Introduction to Artificial Intelligence

Project 1 Report

Amany Elgarf

Bharti Mehta

Part 0 - Setup Your Environments

The maze is generated by a DFS through the maze and traversing each cells neighbor randomly. Before every cell is entered into the open list, we generate a random integer from 1-100. If the integer is greater than 70, the cell is marked as block by making its cost infinity, otherwise the cost is kept as 1.

Part 1 - Understanding the Methods

A* algorithms expand nodes with the lowest f-values first. Based on this expansion, whichever path reaches the goal first, is returned by the algorithm. F-values are the sum of g-values (cost from start to current state) and h-values (in this case, Manhattan Distances). In this example, A (E2) would be popped out of the open list and we traverse its neighbors E1, E3, and D2. They are inserted into the open list with their respective g-values, h-values, and f-values:

Cell	Parent	G-value	H-value	F-value
E3	E2	1	2	3
D2	E2	1	4	5
E1	E2	1	4	5

Since E3 has the lowest f-value, it will be expanded first, resulting in the following open list:

Cell	Parent	G-value	H-value	F-value
E4	E3	2	1	3
D3	E3	2	3	5
D2	E2	1	4	5

E1	E2	1	4	5

Note we do not insert previously expanded nodes into the open list. E4 will be popped out next and expanded to give the following open list:

Cell	**Parent**	**G-value**	**H-value**	**F-value**
E5	E4	3	0	3
D4	E4	3	2	5
D3	E3	2	3	5
D2	E2	1	4	5
E1	E2	1	4	5

At this point the goal, E5, will be popped out, and the path would be determined as: E2 (A), E3, E4, E5. The path returned by A* will, therefore, move east first because these cells have smaller f-values.

There are two main reasons why exploring the gridworld takes a finite amount of time:

The gridworld has a finite number of cells

Each cell is only expanded once in an A* search to avoid falling into infinite loops.

The agent traverses neighbors in search of the goal and adds these cells to the open list. If the goal is popped from the open list, the search successfully ends and returns a path from the start to the goal. Otherwise, the search would end when each cell that is reachable from the start has been expanded and there are no more cells to expand. This means that the goal has not been popped, and therefore cannot be reached from the start. Since the gridworld finite, and each cell is one expanded once, we would find a path in a finite amount of time.

To find the upper bound number of moves for Repeated A* search, let us use the formula:

$$\#\,moves \;=\; \#\,of\ A^{*}\ plannings \;*\; \frac{\#moves}{planning}$$

In the case where the goal is reachable from the start:

The worst case would occur when all the unblocked cells are reachable from the start. For each planning, the same cells cannot be expanded twice, therefore cannot be included in the planned path more than once. Hence, the maximum number of moves would require the agent to step through all unblocked cells once to reach the goal, so assuming $n \;=\; number\ of\ unblocked\ cells$, $\frac{\#moves}{planning} \;\leq n$.

The number of A* plannings depends how far the agent traveled in each A* execution before reaching a blocked cell. Additionally, with each new start the agent has knowledge of it's immediate blocked neighbors. With these two facts in mind, in the worst case, the agent makes only one successful move in the path execution before discovering a blocked cell. This would result in a new A* planning with a new start. In the worst case, each planning would start with the very next unblocked cell, therefore iterating through each unblocked cell (n) to reach the goal, so $\#\,of\ A^{*}\ plannings \;\leq n$.

To get the upper bound number of moves for Repeated A*, we multiply the maximum number of A* plannings by the number of moves per planning:

$$\#\,moves \;=\; \#\,of\ A^{*}\ plannings \;*\; \frac{\#moves}{planning}$$

$$\#\,moves \;\leq\; n \;*\; n$$

$$\#\,moves \;\leq\; n^{2}$$

In the case where the goal is not reachable from the start:

The worst case would occur under the same conditions as the last case, however instead of no unreachable unblocked cells, there exists only one unblocked cell unreachable from the start, which is the goal cell. With everything else the same, the maximum number of moves per planning is still n. The maximum number of A* plannings is still n, therefore the upper bound moves is also n^{2}

Part 2 - Breaking Ties

We implemented Repeated Forward A* algorithm with both tie breaking in favor of larger g-values and smaller g-values. We ran the two versions of the algorithm on 50 mazes of size 101*101 and recorded the number of expanded nodes in figure 1.

