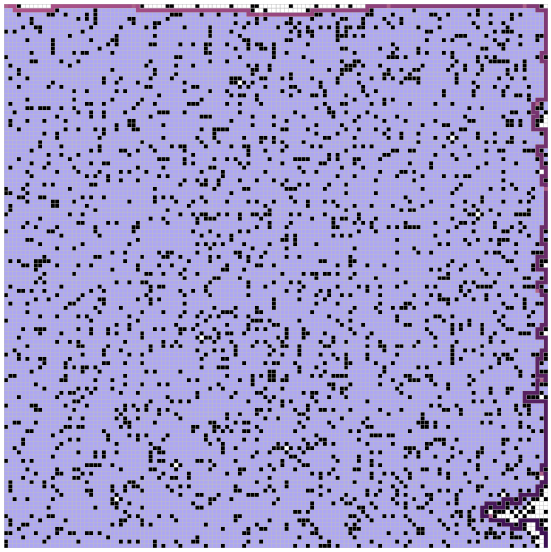

Figure 42: A maze that is hard for DFS in terms of space complexity, whose maximum fringe size is 10032.

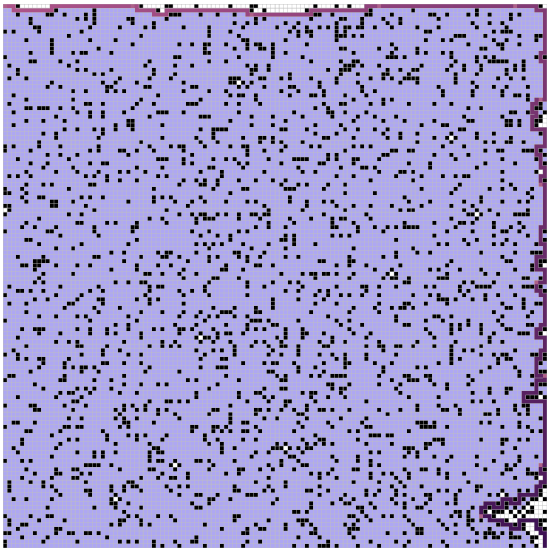

Figure 43: A maze that is hard for DFS in terms of space complexity, whose maximum fringe size is 10031.

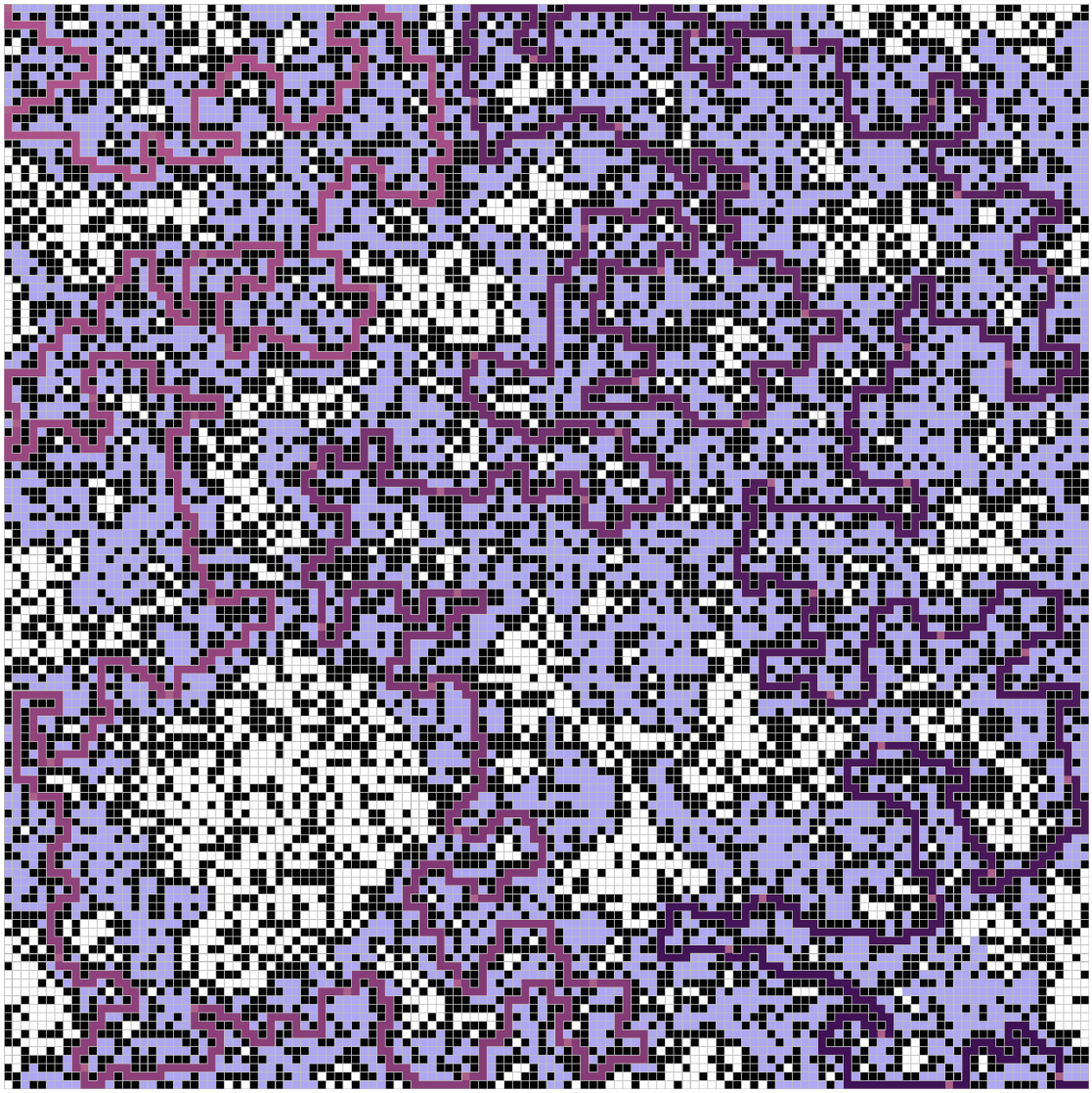

Figure 44: maze that is hard for BFS in terms of path length, which is 2117.

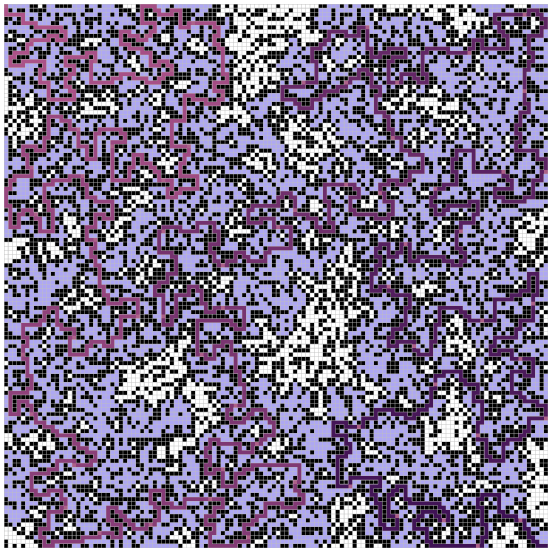

Figure 45: maze that is hard for BFS in terms of path length, which is 2093.

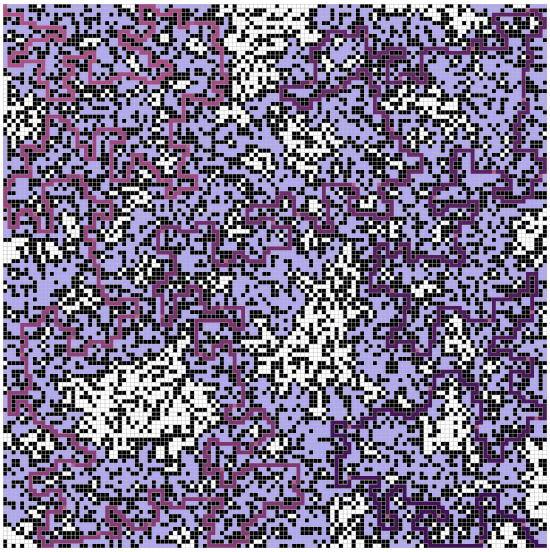

Figure 46: maze that is hard for BFS in terms of path length, which is 2091.

