Fast Trajectory Replanning
CS520 : Intro To AI - (Assignment 1)

Sai Mounica Pothuru (RUID: 198008261)
Chaitanya Sharma Domudala (RUID: 198009015)

October 12 2020

Part 0 - Setup your Environments

Architecture

Two mazes are constructed as a 2d array of size 101X101. The first maze is the true maze with all the obstacles and the second maze is used as knowledge maze for the A*. The mazes will contain 1's to indicate the blockages and 0's to indicate they are free. Values 3 and 4 indicate the Agent and Target respectively.

Load

The mazes are set up such that, all the cells in the mazes are initialized with values as 0. Two sets of x and y coordinates are generated randomly using randint() function of numpy, one set for Agent and Target each.These coordinates are used to load the true maze and knowledge maze. To assign blocks to the truemaze, nditer() of numpy is used to iterate through values and set a block if the random probability is more than 0.3. Cells with random probability less than that will remain 0 as free passage.

Visualize

For visualization, matplotlib colors are used to establish different colors. Black indicates blockage, White is for Free cells, Green shows Agent, Red gives Target, Yellow expresses the Path. The maze is plotted using imshow() function with colors.

Part 1 - Understanding the methods

a) Explain in your report why the first move of the agent for the example search problem from Figure 1 is to the east rather than the north given that the agent does not know initially which cells are blocked.

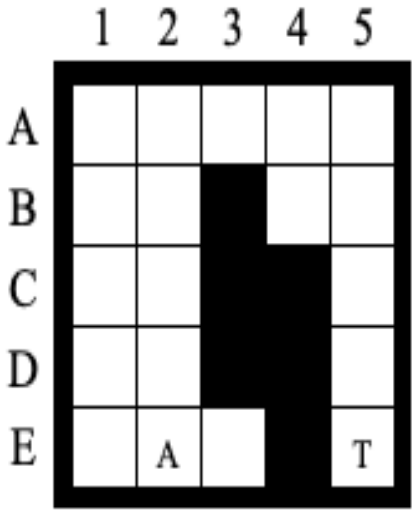

Figure 1: Example Search Problem

Let A be the Agent position and T be the target position. The agent does not know which cells are blocked initially, and we use Manhattan distance as heuristics. When we perform A* from A to T, we first push A to the open list. Then we expand A and add its three neighbors – cell E3, cell D2, and cell E1 to the open list.

Now the open list contains 3 items:

cell E3: g-value:1 h-value: 2 f-value: 3

cell D2: g-value:1 h-value: 4 f-value: 5

cell E1: g-value:1 h-value: 4 f-value: 5

Since cell E3 has the smallest f-value, so cell E3 will be chosen to expand. We will then push cell E3's neighbors – cell E4 and cell D3 to the open list. We do not push cell E2(A - Start point) to the open list because cell E2 is already expanded and is in the closed list.

Now the open list contains 4 items:

cell E4: g-value:2 h-value: 1 f-value: 3

cell D3: g-value:2 h-value: 3 f-value: 5

cell D2: g-value:1 h-value: 4 f-value: 5

cell E1: g-value:1 h-value: 4 f-value: 5

Since cell E4 has the smallest f-value and the agent does not know whether cell E4 is blocked at this point or not, so cell E4 will be chosen to expand. We will then push cell E4's neighbors – cell E5(T) and cell D4 to the open list. We do not push cell E3 to the open list because cell E3 is already expanded and is in the closed list.

Now the open list contains 5 items:

E5: g-value:3 h-value: 0 f-value: 3

D4: g-value:3 h-value: 2 f-value: 5

D3: g-value:2 h-value: 3 f-value: 5

D2: g-value:1 h-value: 4 f-value: 5

E1: g-value:1 h-value: 4 f-value: 5

Since cell cell E5(T) has the smallest f-value, so cell E5 will be chosen to expand. since E5 is the goal point, so A* will terminate at this point.

We found the path: E2(A) - E3 - E4 - E5(T) Therefore, the agent will move to east at the beginning

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 grid worlds 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.

In finite grid worlds, we generally have a finite number of cells. In addition, each cell is connected to its neighboring cells, making it feasible to move from one cell to other in the grid world by following the neighbor pointers assuming there are no blocked cells. Similarly, if we take the blocked cells into account, we can go from one unblocked cell to any other unblocked cells within the same unblocked neighborhood. By unblocked neighborhood, it means an area of consecutive unblocked cells i.e.., neighborhood of cells that are surrounded by either the boundaries of the grid world, or the blocked cells, or both.

