Fast Trajectory Replanning Report

Xinyu Meng netid:xm73 RUID:193002567
Qichao Lin netid:ql180 RUID:196005340

Feb 21 2023

1 Part 1

1.1 a

Explain in your report why the first move of the agent for the example search problem from Figure 8 is to the east rather than the north given that the agent does not know initially which cells are blocked.

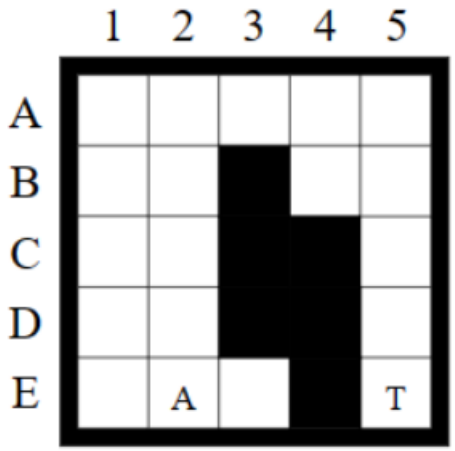


Figure 8: Second Example Search Problem

In the above example, based on the A* algorithm, the north node has g = 1, h = 3+1 = 4, therefore f = g+h = 5.

The east node has g = 1, h = 2, f = 3. The first iteration of A* will make the agent go straight to the right as it will pick the lowest f in the open list. Therefore the execution will follow what A* initially planned to move to the east.

1.2 b

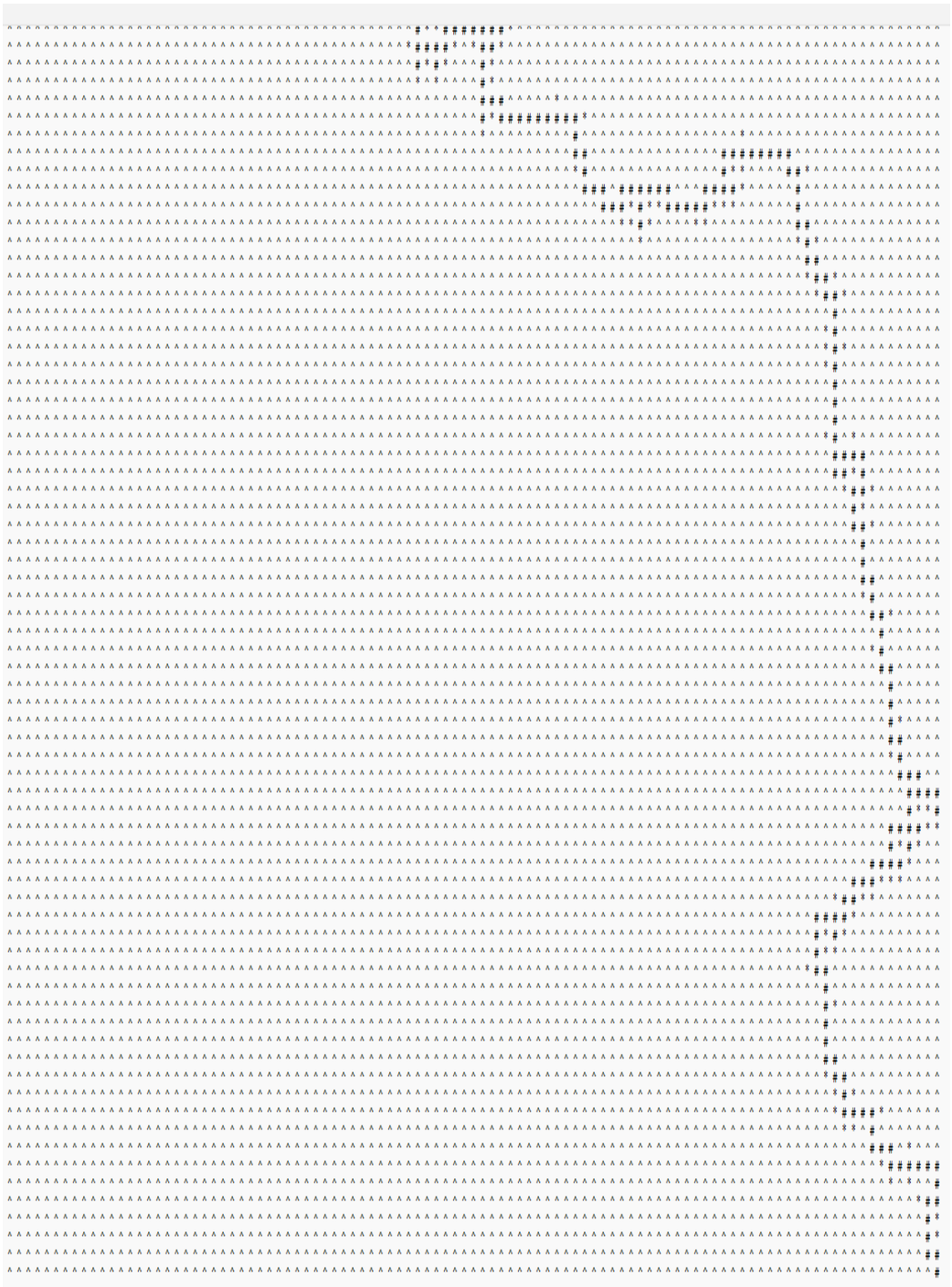
This project argues that the agent is guaranteed to reach the target if it is not separated from it by blocked cells. Give a convincing argument that the agent in finite gridworlds indeed either reaches the target or discovers that this is impossible in finite time. Prove that the number of moves of the agent until it reaches the target or discovers that this is impossible is bounded from above by the number of unblocked cells squared.

