CS 520 : Introduction to Artificial Intelligence
Fast Trajectory Replanning

Avinash Ramachandran (AR2041)
Sreeniketh Aathreya (SPA44)
Darshan Senthil (DS1992)

15 October, 2022

1 Introduction

1.1 Problem Statement

A two-dimensional landmass made up of cells that can either be traversed (unblocked) or not traversed (blocked) can termed as a Grid World. An agent that moves in four directions (East, West, North, South) starts at the source cell (0,0) with a goal of reaching the target cell (n,n) via the least-cost path available. Accounting to the lack of knowledge about the environment, the agent will collect information about the environments as it traverses and updates itself. After repeated trial and error, the agent will find the target in the most optimum way (least-cost path).

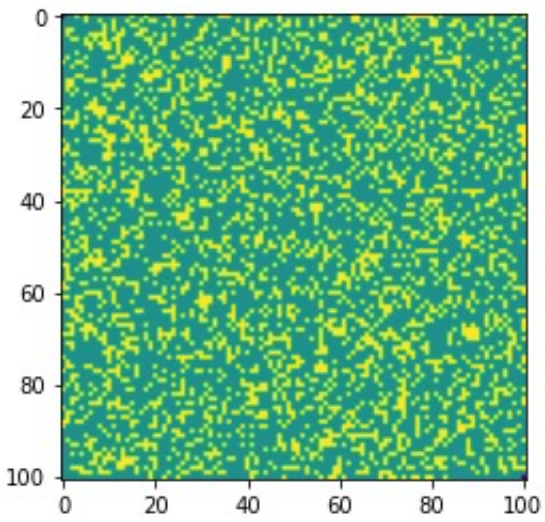


Figure 1: Example gridworld where yellow dots represent obstacles in the path

2 Part 0

The maze generation is done by populating a 2D array with nodes. The nodes are classes in Python and possess an attribute called blocked. There are d*d (with d being the dimension) nodes and the probability that a node has the 'blocked' attribute set to True is 30 percent, and that it is False is 70 percent. 50 such mazes are generated. Sample of a maze:

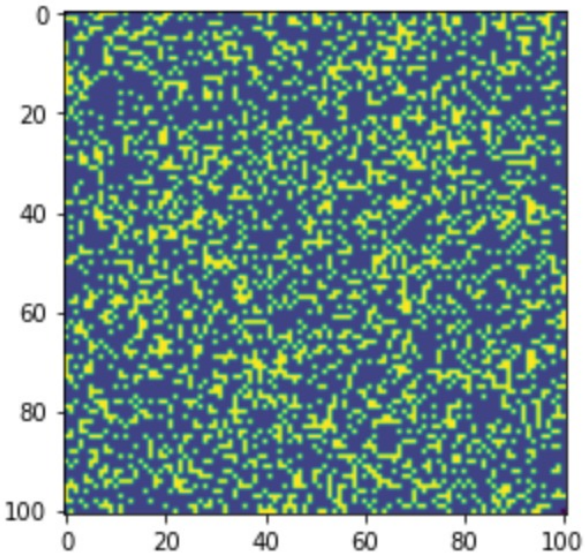


Figure 2: Sample of the Maze

3 Part 1

3.1 A)

In this example the goal of the agent is to travel from the starting point at Cell E3 to the destination at cell E5. In the given algorithm the first move of the agent would be towards the east rather than the north because the first logical step of the A* algorithm would be to proceed towards the shortest unblocked path. The F value towards the east direction which is 3 is lower than that of north which has a value of 5. Hence prompting the agent to traverse towards east. Owing to the initial state environment awareness of the agent, it will know that cell E2 is unblocked and can be traversed.