We maintain a closed List in the A* algorithm, so that it will never expand the cells already explored. As mentioned above,in finite grid world, A* will either find a path from start to target

if both are in same unblocked region, or it will report no path is available after traversing through whole unblocked region & reports it as impossible in finite time.

The Repeated Forward/Backward Algorithm will execute A* algorithm multiple times. For each A*, it will either find a path from start to target under current knowledge of the grid world, or report there is no path. If it reports no path, it means that indeed there is no such path from start to target given the blocked cells. If it finds a path from start to target, then the agent will follow this path to either reach the target, check for blockages and repeat the process again if blocks are found. Since we have a finite number of cells, so the worst case would be the agent moves through all these cells of the grid. Since the agent moves a finite number of cells and each time it hits a blocked cell, it will run A* again which costs a finite time. So the Repeated Forward/Backward algorithm will either reach the target or report as this is impossible in finite time.

Let us denote the number of unblocked cells as n.
The process of Repeated Forward/Backward A* can be written as the following:

Loop:
computePath()
moveAgent()

Since each A* keeps a closed List, the path it finds will contain at most n cells. The number of moves m, after each A* can be at most n. That is,

$$m \leq n$$

we denote the number of iterations of the loop as l. For the number of iterations of the loop, the worst case would be every time the agent follows the presumed path moves one step, the next step is blocked and needs to run A* from current location again and all cells within current unblocked region has to be traversed at least once, in addition, the worst case also requires that there is only one unblocked region. All other cells are blocked, that is we have n cells under the same unblocked region. Since every time the agent moves to a new cell, it will explore the neighboring cells, so we can run A* on each cell at most once. Say we visited cell A once, when the future A* try to find a path, it will take A's blocked neighbors into account so we will never get stuck in A again. Therefore, the number of A* searches, which is the number of iterations of the loop l, can be at most n. That is,

$$l \leq n$$

Therefore, from the above two inequalities, we can conclude that

$$lm \leq (n^2)$$

Where,
n is number of unblocked cells
l is number of loop iterations
m is number of agent moves within one loop iteration
lm is total number of agent moves

Therefore, 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.

Part 2 - The Effects of Ties

The below plot depicts the run time for Repeated Forward A* algorithm with both tie break in favor of larger g-value and smaller g-value.We performed 50 experiments on these 2 tie break strategy with the same 101*101 grid world:

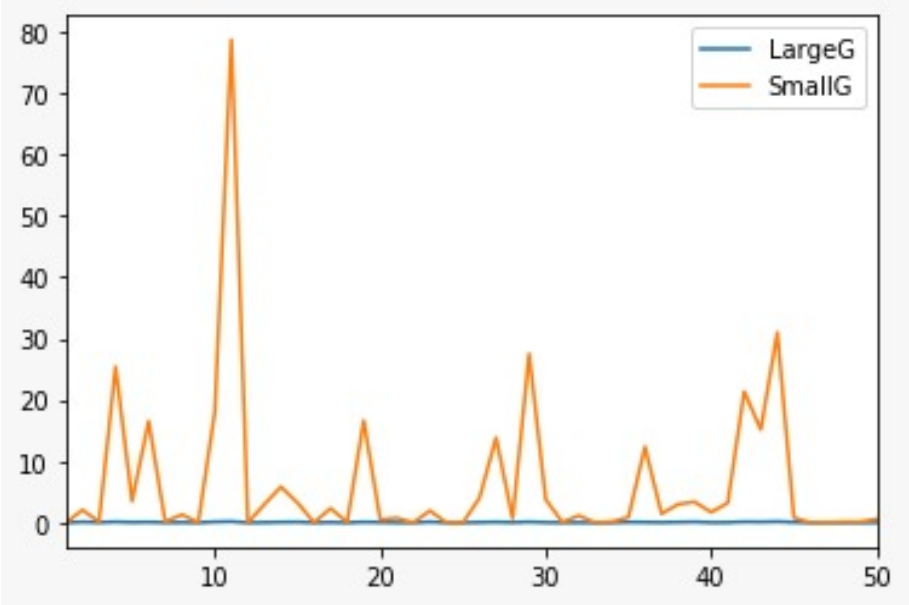

Figure 2: Tie Break Experiment

We found that Repeated Forward A* with larger g-value tie break strategy is much faster than Repeated Forward A* with smaller g-value tie break strategy in general despite some occasional cases such as fail to found path at the very beginning.