From the data we gathered, we found that forward A* in favor of larger g-values tends to be faster than A* in favor of smaller g-value. The average number of expanded nodes for forward

A* in favor of larger g value is 35,733 while the average for forward A* in favor of smaller g values is 43,337.

A cell's f-value is the sum of g-value (distance from start to current cell) and h-value (optimal distance from current to goal cell). As a result, two cells with same f-value, the one with the larger g has a smaller h value, which tells us that the agent's estimated path to the goal is shorter. The cell with the smaller g value, on the other hand, indicates a large h value which tells that the agent is closer to the start and further from the goal. If we choose the one with the smaller-g value we do not move closer to the goal as quickly, and will expand many unnecessary cells.

However, in some special cases A* in favor of a smaller g is faster, for example: a case failing to find the path at the very beginning.

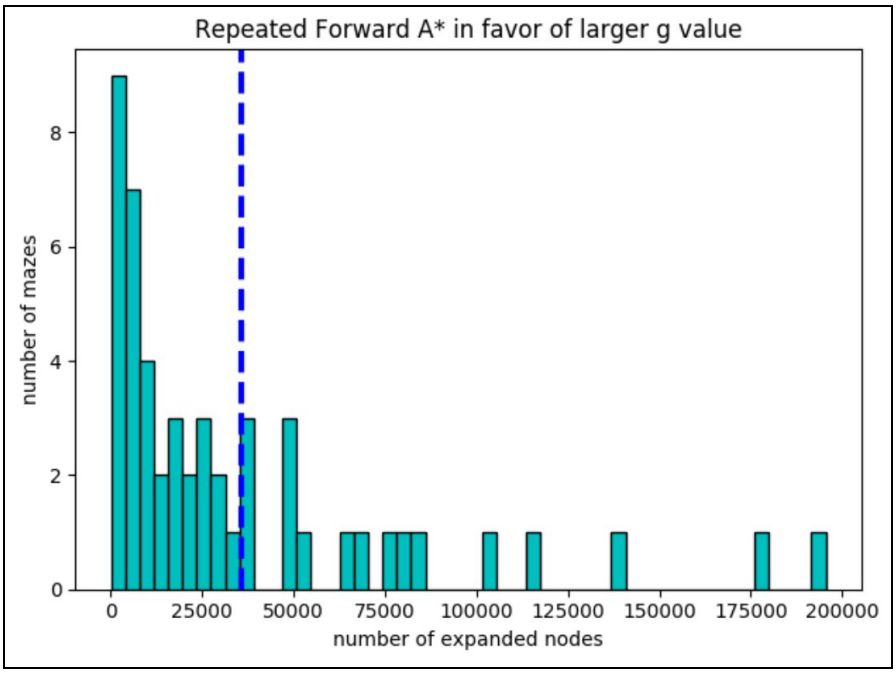

Figure 1

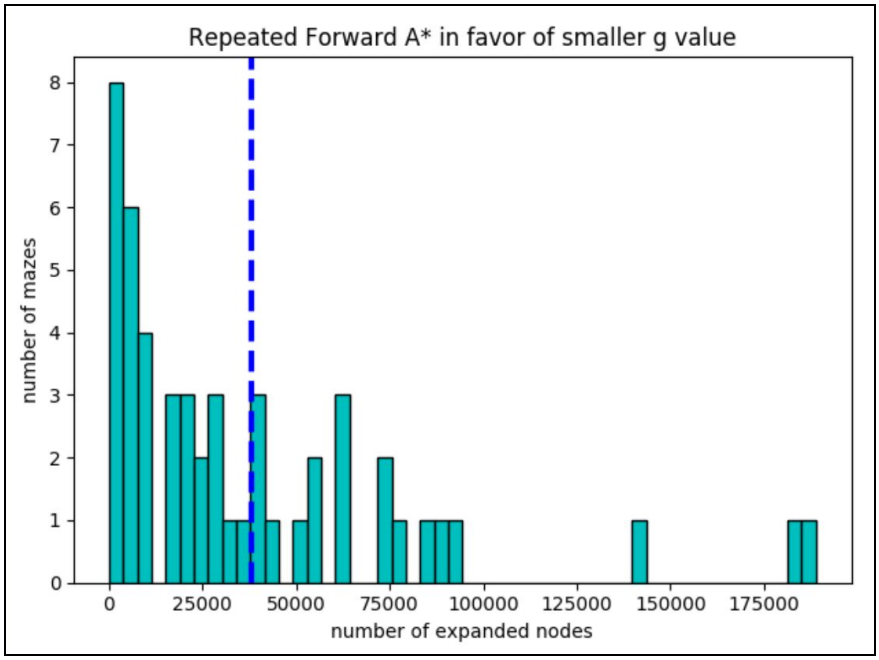

Figure 2

Part 3 - Backward A* vs Forward A*

We implemented both repeated forward and repeated backward versions of the A* algorithm. We ran the two versions of the algorithm on the same 50 mazes of size 101*101 and recorded the number of expanded nodes in figure 4. The average number of expanded nodes for repeated forward was 35,733 while for repeated backward was 49,797.

From the data we gathered, we found that repeated forward A* is faster than repeated backward A* except in some cases such as failing to find the path at the very beginning.

The special case where backward is faster occurs when there is most of the blockage cells are around the goal. Since the search starts from the goal, backward A* will be able to find the result more quickly than forward A*.

Cases where backward is faster:

Case(1): In this case, repeated backward will figure out that there is no path available from the first search, while it will take repeated forward three iterations to be able to reach this conclusion.

G	x	
x	x	
		S

Case(2):

Repeated backward: Repeated Backward A* finds the path from the first time.

G	x	x
	x	x
		S

G	x	x
|	x	x
|		
|	-	S

Repeated forward: Repeated Forward A*, however, has two possible options, one of which would take two iterations before finding the path.