To prove that repeated A* cannot repeat forever in a grid where there is a path to the goal from the initial state, we can use the fact that A* is guaranteed to find the shortest path to the goal if one exists. Therefore, if the agent is moving in an infinite loop, it means that there is no shorter path to the goal than the one it is currently following. However, this contradicts the assumption that there is a path to the goal from the initial state, since if there is a path, the A* will in fact return the correct path to the goal. If the A* algorithm reaches a dead end, where there are no more unexplored cells that are reachable from the current cell, it will backtrack to the previous cell and continue exploring from there. However, since the A* algorithm evaluates cells in order of increasing cost, it will eventually evaluate the cell that leads to a path to the target cell, and it will terminate.Also, as the A* has a mechanism to record the obstacles explored by the executed path, A* will return null if there is no possible path based on the explored map, which will stops the execution if there is not possible path exist.

Moreover, in the worst case, the agent will execute all the unblocked nodes before get stuck. Then, the agent will keep planning and execute all the unblocked nodes again until it reaches a block. Under such scenario, the agent can only start an A* from a unblocked node once and therefore the total number of executions of the agent in the worst scenario is bounded by the square of the number of the unblocked nodes.

2 Part 2 - The Effect of Ties

path found by repeated A* with normal priority
execution time: 1.068s



```
path length: 258
Execution time: 1.0684514045715332
```

path found by repeated A* with priority = 203*f - g 203 means this constant is greater than any g value

