Assignment 1

Fast Trajectory Replanning

Part 0 - Environment setup

To generate mazes of size 101x101, we worked on two algorithms. We wrote the first algorithm, which finds a random path from a starting point to a target point and then randomly assigns the remaining spaces in the grid as an obstacle or free space based on an input probability.

Once we generated multiple mazes of varying sizes, we observed that in many cases, the resultant mazes could be solved easily just by observing the maze. This was the case for most grids generated using this algorithm.

We then tried to look for algorithms that would help us generate more complex mazes. To generate the mazes for this assignment, we used a random acyclic maze generator with a given Entry and Exit point [1].

This algorithm ensures no cycles in the path while keeping track of how a node is discovered. We first check if any of the neighbors to the current node is a target. If so, we add it to the top of the stack; otherwise, we just shuffle the neighbors and add them to the stack, ensuring that the generated maze is random.

In the GUI we have prepared for the project, the agent's movement is represented using a blue square. The target is yellow, and the path is highlighted in green and is displayed after it is discovered. Red cells are the blockers where the agent cannot go.

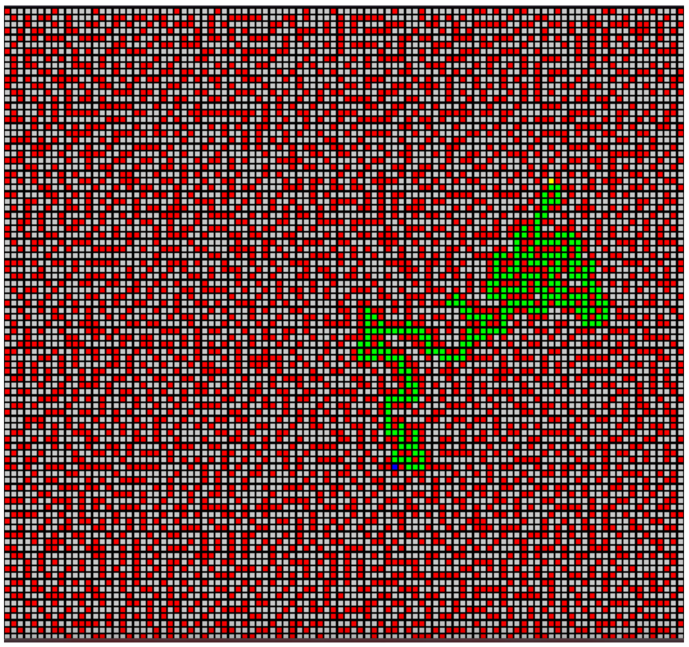


Fig 1: Output of the algorithm once Repeated A is finished.*

Part 1 - Understanding the methods [10 points]:

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.

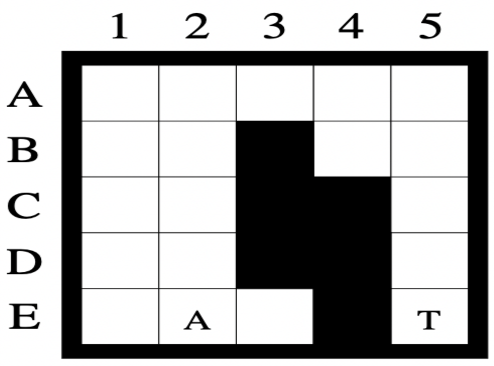


Figure 8: Second Example Search Problem

Considering Figure 8 as a 5x5 grid, the agent is at [5,2], and the target is at [5,5].
A* uses manhattan distance as the heuristic.

It chooses to move east as we get the distance between [5,2] and [5,5] as h = 3, and the cost g = 3. It also doesn't have any obstacles in the cell on its right.

If it chooses to move north, the g and h values will be higher, implying a higher f value. So the preference is for the lowest f-value path.

Moving towards the east has a lesser h value when compared to the north. Hence it moves towards the east.

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.

For any n x n maze, let b be the number of unblocked cells.

The worst case scenario for Repeated A* would have the following:

1. Agent will not be able to move more than 1 step after getting the path from A*. It will encounter an obstacle everytime it takes one step further.
2. It will not be able to reach the goal. However, it will not find this until the agent reaches closer to the goal. Which means the Repeated A* runs for the maximum number of iterations but finds that the goal is unreachable just before it would have reached the goal state.