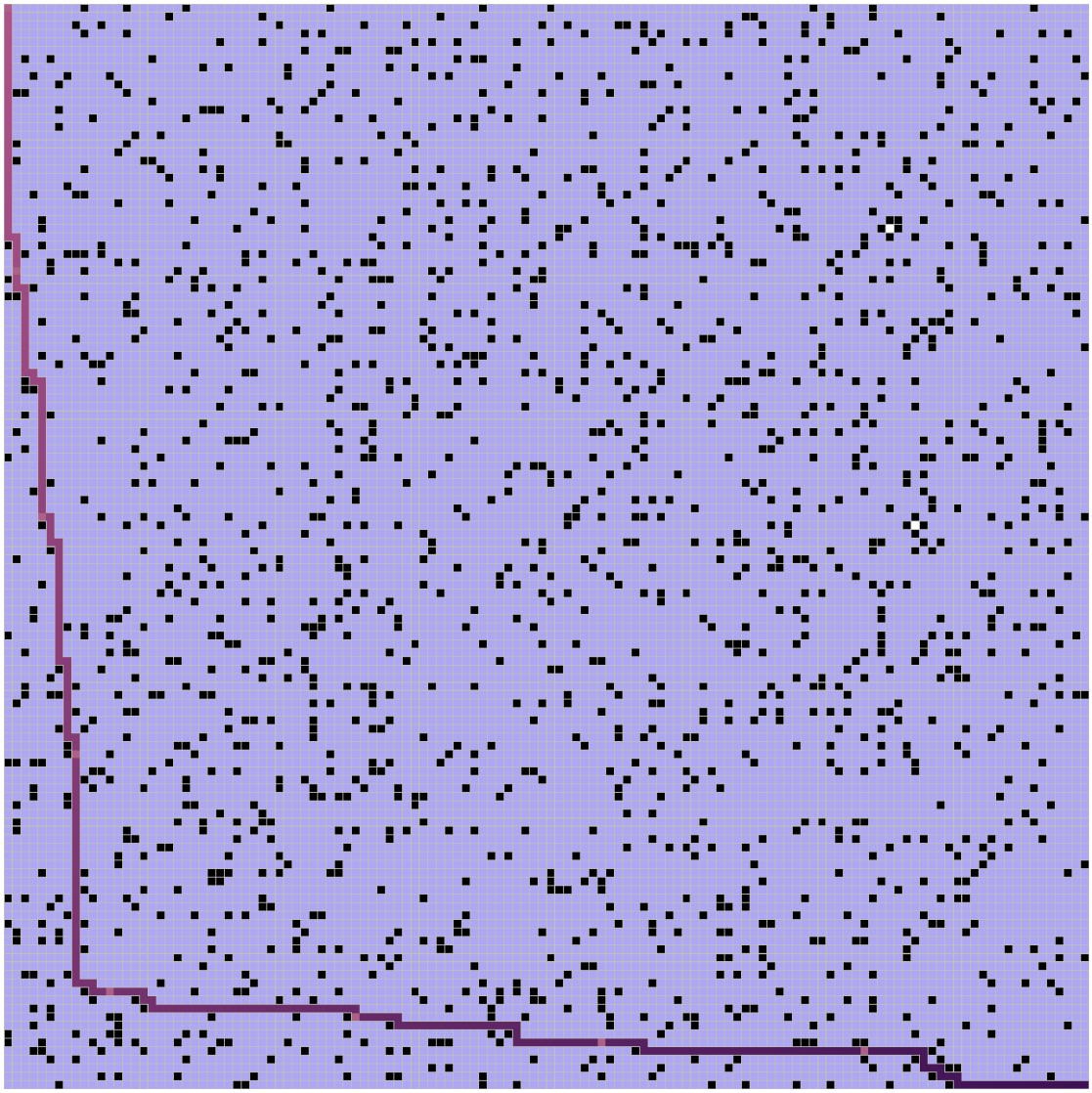

Figure 47: A maze that is hard for BFS in terms of time complexity, which has opend 14776 blocks.

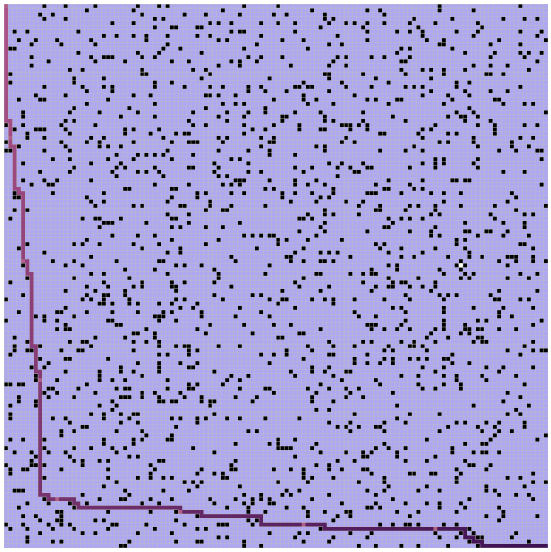

Figure 48: A maze that is hard for BFS in terms of time complexity, which has opend 14772 blocks.

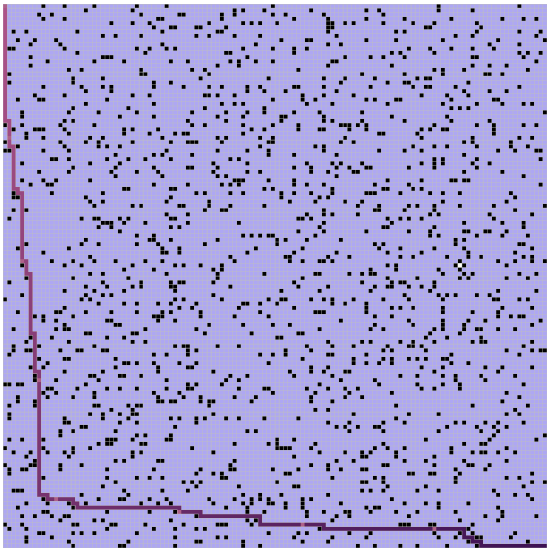

Figure 49: A maze that is hard for BFS in terms of time complexity, which has opend 14772 blocks.

- A nearly "empty"(no walls) maze is usually hard for BFS in terms of time complexity.

Reasons:

- BFS will open shallower blocks first, and then move to deeper blocks.
- Usually, the Goal is the deepest block if the Start is located at 2 ends of a diagonal.
- In this case, BFS will open all the accessible blocks before reaching the Goal. The number of blocks opened is exactly the number of all accessible blocks of this maze.
- To make it harder is to make more blocks accessible. Namely, to delete more walls.

"The hardest" maze:

- A totally "empty" maze whose number of blocks opened is exactly $size \times size = 16384$.
- The best score we have got is 14776, compared with at most 16384. However, it is really difficult to cancel out those walls by random. Therefore, it has been a nearly "hardest" maze.

iii. Maximum size of fringe during runtime: See Figure 50, Figure 51 and Figure 52.

Patterns or trends:

- The same pattern as Question 10.b.ii, but there should be a clear path along the first row and upper half of the last column.

Reasons:

- The time complexity of BFS related to the maximum depth it has searched.
- Hence, a "door sill" pattern can force BFS to search lower left part rather than immediately reach the Goal, which increased the maximum depth of BFS.
- A larger lower left part increases the depth, so the upper right part should be as small as possible.

"The hardest" maze:

- The same maze as Question 10.b.ii.
- The maximum fringe size is theoretically $(size - 1)^2 - WallCost = 16113$, compared with 10032. Noticed distinctFringe actually do not promise all elements in the fringe are distinct because it takes too much time to search in a stack frequently. (Further information is available in description.md)
- Again, it is really difficult to cancel out those outside walls by random without destroying the pattern. Therefore, it has been a nearly "hardest" maze.

(c) A^* with Euclidean Distance Heuristic

i. Length of solution path returned: See Figure 53, Figure 54 and Figure 55.

Patterns or trends:

- There are 2 classical patterns of walls. (See Figure 35) In one case, walls lead agents to go down some rows, then there usually several nearly "empty"(no walls) rows to move horizontally. But at the end of these rows, diagonal walls lead agents to move up. This case looks like "I" and ">". The other case is "J-shape" walls.
- "I" and ">" walls and "J-shape" walls appear alternately.

Reasons:

- The key point is when randomWalk == False, the priority of 4 directions is Right > Down > Left > Up.
- "I" and ">" walls lead agent walk horizontally, and the path it has walked becomes "new wall" because those blocks have been added into the closed set. Now it becomes a "J-shape" walls in the mirror.