execution time: 0.723s

path length: 254
Execution time: 0.7230331897735596

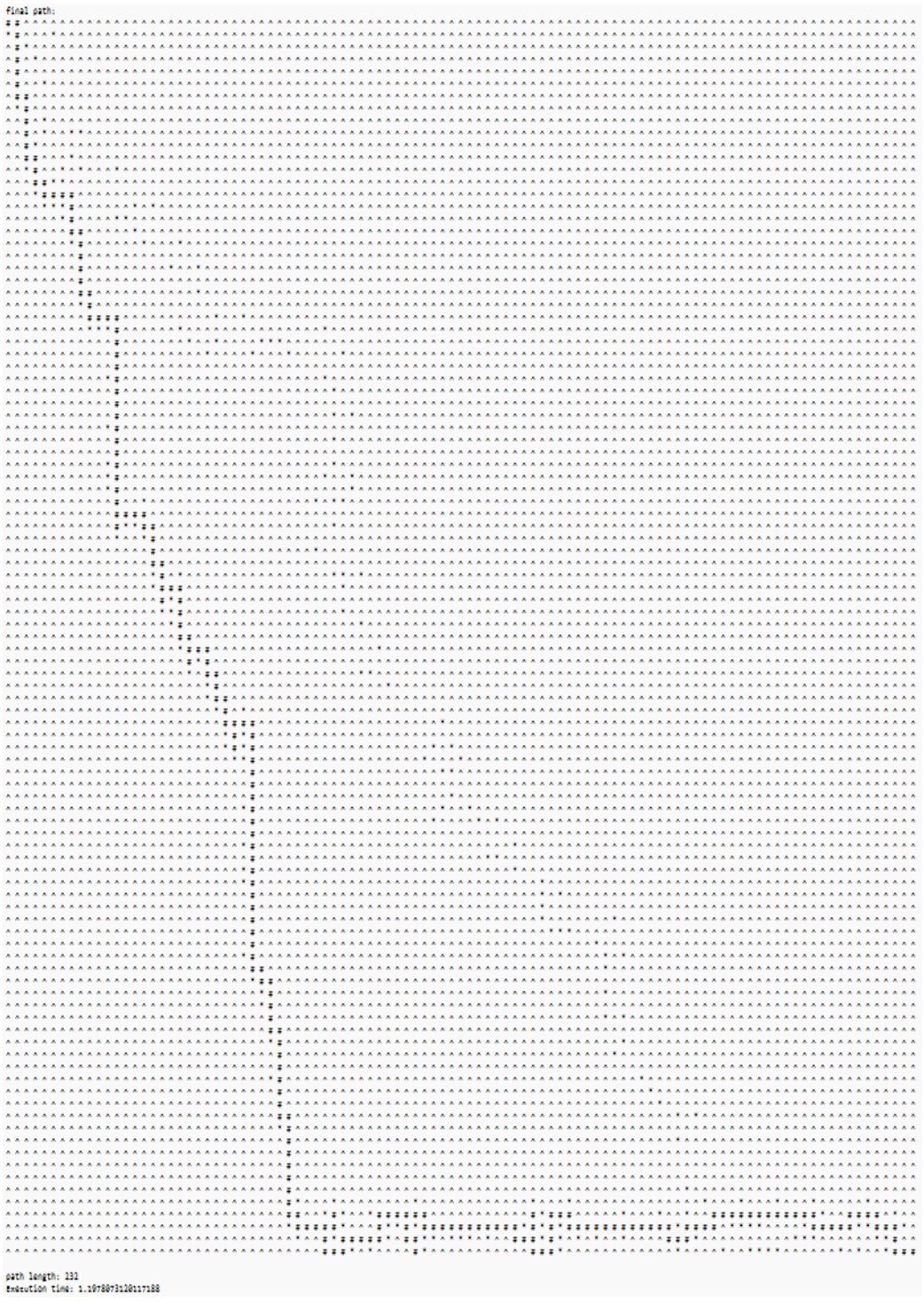
Since we use priority = cf - g where f = g+h