G	x	x
	x	x
		|
		S

G	x	x
|	x	x
-	-	-
		S

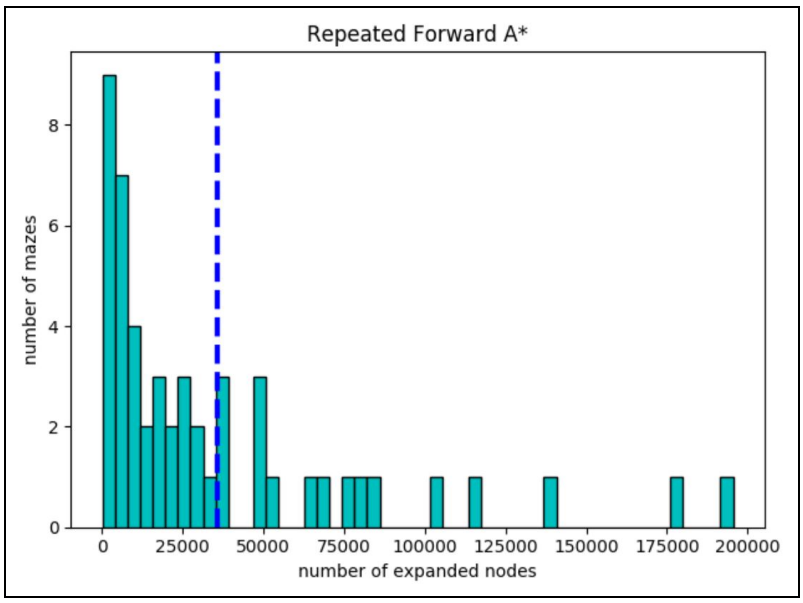

Figure 3

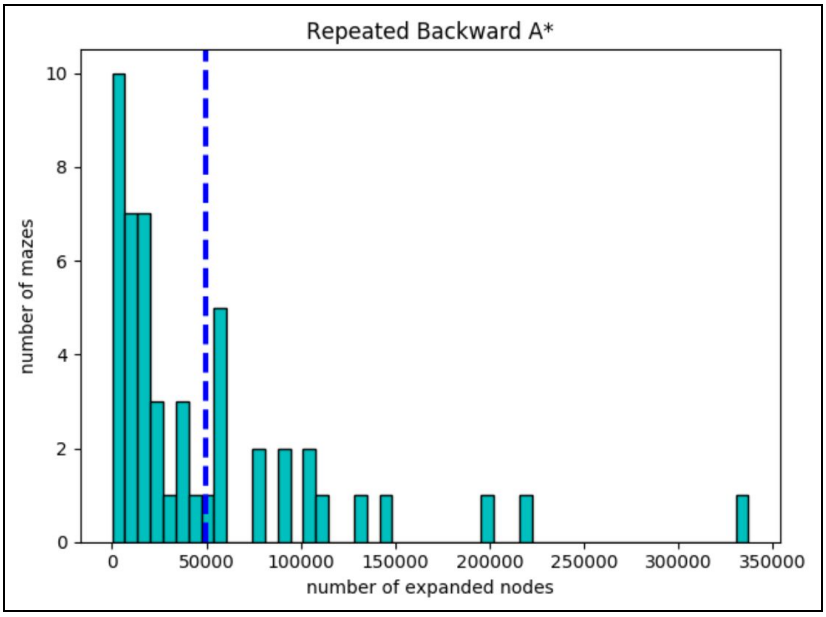

Figure 4

Part 4 - Heuristics in the Adaptive A*

Manhattan distances are consistent if the following inequality holds true for the heuristic:

$h(n) \leq h(s) + C(n,s)$, *where n and s are states*	
$h(n) = \lvert x_n - x_{goal}\rvert + \lvert y_n - y_{goal}\rvert$, $h(s) = \lvert x_s - x_{goal}\rvert + \lvert y_s - y_{goal}\rvert$.	
$h(n) - h(s) \leq C(n,s)$	
$\lvert x_n - x_{goal}\rvert + \lvert y_n - y_{goal}\rvert - \lvert x_s - x_{goal}\rvert - \lvert y_s - y_{goal}\rvert \leq C(n,s)$	Substitution
$\lvert x_n - x_s\rvert + \lvert y_n - y_s\rvert \leq C(n,s)$	Simplify
$h(n,s) \leq C(n,s)$, $h(n,s)$ = Manhattan Distance from state n to s	
Manhattan takes the sum of horizontal and vertical distance between two states. Because the agent only travels in the 4 cardinal directions, Manhattan distance represents the minimum possible cost of travelling between two cells. This occurs when the path is completely unblocked. From $h(n,s)$, the cost can only increase to circumvent blocked cells. Manhattan distance would not be consistent if the agent was allowed to move in diagonals, as the $C(n,s)$ can be shorter if the path was unblocked there.	
If the agent is allowed to move in diagonals:	
$C(n,s) = \sqrt{(x_n - x_s)^2 + (y_n - y_s)^2}$, *on an unblocked path*	