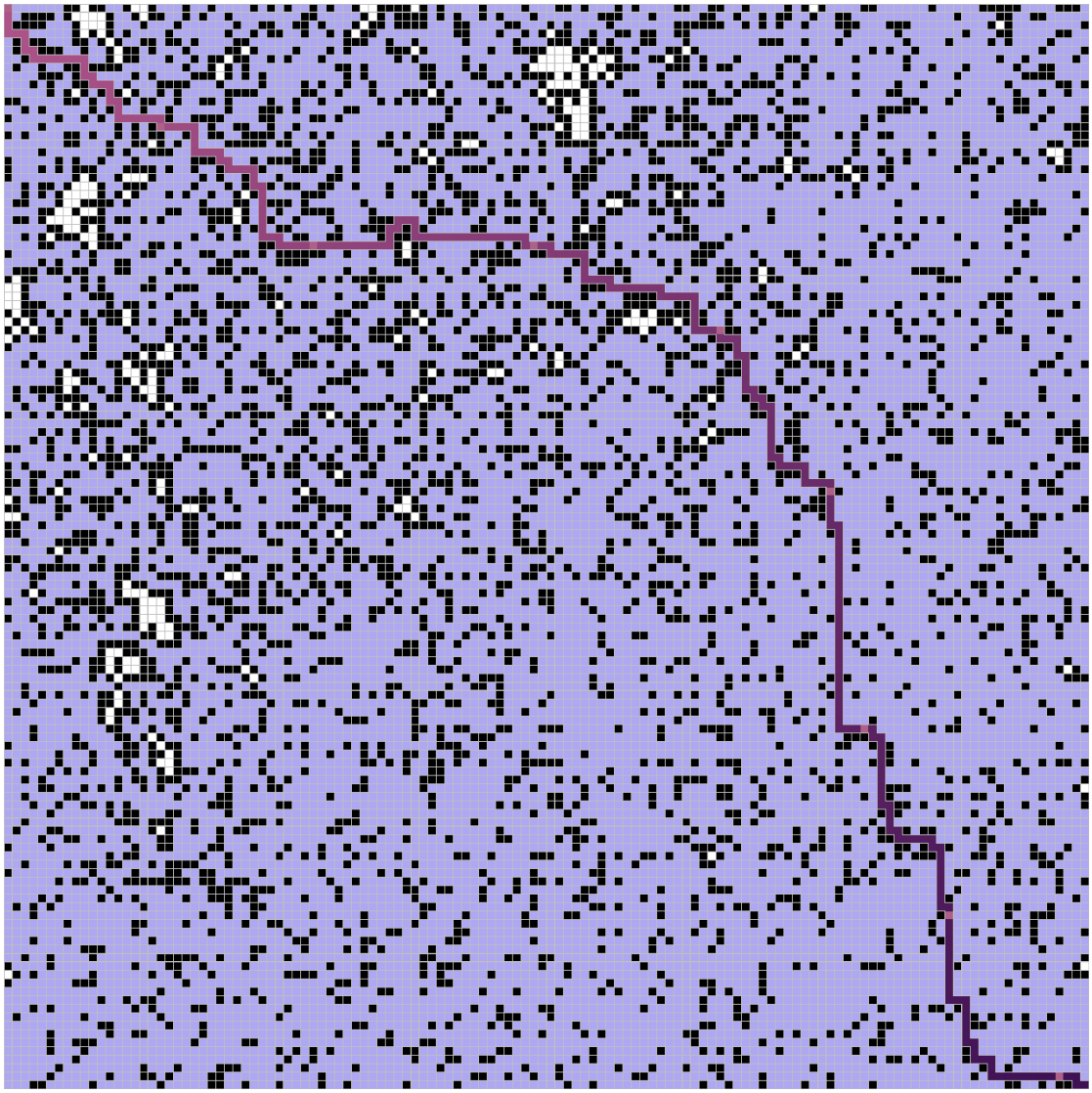

Figure 50: A maze that is hard for BFS in terms of space complexity, whose maximum fringe size is 308.

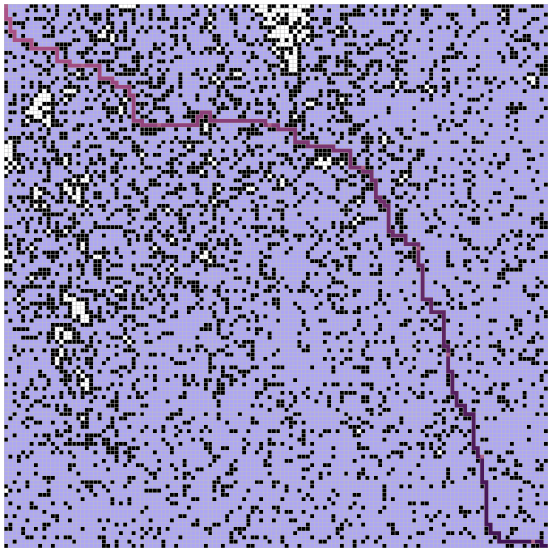

Figure 51: A maze that is hard for BFS in terms of space complexity, whose maximum fringe size is 307.

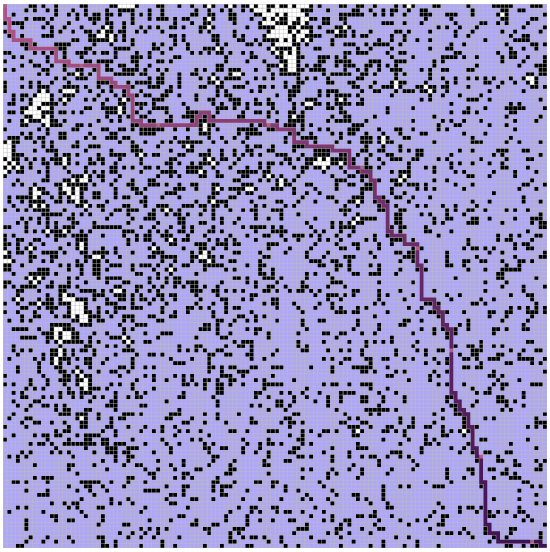

Figure 52: A maze that is hard for BFS in terms of space complexity, whose maximum fringe size is 306.