3.2 B)

We know that the agent is bound to visit the nodes of the maze. The grid world that is generated is finite and each node in the world will be visited

by the agent as long as there is a path between the agent and the node. When the agent has visited all the nodes, it would have found the target and added it to the open list. If the target is unreachable, the open list would be empty and the target will not be visited. The algorithm would stop in this case. In the case that the path to the target is blocked by blocked cells around it, the agent need not visit any other path far from the goal path. This way, the agent will say that it is not possible to find the goal in finite time. The number of moves of the agent until it reaches the target or discovers that it is impossible is bounded by the number of unblocked cells squared. In the worst case, N nodes will expand N number of nodes each time and we will have O maximum moves, where n is the number of unblocked nodes.

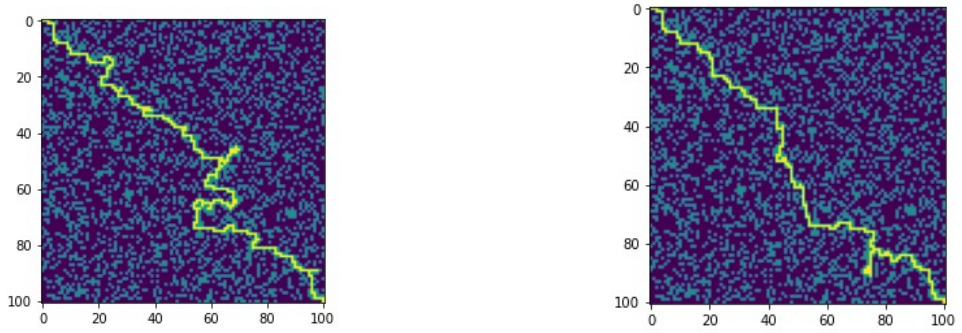
4 Part 2

Using the number of expanded nodes to compare the two pathfinder algorithms, it is evident that the Repeated Forward A* breaking ties with a higher G-value is faster. A G-value g(s) is the length of the shortest path from the start state to the state (S) found by the A* search. The heuristic (h-value) and h(s) (which is user supplied and does not change) estimates the goal distance of the state s and the distance from state s to the goal state (Manhattan distance in this case). An f-value f(s) := g(s) + h(s), estimates the distance from the start state via state S to the goal state. However, in the case of multiple cells that have the same F-value, the algorithm can either break ties with cells that have a smaller G-value or with cells that have a larger G-value. After testing the G-value through our code we found that the algorithm prefers a higher G-value with a smaller time taken. We preferred states closer to the start state than the goal. As the goal is further away from the start state, it would be better to reverse the tie-breaking rule and expand the states with a higher G-cost first. The table below shows the time taken for the G-value for five different mazes.

Greater G value	Cells explored	Smaller G Value	Cells explored
0.17	7828	0.36	6038
2.14	12306	3.45	10955
0.95	9968	1.82	9095
3.4	16619	4.6	9433

5 Part 3

We implemented Repeated Forward A* and Repeated Backward A* with respect to their runtime in second and considering both algorithms breaking ties at higher G values. If we compare both algorithms by their runtime, we observe that Repeated Backward A* is faster. Here, our mean runtime for Forward A* is 0.936, while mean runtime for Backward A* is 1.058.



(a) Repeated Forward A*

(b) Repeated Backward A*

Figure 3: Path taken by A* Algorithm

Forward A*	Cells explored	Backward A*	Cells explored
0.36	8278	0.36	11058
1.44	13306	0.81	7955
0.95	9498	0.82	9095
1.4	15689	0.36	9433

6 Part 4

6.1 A)

Manhattan distances are consistent because they are computed by summing up the shortest possible vertical and horizontal distances. Manhattan distances between two points will always be the same for any horizontal and vertical steps. In our case, agent can not move in diagonal fashion, it only moves in either vertical or horizontal fashion. Thus, this heuristic can never overestimate the cost of reaching the target.

6.2 B)

Here, we will prove this considering a grid world where agent could only move in four directions.

Consider, H-values are consistent. h(s) follows Manhattan Heuristics and $hnew(s) := g(goal) - g(s)$. Also, cost of going from n to $\acute{n}$ will be $c(n, a, \acute{n})$ will be one.