Assume $x_n - x_s = a$, $y_n - y_s = b$, $\lvert a\rvert + \lvert b\rvert \leq \sqrt{a^2 + b^2}$	
$\lvert a\rvert + \lvert b\rvert \leq \sqrt{a^2 + b^2}$	Does not abide by the triangle inequality.

h_{new} are also consistent. To prove this we have 3 cases:

1. Neither n nor s have been previously expanded (proved above)
2. States n and s have both been expanded previously
3. Only state n has been previously expanded (similar proof for if s has not been previously expanded)

Case 2: States n and s have both been expanded previously. Subscript 1 represents values from previous searches, whereas 2 represents current search.

$h_{new}(n) \leq h_{new}(s) + C_2(n,s)$, *where n and s are states*	
$h_{new}(n) = C_1(start, goal) - g_1(n)$ $h_{new}(s) = C_1(start, goal) - g_1(s)$	
$h_{new}(n) - h_{new}(s) \leq C_2(n,s)$	
$C_1(start, goal) - g_1(n) - C_1(start, goal) + g_1(s) \leq C_2(n,s)$	Substitution
$g_1(s) - g_1(n) \leq C_2(n,s)$	Simplify
$C_1(n,s) \leq C_2(n,s)$	Substitute
With each successive planning of A*, the cost between cells must increase or remain the same because when new blocked cells are discovered, they increase the path cost, and therefore, the distance between some cells. The path cost from one planning to the next within Repeated A* path finding does not decrease. The difference between $g_1(s)$ and $g_1(n)$ is the path cost of n and s from the previous search. Since the cost is nondecreasing for each successive search, the inequality holds true.	