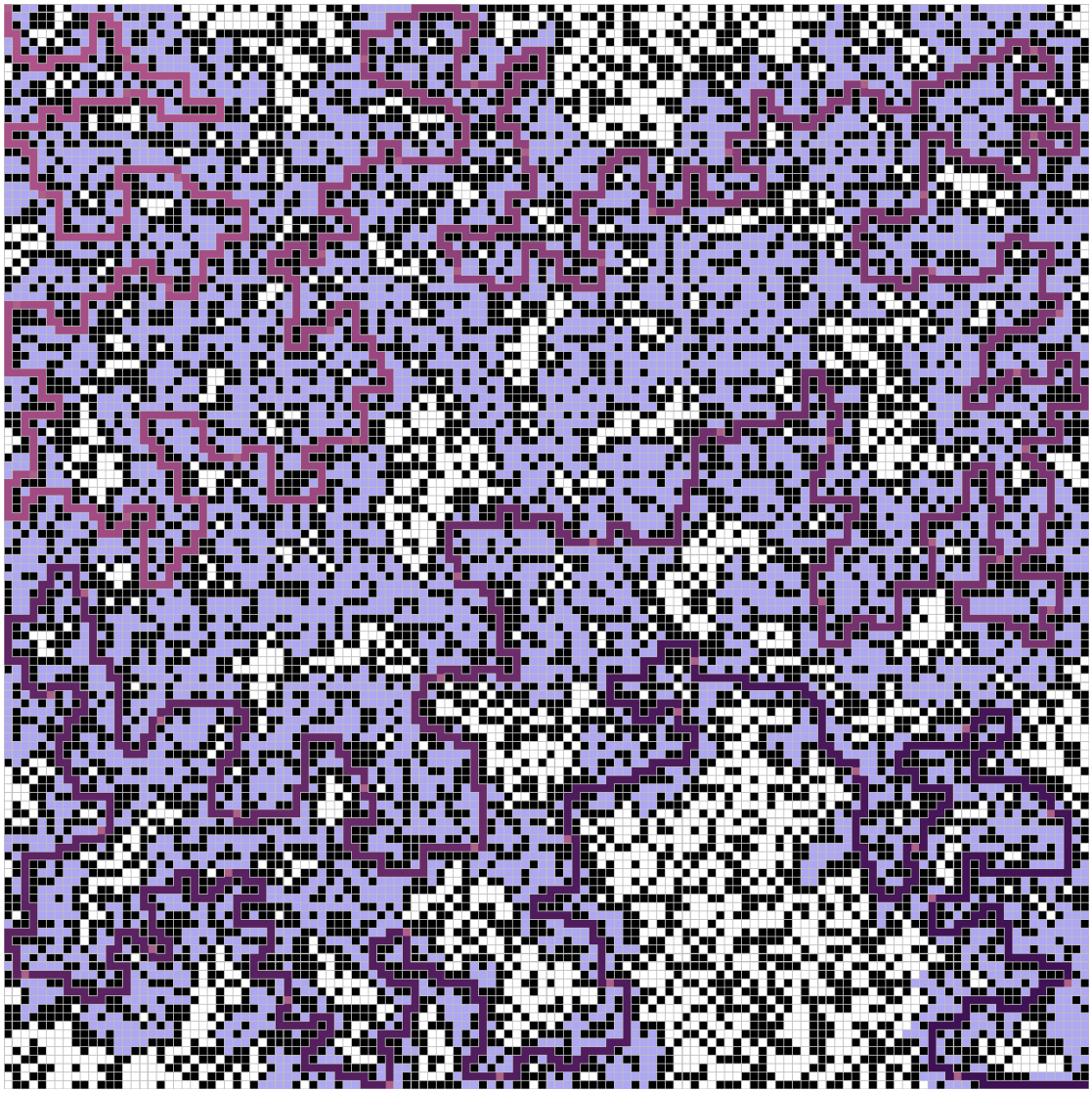

Figure 53: maze that is hard for A^* with Euclidean Distance Heuristic in terms of path length, which is 2033.

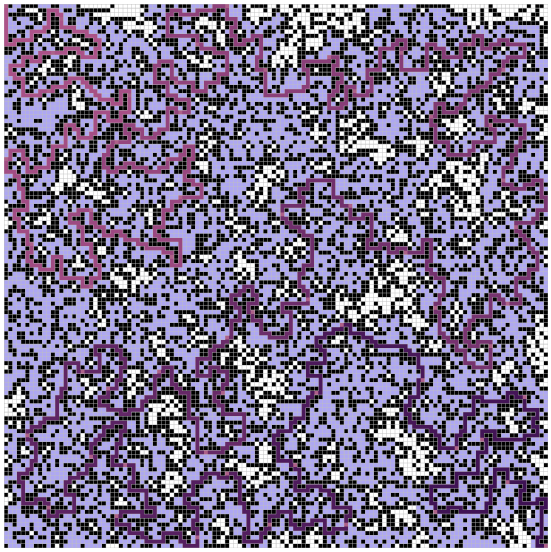

Figure 54: maze that is hard for A^* with Euclidean Distance Heuristic in terms of path length, which is 1921.

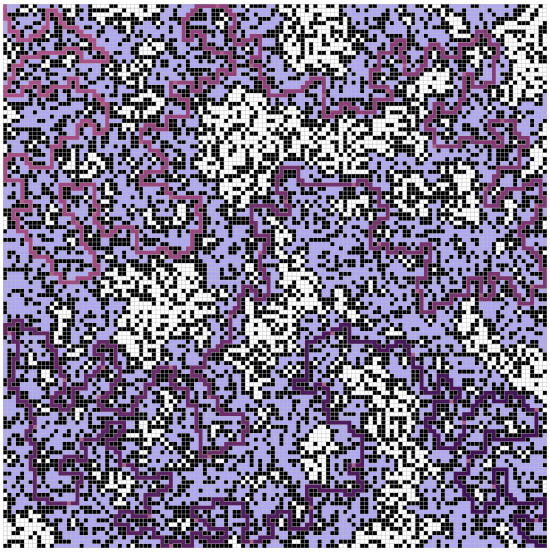

Figure 55: maze that is hard for A^* with Euclidean Distance Heuristic in terms of path length, which is 1909.

- "J- shape" walls force agents search all blocks inside because agents prefer to go right or left than to move up.

"The hardest" maze:

- The ideal "J-shape" walls can be maze-sized. (See Fig 36) In this case, the path length can be $size \times size - WallCost = 16312$. Notice that if size is an odd number, not all the accessible blocks are in the path. Specifically, only the first 3 blocks in the 2nd row are in the path.
- Compared with 16312, 5557 is not a big number. However, it is extremely impossible to randomly set all walls in a line.
- However, compared with less than 1000 block length in a 200 by 200 maze(from Question 5), 5557 in a 128 by 128 maze can be regarded as a "harder" maze, though it is not the "hardest" maze.

ii. Total number of nodes expanded: See Figure 56, Figure 57 and Figure 58.

Patterns or trends:

- The pathway near the Goal(10 * 10 blocks in the lower right corner) is extremely narrow(only 1 block width), opens to left, and begins with a "door sill" wall that forces the path to move up a block.
- "I" and ">" walls and "J-shape" walls appear alternately.

Reasons:

- Notice that when randomWalk == False, the priority of 4 directions is Right > Down > Left > Up.
- The pathway opens to left rather than up makes A^* with Euclidean Distance Heuristic cannot first run right to the last column and then find the pathway. It takes time for A^* with Euclidean Distance Heuristic to search all upper right part(above the path) and then realize "Oops, there is no way to the goal. I have to backtrack."
- After A^* with Euclidean Distance Heuristic reached the "door sill", it prefers to go to any directions but up. Hence, it takes more time for A^* with Euclidean Distance Heuristic to search all lower left part(below the path) and then realize "Alright, I will go up to see if there is a path."
- Therefore, A^* with Euclidean Distance Heuristic nearly searched all the blocks to reach the Goal.
- Notice that if randomWalk == True, this pattern will not always work because A^* with Euclidean Distance Heuristic may go down first rather than right.

"The hardest" maze:

- A maze is "empty"(no walls) except the lower right corner. The lower right corner should be a pathway like this pattern. The smallest pattern can be done in 4 * 4.(See Fig 40). Notice there must be at least 2 block height "door sill" for A^* with Euclidean Distance Heuristic to turn back to search lower left part.
- Compare to $size \times size - WallCost = 16377$, 12993 can still be improved. However, it is really difficult to cancel out those outside walls by random without destroying the pattern. Therefore, it has been a nearly "hardest" maze.

iii. Maximum size of fringe during runtime: See Figure 59, Figure 60 and Figure 61.

Patterns or trends:

- The same pattern as Question 10.b.ii, but there should be a clear path along the first row and upper half of the last column.

Reasons:

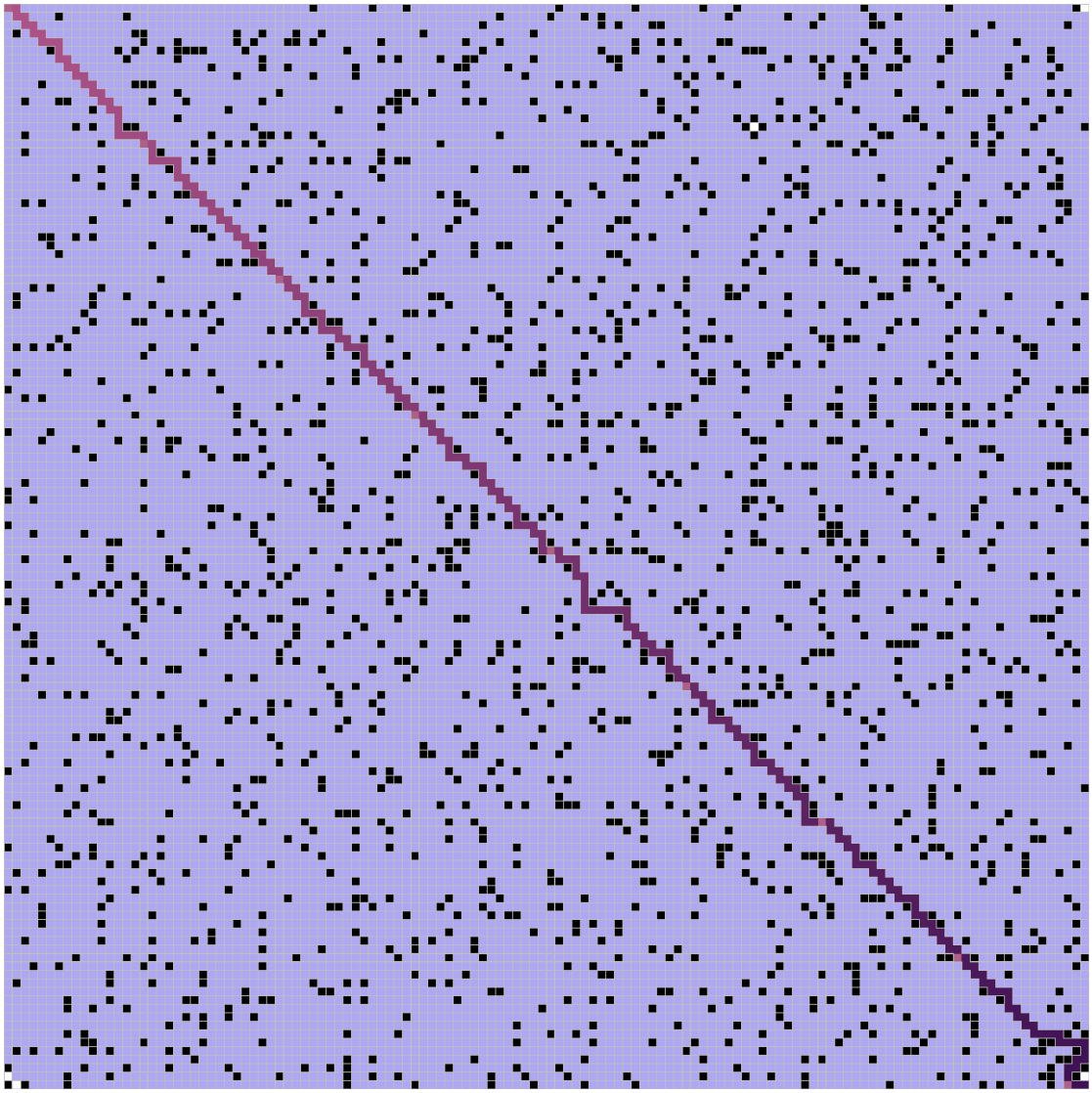

Figure 56: A maze that is hard for A^* with Euclidean Distance Heuristic in terms of time complexity, which has opend 14586 blocks.

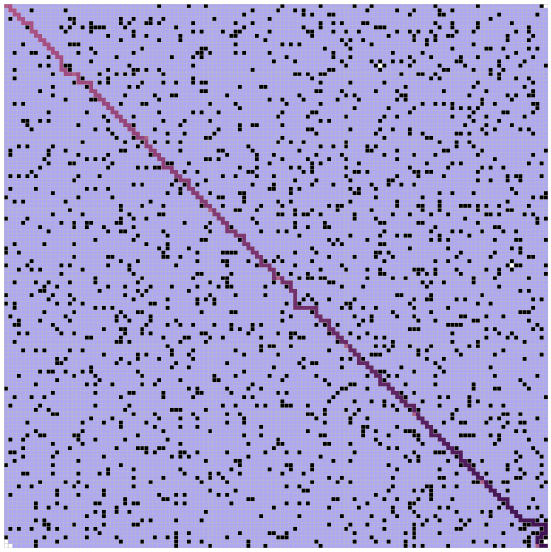

Figure 57: A maze that is hard for A^* with Euclidean Distance Heuristic in terms of time complexity, which has opend 14582 blocks.

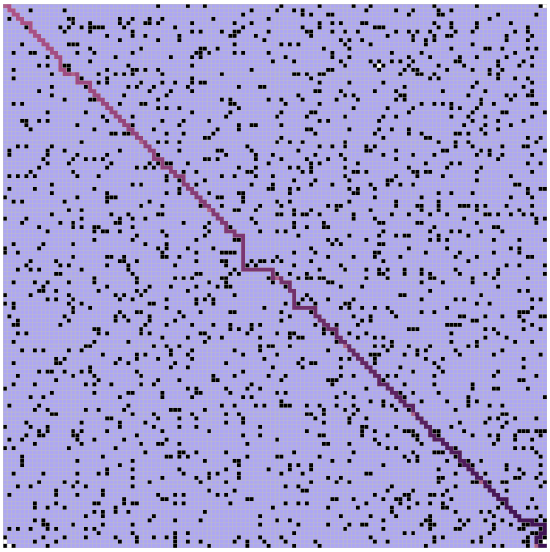

Figure 58: A maze that is hard for A^* with Euclidean Distance Heuristic in terms of time complexity, which has opend 14581 blocks.

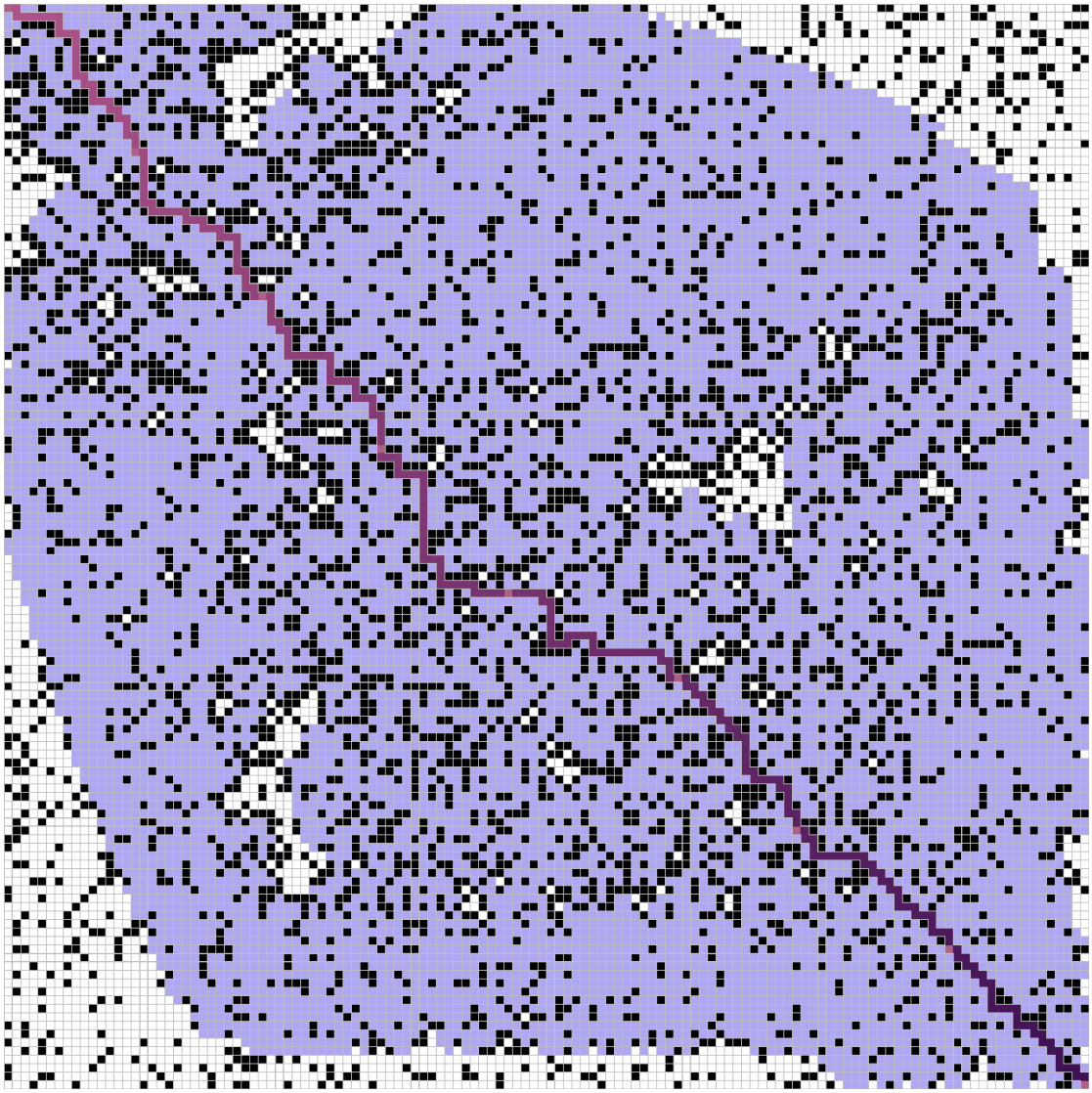

Figure 59: A maze that is hard for A^* with Euclidean Distance Heuristic in terms of space complexity, whose maximum fringe size is 636.