priority = c(g+h) - g = cg + ch - g = (c-1)g + ch

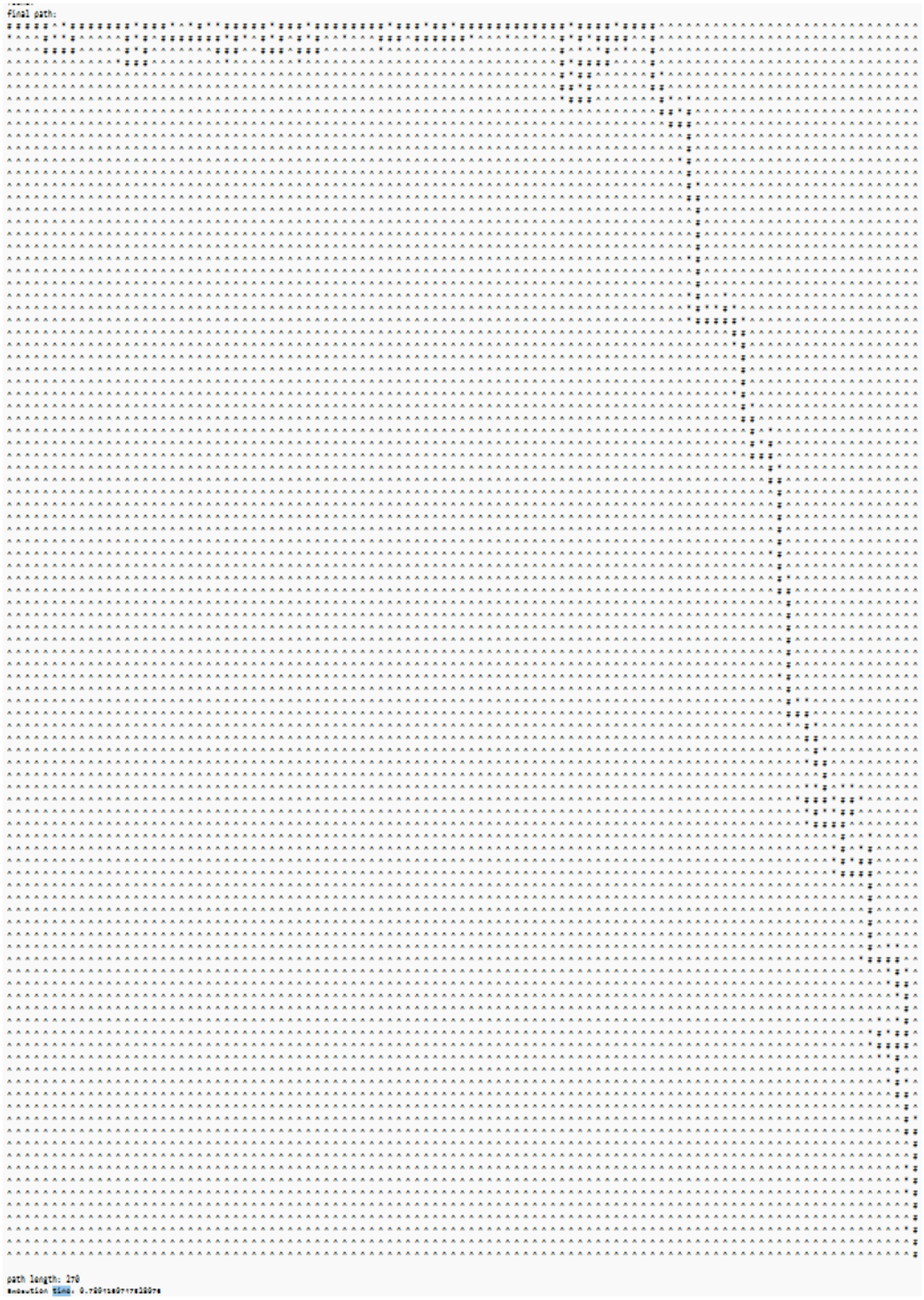
g has higher weigh in the priority, when tie-breaking in favor of cells with smaller g-values, the algorithm tends to prioritize expanding cells that are closer to the start state, which are likely to be part of a shorter path to the goal. On the other hand, when tie-breaking in favor of cells with larger g-values, the algorithm tends to prioritize expanding cells that are further from the start state, which may be part of longer paths to the goal.

3 Part3

Forward A* execution time: 1.1978s


final path:
path length: 232
execution time: 1.1978073120117188

Backward A* execution time: 0.7894s


final path:
path length: 270

In most situations Repeated Backward A* performs better than Repeated Forward A*, in terms of both number of expanded cells and runtime. This is because Repeated Backward A* algorithm may be able to more quickly identify the shortest path to the goal by first exploring the cells closer to the goal state, which are more likely to be part of the shortest path. In contrast, Repeated Forward A* may waste time exploring parts of the grid that are not relevant to the shortest path. But in some mazes, Forward A* may be faster than Backward A* due to the setting of obstacles.

4 Part4

Consistent heuristic meets the condition that the estimated cost of reaching a goal from a given node is always less than or equal to the cost of reaching any successor node, plus the estimated cost of reaching the goal from that successor node. Therefore, h(n) ¡= c(n, a, n') + h(n').