In any iteration of A*, the open list would have at most b nodes/blocks.

That means that the maximum length of the path returned by A* is b. For the worst-case scenario, the agent would call A* after each step, and the longest possible path, in this case, would be if it needs to go through each unblocked cell.

Also, in the worst-case scenario, repeated A* would have to backtrack to previous points/blocks. The algorithm would backtrack its movement until and unless A* found no path. In the worst case, it would have to go to each unblocked position b times.

Therefore the maximum number of moves for the agent is always less than or equal to the square of b, which is the number of unblocked cells.

Part 2 - The Effects of Ties [15 points]:

Breaking ties in favor of smaller g-values:

{image of code explaining what we did to consider the lower g-values}

Breaking ties in favour of larger g-values:

{image of code explaining what we did to consider the lower g-values}

	Runtime	**Number of expanded cells**
Higher g-values	0.18	957.9
Lower g-values	0.24	981.84

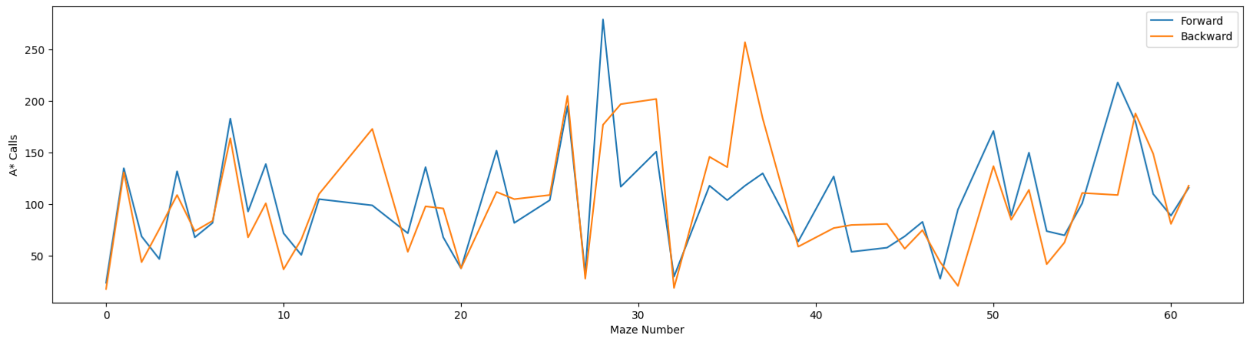
Graph for runtime and number of expanded cells where the g-value is smaller

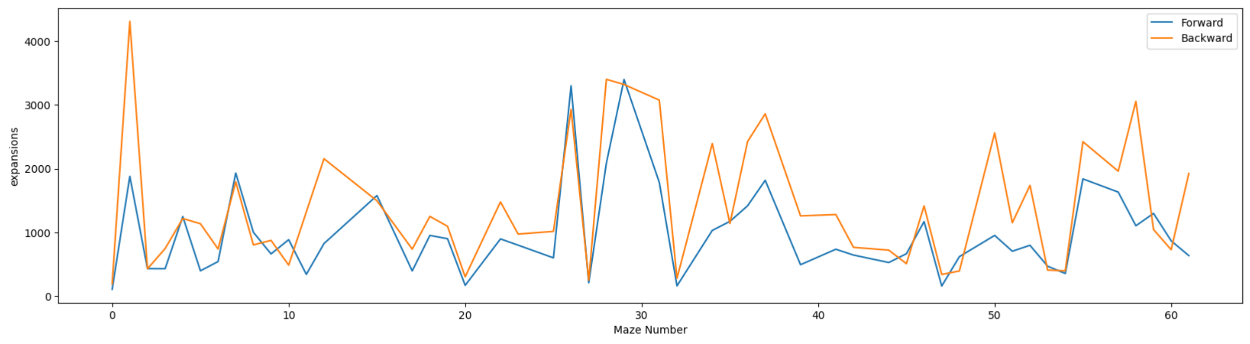
Graph for runtime and number of expanded cells where the g-value is higher