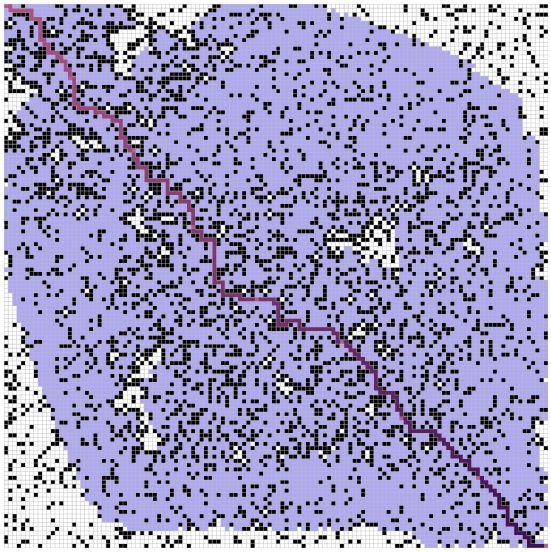

Figure 60: A maze that is hard for A^* with Euclidean Distance Heuristic in terms of space complexity, whose maximum fringe size is 634.

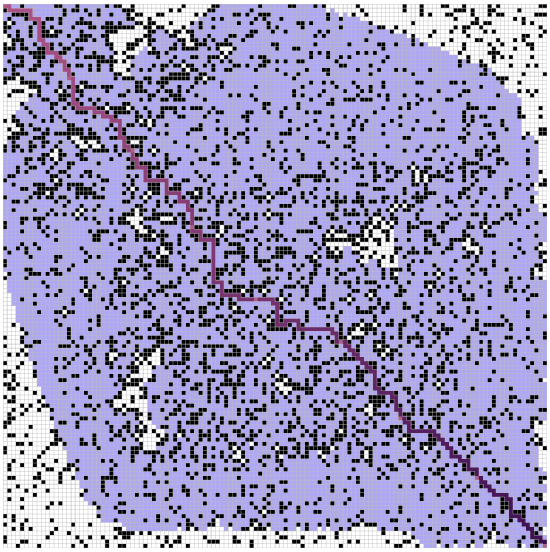

Figure 61: A maze that is hard for A^* with Euclidean Distance Heuristic in terms of space complexity, whose maximum fringe size is 634.

- The time complexity of A^* with Euclidean Distance Heuristic related to the maximum depth it has searched.
- Hence, a "door sill" pattern can force A^* with Euclidean Distance Heuristic to search lower left part rather than immediately reach the Goal, which increased the maximum depth of A^* with Euclidean Distance Heuristic.
- A larger lower left part increases the depth, so the upper right part should be as small as possible.

"The hardest" maze:

- The same maze as Question 10.b.ii.
- The maximum fringe size is theoretically $(size - 1)^2 - WallCost = 16113$, compared with 10032. Noticed distinctFringe actually do not promise all elements in the fringe are distinct because it takes too much time to search in a stack frequently. (Further information is available in description.md)
- Again, it is really difficult to cancel out those outside walls by random without destroying the pattern. Therefore, it has been a nearly "hardest" maze.

(d) A^* with Manhattan Distance Heuristic

i. Length of solution path returned: See Figure 62, Figure 63 and Figure 64.

Patterns or trends:

- There are 2 classical patterns of walls. (See Figure 35) In one case, walls lead agents to go down some rows, then there usually several nearly "empty"(no walls) rows to move horizontally. But at the end of these rows, diagonal walls lead agents to move up. This case looks like "I" and ">". The other case is "J-shape" walls.
- "I" and ">" walls and "J-shape" walls appear alternately.

Reasons:

- The key point is when randomWalk == False, the priority of 4 directions is Right > Down > Left > Up.
- "I" and ">" walls lead agent walk horizontally, and the path it has walked becomes "new wall" because those blocks have been added into the closed set. Now it becomes a "J-shape" walls in the mirror.
- "J- shape" walls force agents search all blocks inside because agents prefer to go right or left than to move up.

"The hardest" maze:

- The ideal "J-shape" walls can be maze-sized. (See Fig 36) In this case, the path length can be $size \times size - WallCost = 16312$. Notice that if size is an odd number, not all the accessible blocks are in the path. Specifically, only the first 3 blocks in the 2nd row are in the path.
- Compared with 16312, 5557 is not a big number. However, it is extremely impossible to randomly set all walls in a line.
- However, compared with less than 1000 block length in a 200 by 200 maze(from Question 5), 5557 in a 128 by 128 maze can be regarded as a "harder" maze, though it is not the "hardest" maze.

ii. Total number of nodes expanded: See Figure 32, Figure 33 and Figure 34.

Patterns or trends:

- The pathway near the Goal(10 * 10 blocks in the lower right corner) is extremely narrow(only 1 block width), opens to left, and begins with a "door sill" wall that forces the path to move up a block.
- "I" and ">" walls and "J-shape" walls appear alternately.

Reasons:

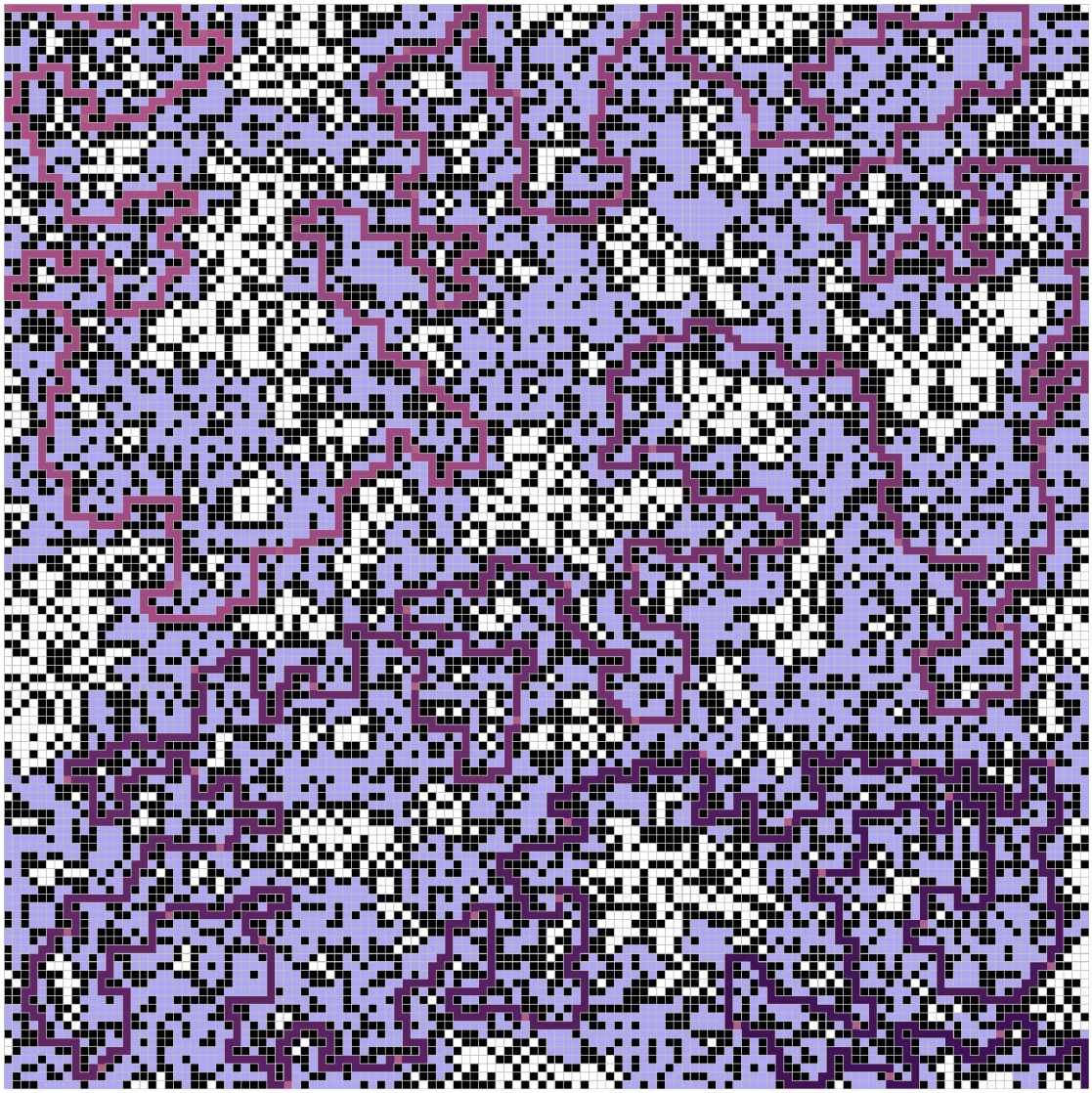

Figure 62: maze that is hard for A^* with Manhattan Distance Heuristic in terms of path length, which is 2171.