Consider two adjacent nodes, n and n' assume that the agent can only move in the four main compass directions (north, south, east, west). Then the cost of moving from n to n' is always 1, regardless of the action taken. h(n) = abs(n.x-g.x) + abs(n.y-g.y) and h(n') = abs(n'.x-g.x) + abs(n'.y-g.y)

by triangle inequality:

h(n) = abs(n.x-n'.x+n'.x-g.x) + abs(n.y-n'.y+n'.y-g.y) and h(n') = abs(n'.x-n.x+n.x-g.x) + abs(n'.y-n.y+n.y-g.y)

after simplify, we get

h(n) = h(n')+1+abs(n.x-n'.x)+abs(n.y-n'.y)-2abs(n'.x-n.x)-2abs(n'.y-n.y)

since the agent can only move in the four main compass directions, abs(n.x-n'.x) and abs(n.y-n'.y) can only be 0 or 1. Therefore, we can simplify the inequality to:

h(n) ¡= h(n') + 1, which proves that the Manhattan distance heuristic is consistent in gridworlds where the agent can only move in the four main compass directions.

Adaptive A* leaves initially consistent h-values consistent even if action costs can increase.

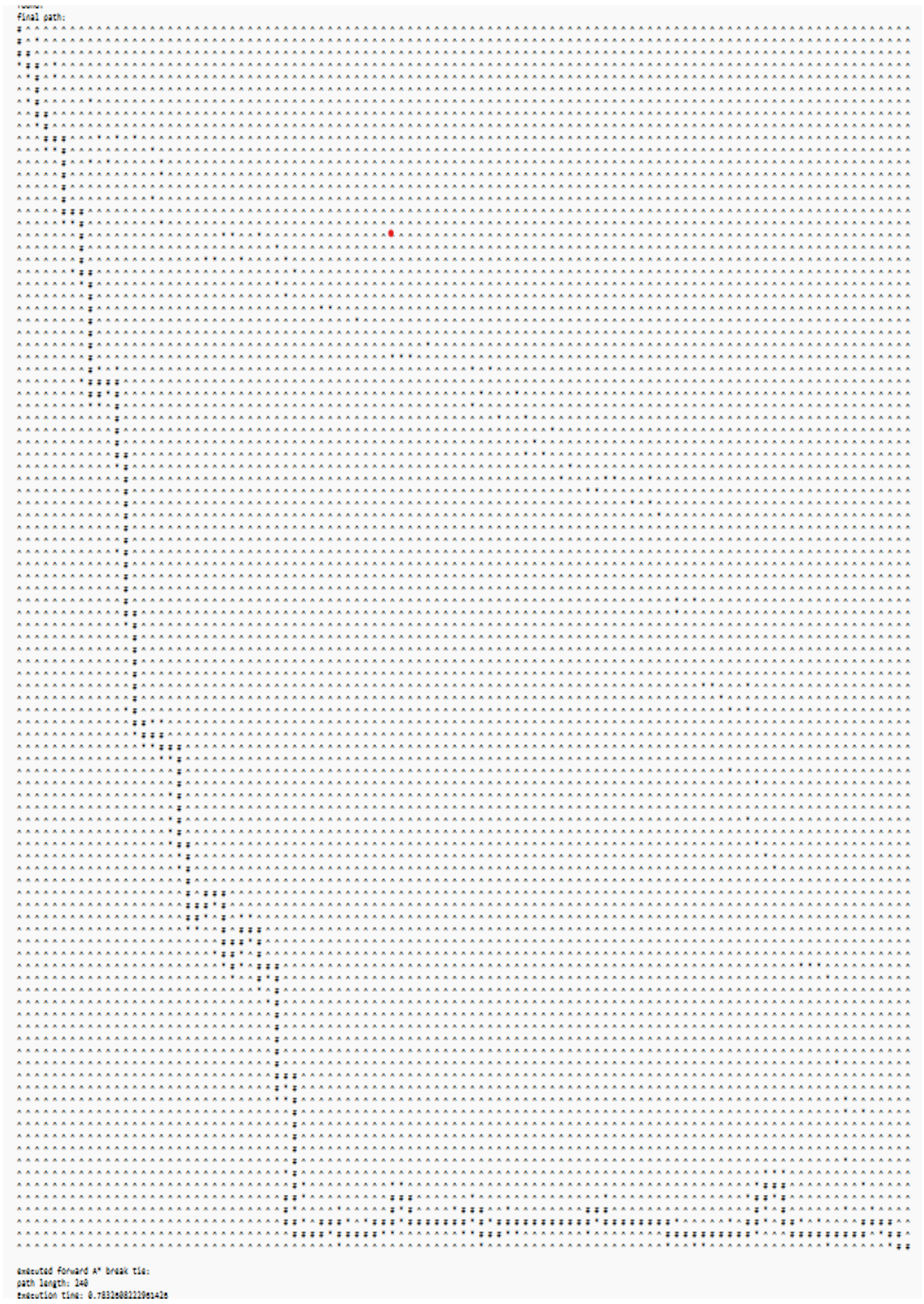
Suppose we have a consistent heuristic function h(s) that is initially admissible for a gridworld with action costs that may increase. Let hnew(s) be the new heuristic function obtained by updating the estimated costs of reaching the goal from each node s, based on the actual costs of reaching s from the start node. If the h-values consistent, h(n) ¡= c(n, n') + h(n'),