Case 3: Only state n has been expanded in a previous A* search (similar proof for if *s* has not been previously expanded)

$h_{new}(n) \leq h(s) + C_2(n,s)$, *where n and s are states*	
$h_{new}(n) = C_1(start, goal) - g_1(n)$ $h(s) = C_1(start, goal) - g_1(s)$	
$h_{new}(n) - h(s) \leq C_2(n,s)$	
$C_1(start, goal) - g_1(n) - \lvert x_s - x_{goal}\rvert - \lvert y_s - y_{goal}\rvert \leq C_2(n,s)$	Substitution
$C_1(n, goal) - \lvert x_s - x_{goal}\rvert - \lvert y_s - y_{goal}\rvert \leq C_2(n,s)$	Simplify
This case assumes the most optimistic path between *s* to *goal*, meaning that the node *s* has not been considered before, therefore must be a node A* is using to circumvent newly discovered blocked cells. Assuming the distance from *s* to *goal* is in fact a direct path, then the cost of the current search, $C_2(n,s)$, will result in a minimum distance between *n* and *s* calculated by the left side of the inequality. $C_2(n,s)$ cannot be lower than this distance, as this distance assumes there are only as many blocks between *n* to *s* as previously discovered in the path, however there may be	

more blocks resulting in a higher $C_2(n, s)$.