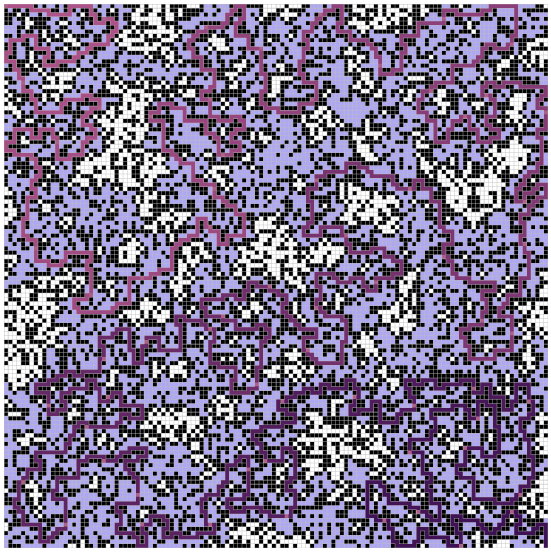

Figure 63: maze that is hard for A^* with Manhattan Distance Heuristic in terms of path length, which is 2167.

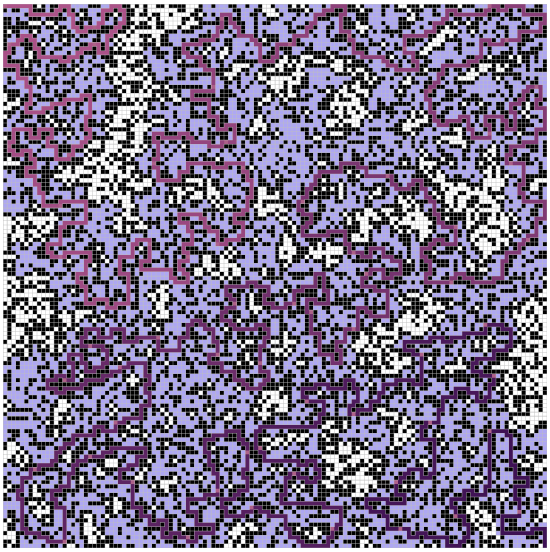

Figure 64: maze that is hard for A^* with Manhattan Distance Heuristic in terms of path length, which is 2137.

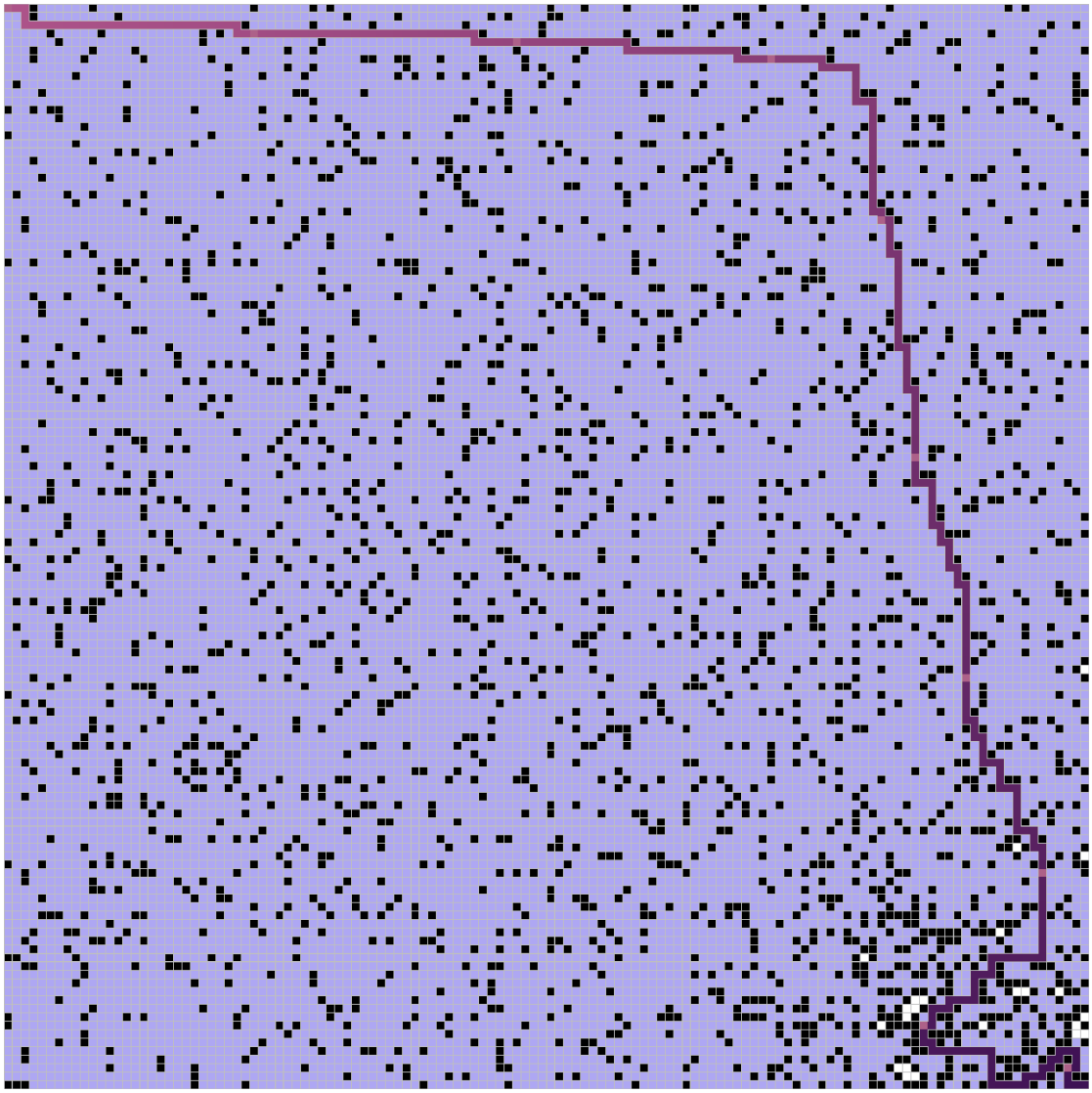

Figure 65: A maze that is hard for A^* with Manhattan Distance Heuristic in terms of time complexity, which has opend 14210 blocks.

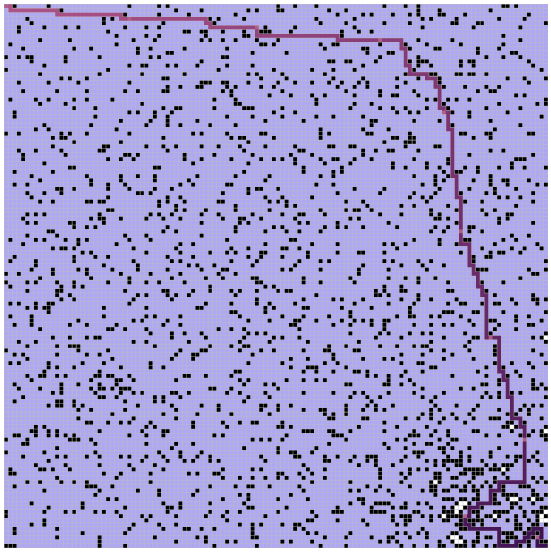

Figure 66: A maze that is hard for A^* with Manhattan Distance Heuristic in terms of time complexity, which has opend 14209 blocks.

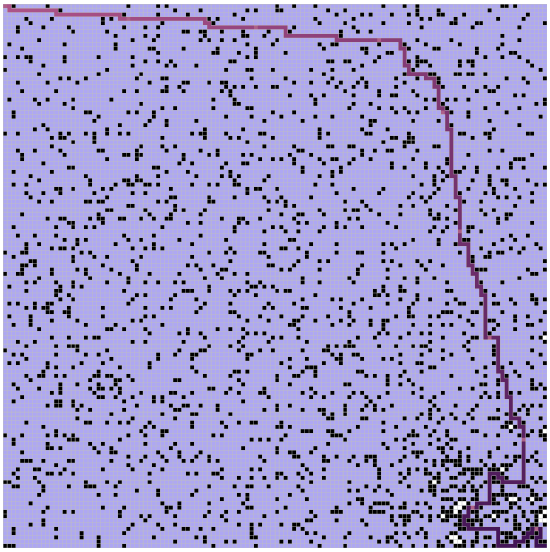

Figure 67: A maze that is hard for A^* with Manhattan Distance Heuristic in terms of time complexity, which has opend 14205 blocks.