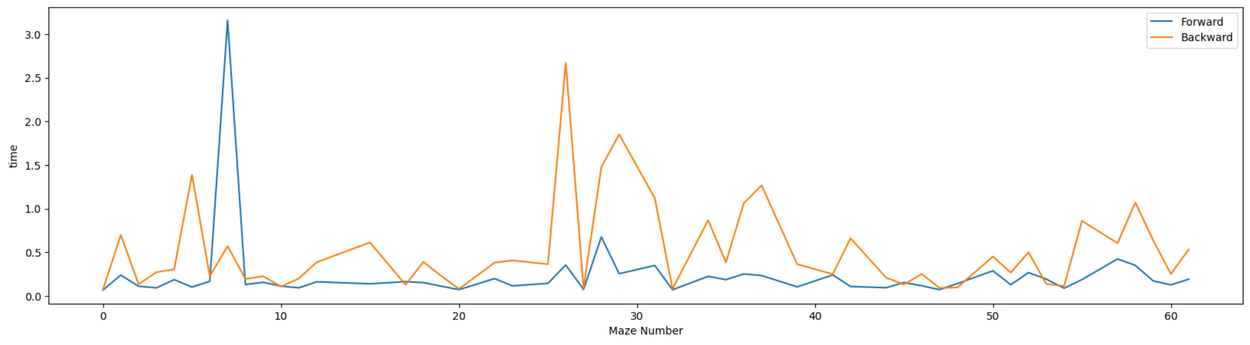
Observation from the numerals and the graph above:

Part 3 - Forward vs. Backward [20 points]:

Below are the images comparing Forward A* and Backward A* w.r.t A* calls, number of cells expanded and runtime. We notice that there is no specific pattern for any parameter and it depends only on the number of blockers in the maze and the position of the source and the target.







	Runtime	Number of expanded cells	Number of times A* calls
Forward A*	0.24	981.84	103.44
Backward A*	0.51	1415.28	102.16

Image of the number of times A* is called for backward and forward

Part 4 - Heuristics in the Adaptive A* [20 points]:

A heuristic function h is said to be consistent if

$\forall (n, a, n0) : h(n) \leq c(n, a, n0) + h(n0)$.

We need to prove that if we consider Manhattan distance as a heuristic is consistent.

Let us assume that Source is at (i,j) and the target is at (x,y).

Manhattan distance between these two points is $|x-i| + |y-j|$ which is the actual heuristic value.

Let us have the cost value after applying an action to move from n to n0 is always 1. With that we have c(n, a, n0) as 1 for any a.

Therefore, we need to prove that $h(n) \leq c(n, a, n0) + h(n0)$.

h(n0) - is the heuristic of the new position we get when we do an action from h(n)

There are four possibilities of h(n0) from the initial step. We calculate the updated heuristic w.r.t the standard manhattan distance $|x-i| + |y-j|$

Case 1:

The agent moves one step above the current position, where (i,j) is changed to (i,j+1). Replacing it on the equation we get:

$h(n) \leq c(n, a, n0) + h(n0)$

$|x-i| + |y-j| \leq 1 + |x-i| + |y-(j+1)|$

$|x-i| + |y-j| \leq 1 + |x-i| + |y-j-1|$

This holds the condition as c(n, a, n0) + h(n0) will be equal to h(n).

Case 2:

The agent moves one step below the current position, where (i,j) is changed to (i,j-1). Replacing it on the equation we get:

$h(n) \leq c(n, a, n0) + h(n0)$

$|x-i| + |y-j| \leq 1 + |x-i| + |y-(j-1)|$

$|x-i| + |y-j| \leq 1 + |x-i| + |y-j+1|$

This holds the condition as c(n, a, n0) + h(n0) has value higher by 2 when compared to the h(n)

Case 3:

The agent moves one step right to the current position, where (i,j) is changed to (i+1,j). Replacing it on the equation we get:

$h(n) \leq c(n, a, n0) + h(n0)$

$|x-i| + |y-j| \leq 1 + |x-(i+1)| + |y-j|$

$|x-i| + |y-j| \leq 1 + |x-i-1| + |y-j|$

This holds the condition as c(n, a, n0) + h(n0) will be equal to h(n).