where c(n, n') is the cost of moving from n to n'.

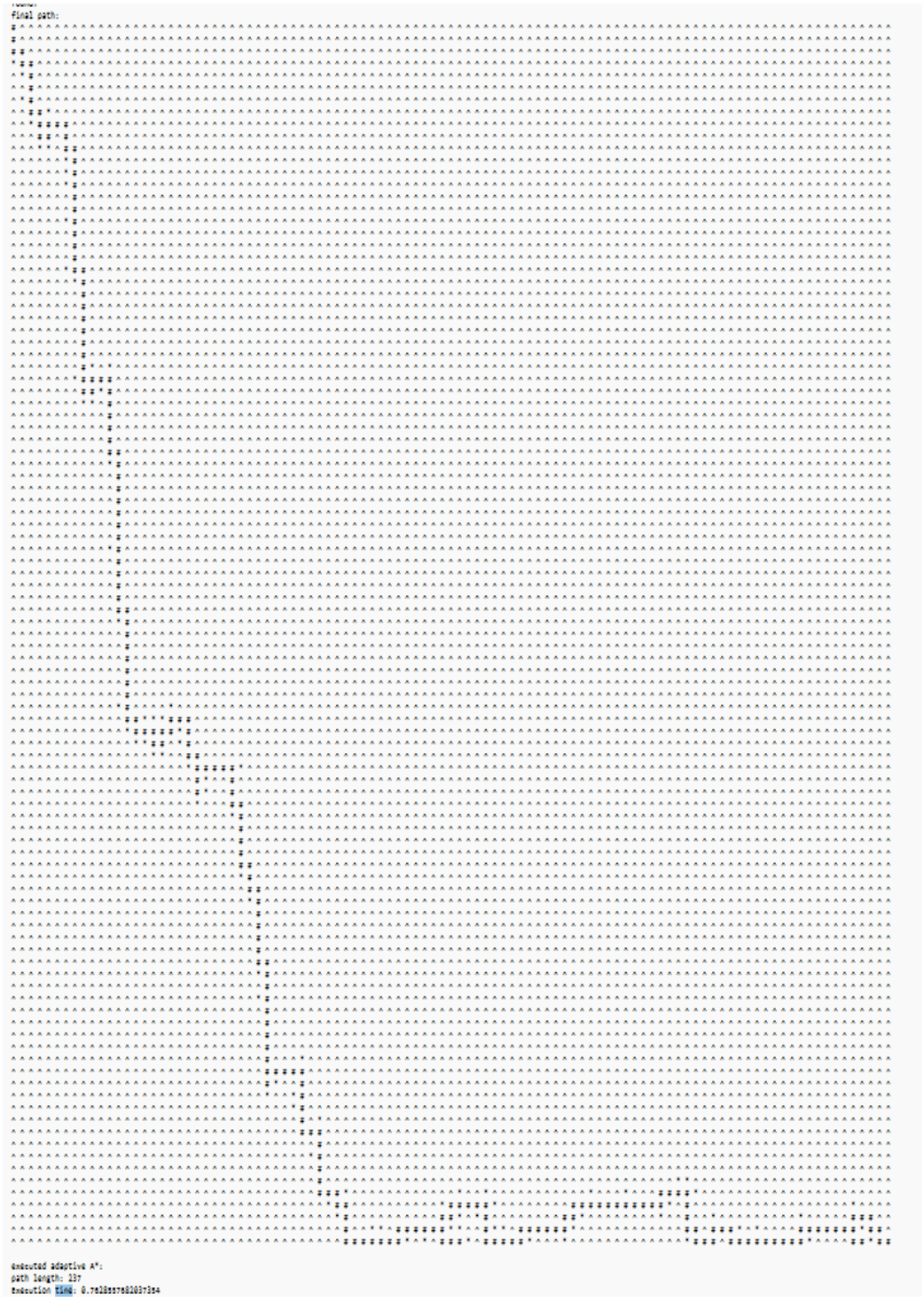
In Adaptive A*, the heuristic values h(n) are updated based on the actual costs of the paths found during the search, rather than being fixed at the beginning of the search. Specifically, when a new path is found that is cheaper than the previously known path to a node n, the heuristic value h(n) is updated to the actual cost of the new path plus the estimated cost of reaching the goal from the end of the new path. This new value of h(n) is guaranteed to be admissible, as it is based on the actual cost of the path found. Suppose that we have two nodes n and n' such that h(n) ¿ c(n, n') + h(n'). If the cost of moving from n to n' increases, the actual cost of the path from the start node to n' may increase as well. However, when Adaptive A* considers the node n again, it will find a cheaper path to n' that takes into account the increased action costs. Specifically, the actual cost of the path from the start node to n' will be updated, and the heuristic value h(n) will be updated to the actual cost of the new path plus the estimated cost of reaching the goal from n'. Because the estimated cost of reaching the goal from n' is fixed and the new path has a higher actual cost, the new value of h(n) will still be less than c(n, n') + h(n'). Therefore, the consistency of the heuristic function is maintained even if the action costs increase. Therefore, the Adaptive A* will keeps the h values consistent.