The reason of this is because under the situation of multiple cells having the same f-value, if we choose cells with smaller g-value, this will take us back to cells near the agent position whereas choosing cells with larger g-value will let us explore cells near the target position. If we choose to expand cells near the agent position, we will expand a lot of useless cells.

we can consider a problem where we want to search a path from A to T within the environment on figure 3:

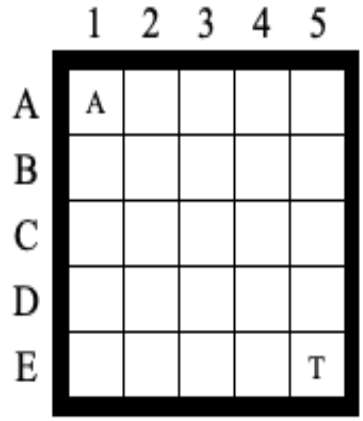

Figure 3: Third Example Search Problem

The first cell to be expanded, is cell A1, which is the start state. After expanding cell A1, the open list contains 2 cells:

cell A2: g-value:1 h-value: 7 f-value: 8

cell B1: g-value:1 h-value: 7 f-value: 8

Since cell A2 and cell B1 have the same f-value and g-value, so we can randomly choose one to expand, in this case,let it be cell B1. After expanding cell B1, the open list now contains:

cell A2: g-value:1 h-value: 7 f-value: 8

cell B2: g-value:2 h-value: 6 f-value: 8

cell C1: g-value:2 h-value: 6 f-value: 8

Now, since all cells in open list have same f values, we choose cell A2 as it got smaller g-value,if we choose to expand cell A2 after expanding it the open list contains:

cell B2: g-value:2 h-value: 6 f-value: 8

cell C1: g-value:2 h-value: 6 f-value: 8

cell A3: g-value:2 h-value: 6 f-value: 8

Now, since all cells in the open list have the same f-value and g-value, so we can randomly choose one to expand, in this case, cell C1.After expanding C1, the open list contains:

cell B2: g-value:2 h-value: 6 f-value: 8

cell A3: g-value:2 h-value: 6 f-value: 8

cell C2: g-value:3 h-value: 5 f-value: 8

cell D1: g-value:3 h-value: 5 f-value: 8

Now, all cells in the open list have the same f-value, if we choose to expand cell with smaller g-value, we will choose one from cell B2 or cell A3 to expand.

We can find the pattern from the above steps that under the situation of all cells with the same f-value, if we break tie in favor of smaller g-values, we will end up expanding a lot of useless cells near the agent position, in this particular case, we expand all diagonals. Where as if we break tie in favor of larger g-values, we will always expand cells near the target position.

Therefore, Repeated Forward A* with larger g-value tie break strategy is much faster than Repeated Forward A* with smaller g-value tie break strategy in general as Repeated Forward A* with smaller g-value tie break strategy will waste a lot of time on expanding useless cells near the agent position

Part 3 - Forward vs. Backward

We ran the forward A* algorithm and backward A* algorithm by interchanging the agent and target nodes. The algorithm had been run for 50 iterations of maze size 101X101.The observations is as per the figure shown below.

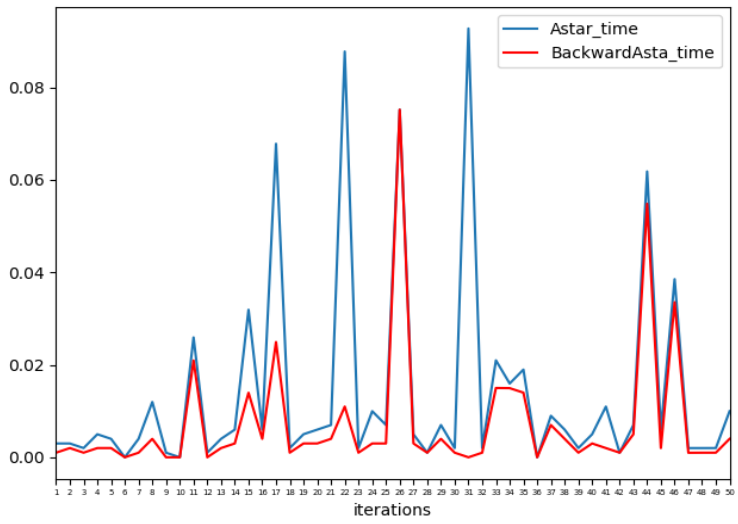

Figure 4: Forward A* Vs Backward A*

We observed that the time difference is due to the obstacles. In case if the obstacles are concentrated near the agent neighborhood, then Forward A* took lesser time as it immediately finds the obstacles and finds a better path while backward A* takes a longer time. In case if the obstacles are sparse near the agent neighborhood, then Forward A* takes a longer time when compared to backward A*. This can be justified by the number of nodes expanded as shown in the below figure. The larger the number of nodes that should be expanded the larger the time taken to reach the goal.