Part 5 - Heuristics in the Adaptive A*

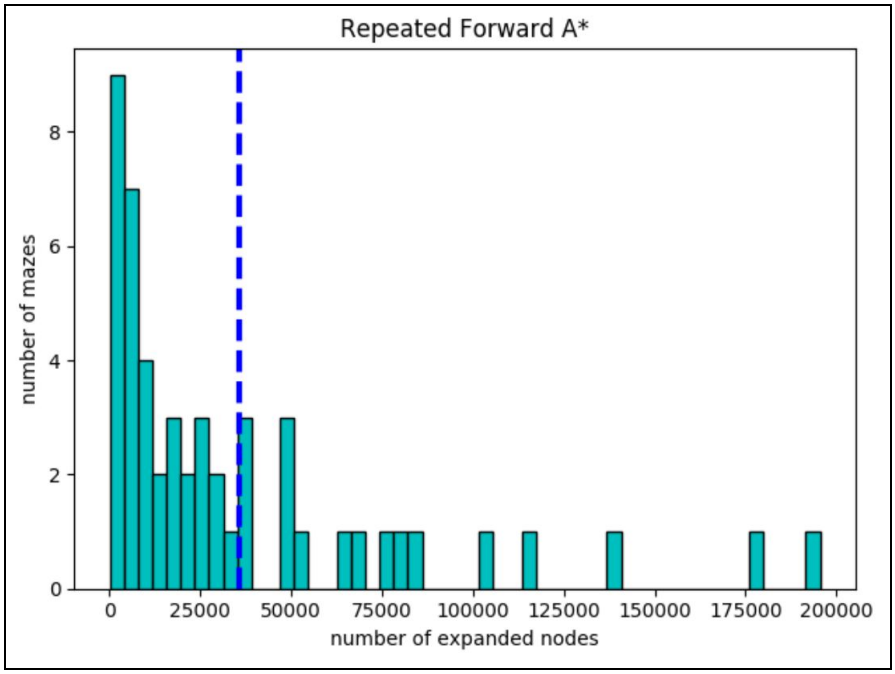

Figure 5

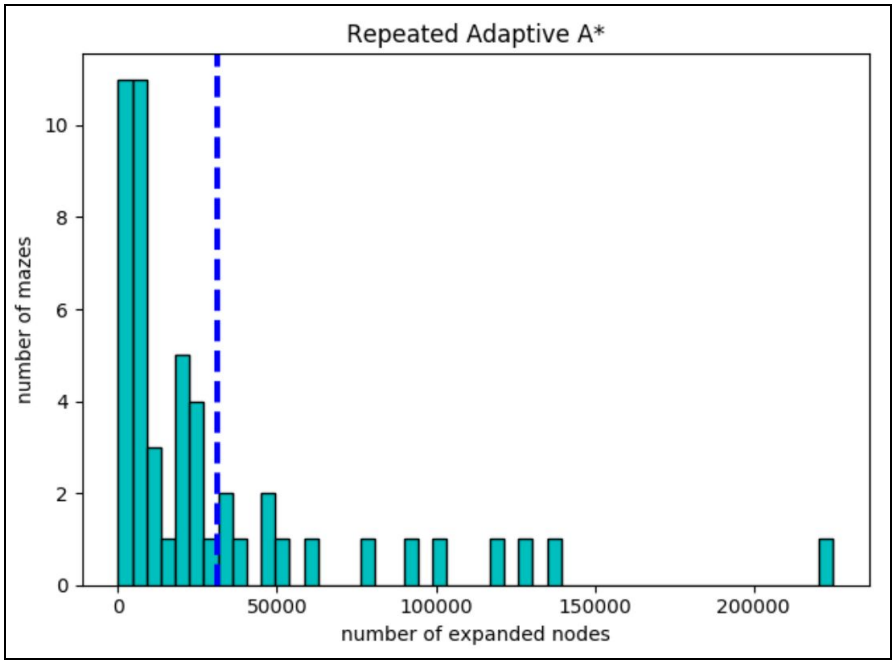

Figure 6

We measured the amount of expanded cells to compare forward A* and adaptive A*. The average number of expanded cells for forward A* is 35,733 whereas the average for adaptive A* 31, 296. The number of cells expanded by adaptive A* is therefore less than the number of cells expanded by forward A*. This is because adaptive A* is a more focused search as it takes in account more accurate heuristics with each successive search. With adaptive A*, the cost of the actual path due to blocked cells is taken into account, therefore cells previously assumed to be closer to the goal might not be expanded due to discovered blocked cells. Additionally, the standard deviation for adaptive A* is 44,640, whereas standard deviation for forward A* is 44,741. There standard deviations for both are quite close, however Adaptive A* seems to be a more reliable search method due to its smaller standard deviation.

The closed list and goal cost from the previous search is saved and used in the current search to update heuristic values. When a node is first generated in the current A* planning, we check if it is in the last closed list, if it is then the heuristic is updated as:

$$h_{new}(n) = g(goal) - g(n)$$

where g(n) is the g(n) from the previous search. If the cell is not in the last closed list and was never generate in previous plannings, or if it the first planning, we update the h value to be Manhattan distances.

Part 6 - Memory Issues

In the current implementation of the gridworld, each cell has eleven attributes: g, h, f, cost, x, y, and 5 pointers that reference parent, left_child, right_child, top_child, down_child, parent. The first six attributes are of type int which cost 4 bytes each, and each of the five pointers costs 8 bytes.

For future improvements:

- We can delete the four pointers to the four children and use the indices of the gridworld with direct access.
    - Additionally, we can call functions to calculate the indices of the children, as we know the current cell's x and y value.
- We can save the x and y coordinates of each cell in a single int. Since the maze is of 101* 101, each cell can have an index value ranging from 0 to 100*100. This would require 4 bytes of int memory.

- Instead of the parent being a node(pointer), it could be an integer ranging from 0-3, where each integer represents either left child, right child, top child, or down child. This would only require 2 bits per cell.
- Since we calculate the h scores using manhattan distance, we can stop storing them as an attribute to each cell and recalculate them each time.

Calculate the amount of memory that they need to operate on gridworlds of size 1001 × 1001:

Since each cell in the current implementation consumes 4 * 6 + 8 * 5 =64 bytes

The amount of memory needed = 1001 * 1001 * 64 = 64128064 bytes

The largest gridworld that they can operate on within a memory limit of 4 MBytes:

4 MBytes = 4000000 bytes

Since each cell consumes 64 bytes, the total number of cells can be in the gridworld is 4000000/64 = 62500 cells

The gridworld is of size sqrt(62500) = 250 * 250