5 Part5

Forward A* tie-breaking execution time: 0.7832608222961426


final path:
executed forward A* break tie:
path length: 248
execution time: 0.7831e0822109e142e

Adaptive A* tie-breaking execution time: 0.7628557682037354


final path:
executed adaptive A*:
path length: 237
execution time: 0.7628557681203735

Repeated Adaptive A* perform better than Repeated Forward A* in some cases because it has access to more information about the search state and can adapt its heuristic values more efficiently. However, the improvement in performance may not be significant in all cases, and the specific characteristics of each maze may affect the runtime differently for each algorithm.

Also, the Repeated Forward A* perform better in some other mazes. This is because the Repeated Forward A* seems gets more advantages by using the tie-breaking approach over Adaptive A*. The overall direction for the final paths for each A* is similar while they still vary in details.

6 Part6

On the problem above "Forward vs Backward" we can perform a statistical hypothesis to verify if the repeated Forward A* and repeated Backward A* have a systematic difference in their runtime.

t Table

cum. prob	$t_{.50}$	$t_{.75}$	$t_{.80}$	$t_{.85}$	$t_{.90}$	$t_{.95}$	$t_{.975}$	$t_{.99}$	$t_{.995}$	$t_{.999}$	$t_{.9995}$
one-tail	**0.50**	**0.25**	**0.20**	**0.15**	**0.10**	**0.05**	**0.025**	**0.01**	**0.005**	**0.001**	**0.0005**
two-tails	**1.00**	**0.50**	**0.40**	**0.30**	**0.20**	**0.10**	**0.05**	**0.02**	**0.01**	**0.002**	**0.001**
df											
1	0.000	1.000	1.376	1.963	3.078	6.314	12.71	31.82	63.66	318.31	636.62
2	0.000	0.816	1.061	1.386	1.886	2.920	4.303	6.965	9.925	22.327	31.599
3	0.000	0.765	0.978	1.250	1.638	2.353	3.182	4.541	5.841	10.215	12.924
4	0.000	0.741	0.941	1.190	1.533	2.132	2.776	3.747	4.604	7.173	8.610
5	0.000	0.727	0.920	1.156	1.476	2.015	2.571	3.365	4.032	5.893	6.869
6	0.000	0.718	0.906	1.134	1.440	1.943	2.447	3.143	3.707	5.208	5.959
7	0.000	0.711	0.896	1.119	1.415	1.895	2.365	2.998	3.499	4.785	5.408
8	0.000	0.706	0.889	1.108	1.397	1.860	2.306	2.896	3.355	4.501	5.041
9	0.000	0.703	0.883	1.100	1.383	1.833	2.262	2.821	3.250	4.297	4.781
10	0.000	0.700	0.879	1.093	1.372	1.812	2.228	2.764	3.169	4.144	4.587
11	0.000	0.697	0.876	1.088	1.363	1.796	2.201	2.718	3.106	4.025	4.437
12	0.000	0.695	0.873	1.083	1.356	1.782	2.179	2.681	3.055	3.930	4.318
13	0.000	0.694	0.870	1.079	1.350	1.771	2.160	2.650	3.012	3.852	4.221
14	0.000	0.692	0.868	1.076	1.345	1.761	2.145	2.624	2.977	3.787	4.140