Consider, triangle inequality

$$h(n) \leq h(\acute{n}) + c(n, a, \acute{n}) \tag{1}$$

The fact that this inequality holds for h(s) tells us that those h-values are consistent.

Now, prove that $hnew(n) \leq hnew(\acute{n}) + c(n, a, \acute{n})$.
First, we substitute

$$hnew(n) := g(goal) - g(n) \tag{2}$$

$$hnew(\acute{n}) = g(goal) - g(\acute{n}) \tag{3}$$

We get $g(goal) - g(n) \leq g(goal) - g(\acute{n}) + c(n, a, \acute{n})$
Eliminate g(goal) from both sides,

$$g(n) \geq g(\acute{n}) + c(n, a, \acute{n}) \tag{4}$$

Here, if the agent can move only in the four main compass directions this equation is always true. Here, c(n,a,ń) is one. If g(ń) is smaller than g(n), then subtracting one from it will make it even smaller. Also, g(ń) is greater than g(n), then subtracting one from it will make them equal. Therefore the triangle inequality is satisfied and the h new (s) values are consistent. H-values h new (n) remain consistent even if action costs can increase.
Now, we consider again the triangle inequality: $hnew(n) \leq hnew(\acute{n}) + c(n, a, \acute{n})$. Let's assume that there is an action cost increase, where c: action cost before increase $\acute{c}$: action cost after increase.
Thus: $hnew(n) \leq hnew(\acute{n}) + c(n, a, \acute{n}) \leq hnew(\acute{n}) + \acute{c}(n, a, \acute{n})$.
Therefore the heuristic remains consistent.

7 Part 5

Assume that we are running several A* searches with consistent heuristics in the same state space and with the same goal states but possibly different start states. Adaptive A* would always be better than Repeated Forward A*. In Adaptive A* the idea is to make the heuristics more informed after each A* search in order to speed up future A* searches. Adaptive A* uses informed h-values to focus its searches. The initial h-values are provided

by the user and must be consistent for the initial action costs. Adaptive A* updates its h-values after each search to make them more informed and focus its searches even better. An iteration of Adaptive A* proceeds as follows: It first updates the action costs, if necessary, to reflect any increases in action costs. It then picks a start state and runs a forward A* search to find a cost-minimal path from the start state to any state in the given set of goal states. Assume that the search determined that the cost of the cost-minimal path is g*. The h-values remain consistent and Adaptive A* thus continues to find cost-minimal paths over time without having to re-expand states during the same search.

8 Part 6

Question : Performance differences between two search algorithms can be systematic in nature or only due to sampling noise (= bias exhibited by the selected test cases since the number of test cases is always limited). One can use statistical hypothesis tests to determine whether they are systematic in nature. Read up on statistical hypothesis tests (for example, in Cohen, Empirical Methods for Artificial Intelligence, MIT Press, 1995) and then describe for one of the experimental questions above exactly how a statistical hypothesis test could be performed. You do not need to implement anything for this question, you should only precisely describe how such a test could be performed

The following is a description of how statistical significance can be used when comparing Repeated Forward A* and Repeated Backward A*:

1.Null Hypothesis Description/Statement:
Observing the runtime between Repeated Forward A* and Repeated Backward A* there seems to be no difference.

2.Alternate Hypothesis Description/Statement:
After observation, it was noticed that there was a significant run time difference in the performance of Repeated Forward A* and Repeated Backward A*.

3.Setting the significance level:

Alpha = 0.05

4.Run Repeated Forward A* and Repeated Backward A* on the same Maze and collect the runtimes
Repeat the process for 50 more mazes

5.Calculate a test statistic and p-value
Calculate mean Runtime for Repeated Forward A* over the 50 mazes
Calculate the mean runtime for Repeated Backward A* over 50 mazes

Calculate the p-value using this test statistic