Case 4:

The agent moves one step left to the current position, where (i,j) is changed to (i-1,j). Replacing it on the equation we get:

$h(n) \leq c(n, a, n0) + h(n0)$

$|x-i| + |y-j| \leq 1 + |x-(i-1)| + |y-j|$

$|x-i| + |y-j| \leq 1 + |x-i+1| + |y-j|$

This holds the condition as c(n, a, n0) + h(n0) has value higher by 2 when compared to the h(n).

4.(b)

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

A heuristic function h is said to be consistent if $\forall$(n, a, n0) : $h(n) \leq c(n, a, n0) + h(n\ 0)$. where c(n, a, n0) is the step cost for going from n to n 0 using action a.

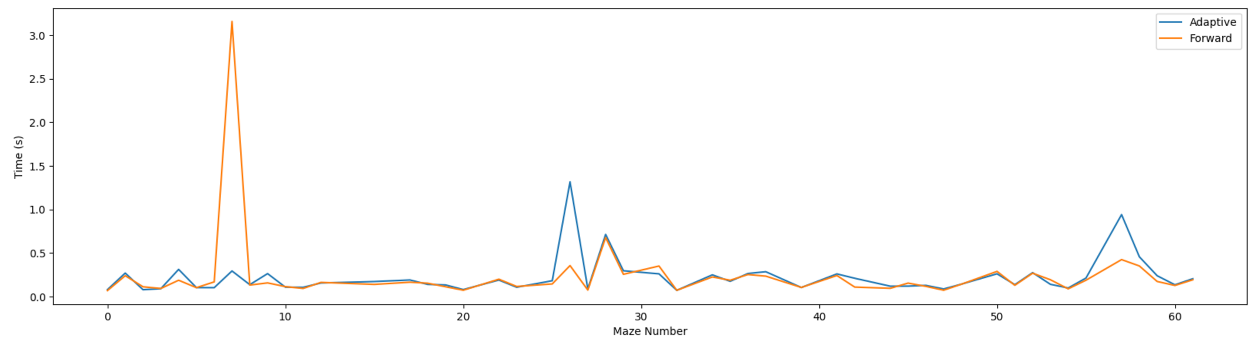
In an ideal case, following the optimal path, the heuristic values should decrease as we move one step away from the source and closer to the target. This way the value of h(succ(s,a)) iteratively goes to zero as we reach the target. And the h(s - goal) is zero. By this we can also say that the heuristic value will be 0 only for the goal state and not for any other non-goal state for any action a.

Considering the above statement, the h-value of the successor can either be equal to the previous one which means equal, greater than the previous value and decreased when compared to the previous value.

Let us assume that the starting point is S(i,j) and the ending point is S(x,y)

Part 5 - Heuristics in the Adaptive A* [15 points]:

	Runtime
Repeated Forward A*	0.24
Adaptive A*	0.22



Part 6 - Statistical Significance [10 points]:

REFERENCES:

Maze Generation:
https://www.geeksforgeeks.org/random-acyclic-maze-generator-with-given-entry-and-exit-point/

Explanation:

The Adaptive A star algorithm takes more time for certain cases. But this time difference is very very small and hence it is not statistically significant.

Outliers dominate the means, so test isn’t significant

Useful Terms

Frequency distribution: The frequencies with which the values in a distribution occur (e.g., the frequencies of all the values of "age" in the room)

Outlier: Extreme, low-frequency values.

Mean: The average.

Means are very sensitive to Outliers

We should compare mean and median values. Mean can be higher for one algorithm and its median can be lower since means are highly susceptible to outliers

Median: The value which splits a sorted distribution in half. The 50th quantile of the distribution.

Since the H new values of Adaptive A star are consistent, then this eqn $h(s) \leq c(s, a) + h(succ(s, a))$ for a state and all actions a that can be executed in state is satisfied. So the new h values in adaptive a star are consistent when the action costs are constant.

f(s) <=

The new h values of all expanded states s are not smaller than the immediately preceding h-values h(s) and thus, by induction, also all previous h-values, including the user-supplied h-values. So, the h values of