15	0.000	0.691	0.866	1.074	1.341	1.753	2.131	2.602	2.947	3.733	4.073
16	0.000	0.690	0.865	1.071	1.337	1.746	2.120	2.583	2.921	3.686	4.015
17	0.000	0.689	0.863	1.069	1.333	1.740	2.110	2.567	2.898	3.646	3.965
18	0.000	0.688	0.862	1.067	1.330	1.734	2.101	2.552	2.878	3.610	3.922
19	0.000	0.688	0.861	1.066	1.328	1.729	2.093	2.539	2.861	3.579	3.883
20	0.000	0.687	0.860	1.064	1.325	1.725	2.086	2.528	2.845	3.552	3.850
21	0.000	0.686	0.859	1.063	1.323	1.721	2.080	2.518	2.831	3.527	3.819
22	0.000	0.686	0.858	1.061	1.321	1.717	2.074	2.508	2.819	3.505	3.792
23	0.000	0.685	0.858	1.060	1.319	1.714	2.069	2.500	2.807	3.485	3.768
24	0.000	0.685	0.857	1.059	1.318	1.711	2.064	2.492	2.797	3.467	3.745
25	0.000	0.684	0.856	1.058	1.316	1.708	2.060	2.485	2.787	3.450	3.725
26	0.000	0.684	0.856	1.058	1.315	1.706	2.056	2.479	2.779	3.435	3.707
27	0.000	0.684	0.855	1.057	1.314	1.703	2.052	2.473	2.771	3.421	3.690
28	0.000	0.683	0.855	1.056	1.313	1.701	2.048	2.467	2.763	3.408	3.674
29	0.000	0.683	0.854	1.055	1.311	1.699	2.045	2.462	2.756	3.396	3.659
30	0.000	0.683	0.854	1.055	1.310	1.697	2.042	2.457	2.750	3.385	3.646
40	0.000	0.681	0.851	1.050	1.303	1.684	2.021	2.423	2.704	3.307	3.551
60	0.000	0.679	0.848	1.045	1.296	1.671	2.000	2.390	2.660	3.232	3.460
80	0.000	0.678	0.846	1.043	1.292	1.664	1.990	2.374	2.639	3.195	3.416
100	0.000	0.677	0.845	1.042	1.290	1.660	1.984	2.364	2.626	3.174	3.390
1000	0.000	0.675	0.842	1.037	1.282	1.646	1.962	2.330	2.581	3.098	3.300
z	0.000	0.674	0.842	1.036	1.282	1.645	1.960	2.326	2.576	3.090	3.291
	0%	50%	60%	70%	80%	90%	95%	98%	99%	99.8%	99.9%
	Confidence Level										

By using t-test

t = (avg1 - avg2) / (sqrt((s1**2/n1) + (s2**2/n2)))

Where avg1 is the average running time of Repeated Forward A*, avg2 is the average running time of Repeated Backward A*, s1 is the standard deviation of Repeated Forward A* running time, s2 is the standard deviation of Repeated Backward A* running time, n1 is the number of data of Forward A*, n2 is the number of data of Repeated Backward A*.

Get the degrees of freedom using: df = n1 + n2 - 2

Finding the critical t-value in the t-table using the degrees of freedom and the desired level of significance.

Compare the calculated t-value with the critical t-value. If the calculated t-value is greater than the critical t-value, we reject the null hypothesis which means there is a systematic difference in the runtime of the Repeated Forward A* and Repeated Backward A*. If the calculated t-value is less than the critical t-value, we fail to reject the null hypothesis.