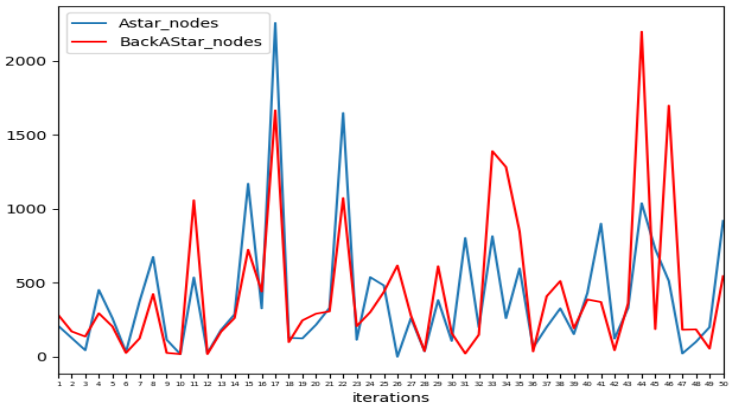

Figure 5: Forward A* Vs Backward A*

Part 4 - Heuristics in the Adaptive A*

The project argues that "the Manhattan distances are consistent in grid worlds in which the agent can move only in the four main compass directions." Prove that this is indeed the case.

Since in the grid world, agent can move only in four directions i.e..,(up, down, left and right). Manhattan distance is the distance where we let agent move vertically until it is on the same horizontal line with the goal, and then moves horizontally to the goal. This is the fastest possible path in grid world and it requires that there are no blocked cells on its path. If we have blocked cells on its path, we would have to detour a few cells and it will increase the cost. If we do not follow the path which has cost equal to Manhattan distance, if we go up 'n' number of cells, we would eventually have to go down the same number of cells in order to get back to the goal. And this applies also to go down, go left, and go right.Since the agent cannot move diagonally, we can confirm that shortest path will always be found, if not then the heuristic would overestimate the distance.

Therefore, Manhattan Distance is indeed a consistent heuristics in grid worlds.

Furthermore, it is argued that "The h-values hnew(s) ... are not only admissible but also consistent." Prove that Adaptive A* leaves initially consistent h-values consistent even if action costs can increase.

Since g(goal) is the smallest goal-cost recorded by A* search under current knowledge of the grid world, and g(s) is the smallest cost to s detailed by A* search under current knowledge, so g(goal) - g(s) is also the smallest cost from s to goal under current knowledge of the grid world. Since as

we explore the world, the number of unblocked cells can only increase, that is the cost to goal can only increase as well, so the new heuristics used by Adaptive A* is the smallest cost from s to goal under current knowledge of the grid world.

Part 5 - Heuristics in the Adaptive A*

We performed 50 iterations on both Repeated Forward A* and Adaptive A* on 101*101 grid world by randomizing the agent and target nodes and got the following result for their run times:

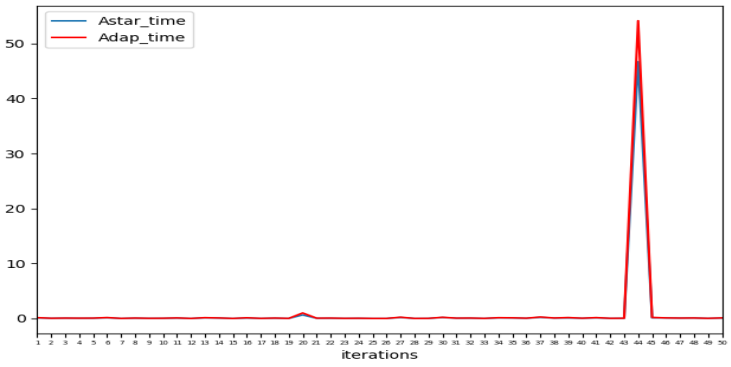

Figure 6: Forward A* Vs Adaptive A*

We found that Repeated Forward A* with Adaptive A* and Repeated Forward A* with A* took almost similar time when running time is taken into account. But when we consider the number of nodes expanded, Adaptive A* algorithm expanded much lesser nodes when compared to A* algorithm. The below figure shows the number of nodes expanded in each case for every iteration.