- Notice that when randomWalk == False, the priority of 4 directions is Right > Down > Left > Up.
- The pathway opens to left rather than up makes A^* with Manhattan Distance Heuristic cannot first run right to the last column and then find the pathway. It takes time for A^* with Manhattan Distance Heuristic to search all upper right part(above the path) and then realize "Oops, there is no way to the goal. I have to backtrack."
- After A^* with Manhattan Distance Heuristic reached the "door sill", it prefers to go to any directions but up. Hence, it takes more time for A^* with Manhattan Distance Heuristic to search all lower left part(below the path) and then realize "Alright, I will go up to see if there is a path."
- Therefore, A^* with Manhattan Distance Heuristic nearly searched all the blocks to reach the Goal.
- Notice that if randomWalk == True, this pattern will not always work because A^* with Manhattan Distance Heuristic may go down first rather than right.

"The hardest" maze:

- Compare to $size \times size - WallCost = 16377$, 12993 can still be improved. However, it is really difficult to cancel out those outside walls by random without destroying the pattern. Therefore, it has been a nearly "hardest" maze.

iii. Maximum size of fringe during runtime: See Figure 68, Figure 69 and Figure 70.

Patterns or trends:

- The same pattern as Question 10.b.ii, but there should be a clear path along the first row and upper half of the last column.

Reasons:

- The time complexity of A^* with Manhattan Distance Heuristic related to the maximum depth it has searched.
- Hence, a "door sill" pattern can force A^* with Manhattan Distance Heuristic to search lower left part rather than immediately reach the Goal, which increased the maximum depth of A^* with Manhattan Distance Heuristic.
- A larger lower left part increases the depth, so the upper right part should be as small as possible.

"The hardest" maze:

- The same maze as Question 10.b.ii.
- The maximum fringe size is theoretically $(size - 1)^2 - WallCost = 16113$, compared with 10032. Noticed distinctFringe actually do not promise all elements in the fringe are distinct because it takes too much time to search in a stack frequently. (Further information is available in description.md)
- Again, it is really difficult to cancel out those outside walls by random without destroying the pattern. Therefore, it has been a nearly "hardest" maze.

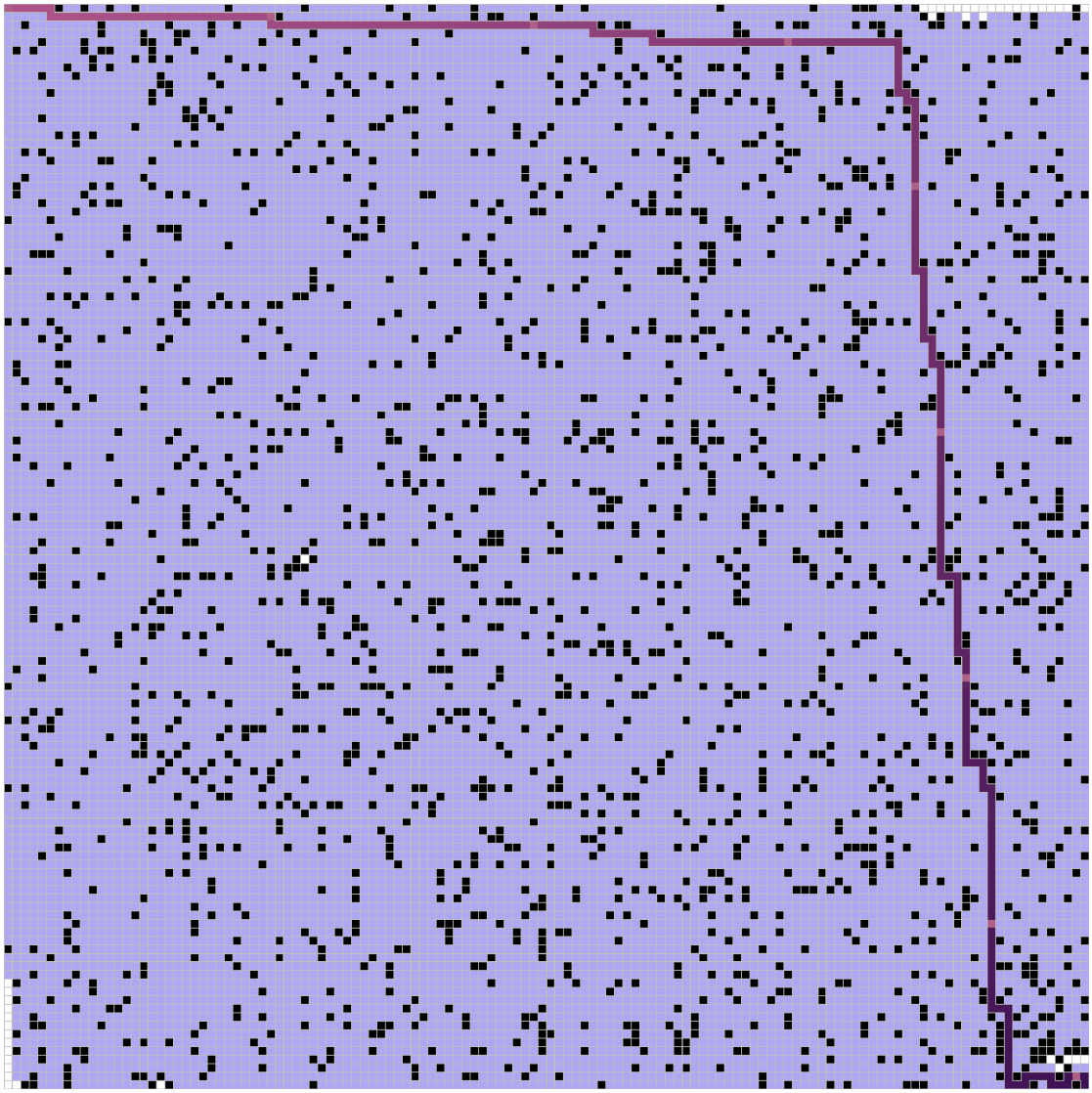

Figure 68: A maze that is hard for A^* with Manhattan Distance Heuristic in terms of space complexity, whose maximum fringe size is 9720.

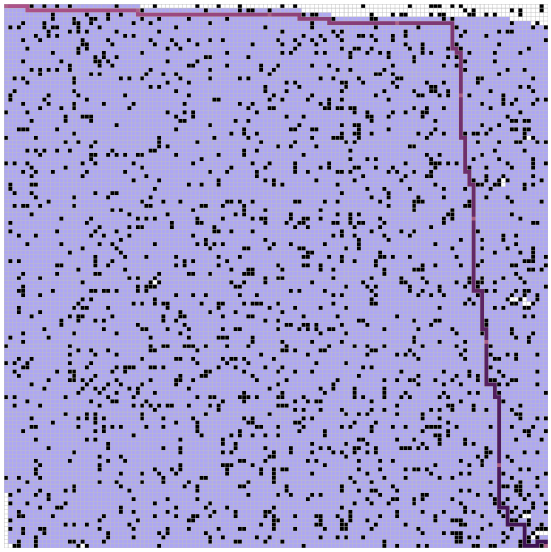

Figure 69: A maze that is hard for A^* with Manhattan Distance Heuristic in terms of space complexity, whose maximum fringe size is 9716.

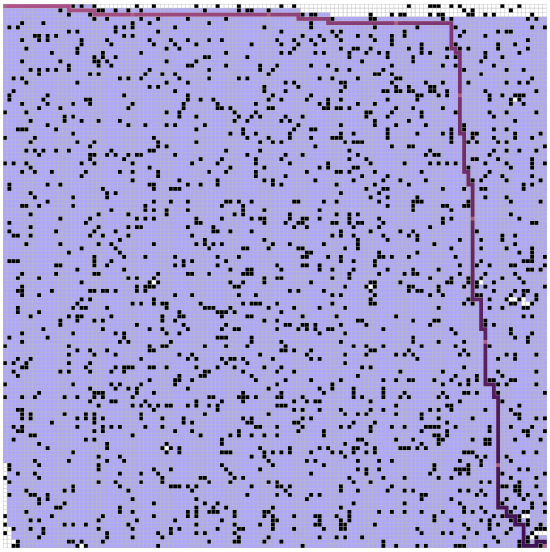

Figure 70: A maze that is hard for A^* with Manhattan Distance Heuristic in terms of space complexity, whose maximum fringe size is 9696.