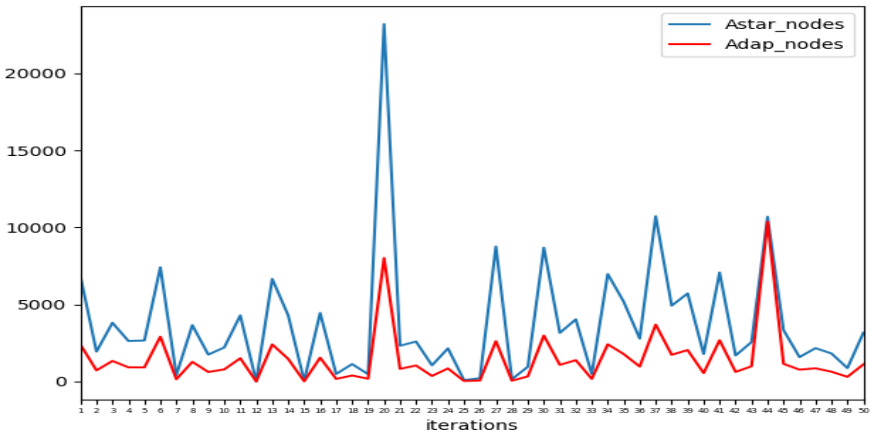

Figure 7: Forward A* Vs Adaptive A* Expanded Nodes

The reason is, as Adaptive A* updates the h-value of expanded cells to a higher valued consistent heuristic, which is more accurate, after each planned path. So when next time Adaptive A* take these cells to the open list, it could make a better choice than the typical A* does. So the number of cells expanded by each Adaptive will be less than number of cells expanded by usual A*, in general.

Part 6 - Statistical Significance

Let us determine the statistical significance of Repeated Forward A* and Adaptive A* algorithms. We can compare the performance of both methods based on total number of expanded cells in the grid worlds.

We executed each search algorithm on 50 randomly generated grid worlds of size 101X101 with randomised agent target positions and obtained two sample data sets. Various types of hypothesis tests such as t-est, z-test, ANOVA can be implemented on these data sets.

Let's consider a t-test make use of two sample t-test to validate the hypothesis. For Evaluation purpose, one tailed t-test is used to accept/reject the hypothesis.

Let AA* be Adaptive A* test sample RA* be Repeated Forward A* test sample

H0: AA* = RA*

(Null Hypothesis) - Both the algorithms perform equally well

H1: AA > RA

(Alternative Hypothesis) - Adaptive A* performs better compared to Repeated Forward A*

$$T_{stat} = \frac{AA^* - RA^*}{\sigma_{AA^*-RA^*}}$$

$$\sigma_{AA^*-RA^*} = \sqrt{\frac{(n_{AA^*}-1)s^2_{AA^*} + (n_{RA^*}-1)s^2_{RA^*}}{n_{AA^*}+n_{RA^*}-2}(\frac{1}{n_{AA^*}} + \frac{1}{n_{RA^*}})}$$

$$df = n_{AA^*} + n_{RA^*} - 2$$

- $\sigma_{AA*-RA*}$: $Standarderrorofdifferencebetween AA* and RA*$
- s_{AA*} : $StandarderrorofAdaptiveA*$
- s_{RA*} : $StandarderrorofRepeatedForwardA*$
- n_{AA*} : $No.ofsamplesinAdaptiveA*algorithm$
- n_{RA*} : $No.ofsamplesinRepeatedForwardA*algorithm$
- df : Degrees of freedom

A confidence interval should be chosen to perform the test. Using this, significance of the test is calculated. Based on the type of the t-test(In this case, it's a 1-tailed T-test as mentioned above), critical values are calculated from the T-test statistic table.

For example, test is conducted at 95% confidence. Critical values are calculated based on the confidence level and degrees of freedom. If the t-stat is greater than critical value, Null Hypothesis is rejected. Thus it can be confirmed that the alternative hypothesis is accepted with 95% confidence. It means that Adaptive A* will be outperforming Repeated Forward A*.

References

[1] Numpy Functions https://numpy.org/doc/stable/.

[2] Maze generation. *https://rosettacode.org/wiki/Maze_generation*.

[3] Xiaoxun Sun,Sven Koenig, William Yeoh,
`http://idm-lab.org/bib/abstracts/papers/aamas08b.pdf`