RBE550: Motion Planning

Project 2: Point Planning and Discrete Search

Student 1: Jeel, Chatrola
Student 2: Dev, Soni

Theoretical Questions:

Solution 1

Difference between Bug 1 and Bug 2 Algorithm.

Bug 1	**Bug 2**
It circumnavigates around the obstacle before moving towards the goal	It moves along the obstacle until it encounters m-line then moves along the m-line towards goal.
It is a exhaustive search algorithm	It is a greedy algorithm

Solution 2

A heuristic h is admissible if $h(n) \leq T(n)$. Where T is the true cost from n to the goal, and n is a node in the graph. In other words an admissible heuristic never overestimates the true cost from the current state to the goal.

A stronger property is consistency. A heuristic is consistent if for all consecutive states (n, n') then $h(n) \leq T(n, n') + h(n')$ where T is the true cost from node n to its adjacent node n' .

(a) Let, the true cost between current node ($\mathbf{c} = [\mathbf{c_x}, \mathbf{c_y}]$) and goal node ($\mathbf{f} = [\mathbf{g_x}, \mathbf{g_y}]$) be

Euclidean Distance $\implies T(n) = \sqrt{(g_x - c_x)^2 + (g_y - c_y)^2}$

Assume that any movement from current to next adjacent node (top, down, left, right, diagonal) has a constant cost of euclidean distance (i.e. top,down,left,right - 1 and diagonal - $\sqrt{2}$).

Admissible Heuristic

Let us consider Chebyshev Distance as a heuristic.
$\implies h_a(n) = \max\left(|g_x - c_x|, |g_y - c_y|\right).$
This Heuristic is admissible because according to definition ($\therefore h(n) \leq T(n)$)
$\therefore\ h_a(n) \leq T(n)$ ($\because$ Chebyshev Distance is always less than Euclidean Distance)

Non-admissible Heuristic

Let us consider Manhattan distance as a heuristic

$\implies h_b(n) = |g_x - c_x| + |g_y - c_y|.$

This Heuristic is non-admissible because according to definition ($\therefore h(n) \leq T(n)$)

$\therefore\ h_b(n) \geq T(n)$ ($\because$ Manhattan Distance is always less than Euclidean Distance)

(b) **Consist/Monotone heuristic**

Given h_1, h_2 is consistent implies

$$h_1(n) \leq T(n, n') + h_1(n')$$

$$h_2(n) \leq T(n, n') + h_2(n')$$

a new heuristic

$$h(n) = \max\left(h_1(n), h_2(n)\right)$$

To prove new heuristic is consistent let us assume that $h_1(n) \geq h_2(n)$ any given set of given nodes.
Now,

$$h(n) = \max\left[T(n, n') + h_1(n'), T(n, n') + h_2(n')\right]$$

$$h(n) = \max\left[h_1(n'), h_2(n')\right] \quad (\because max(a + x_1, a + x_2) = max(x_1, x_2))$$
$$h(n) = h_1(n') \quad (\because h_1(n) \geq h_2(n))$$

Solution 3

(a) Assumptions,
Intersection of two line segments can be computed in constant time.
n = the total number of vertices of the obstacles.

- We have n vertices of obstacles $\implies$ n edges of obstacles.
- To match n vertices with (n - 1) other vertices takes $O(n^2)$
- Compare this pair of vertices with n edges takes $O(n^2 * n) = O(n^3)$

Pseudocode for the Algorithm.

Algorithm 1 Brute Force Visibility Graph

```
function CONSTRUCT_VISIBILITY_GRAPH(L1, L2)
    array_nodes = [ obstacles_vertices, start, goal ]        ▷ Add all the vertices of obstacles, start and goal

    for every pair (n1, n2) of vertices in array_nodes do
        if  check if pair (n1, n2) is a obstacle edge then
            Insert (n1, n2) into graph
        else
            for  each obstacle edge do
                if  segment (n1, n2) intersects edges  then
                    goto to next pair
            Insert (n1, n2) into graph
```

(b) **Yes**, we can use the visibility graph to plan the path from start to goal as visibility graph is a standard graph with n nodes and m edges. We can use any graph search algorithm such as

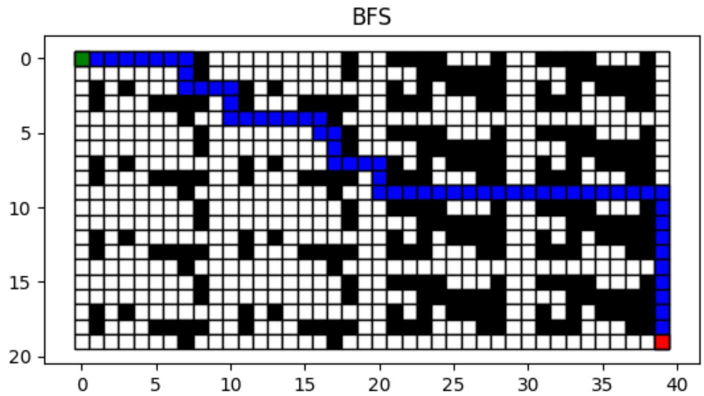
- Djikstra's Algorithm - $O(m \log n)$
- Breadth First Search/Depth First Search - $O(n + m)$
- A^* Algorithm (Depends on Heuristic) - $O((m + n) \log n)$

Programming Questions:

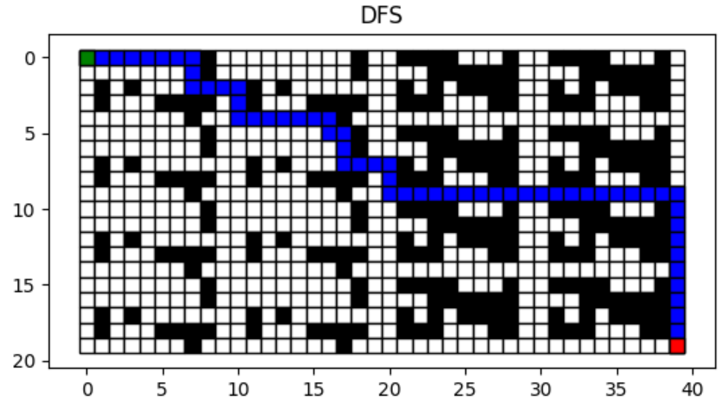
Solution 1 - Refactoring the Code

Solution 2 - Basic Doc Test

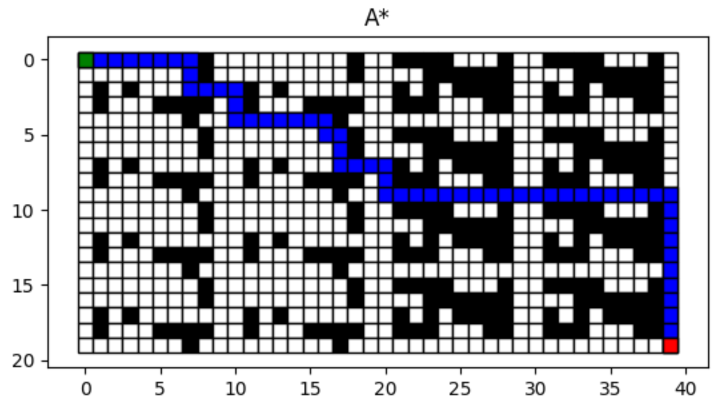
Solution 3 - Various Test Cases



(a) BFS Large Map



(b) DFS Large Map



(c) A^* Large Map

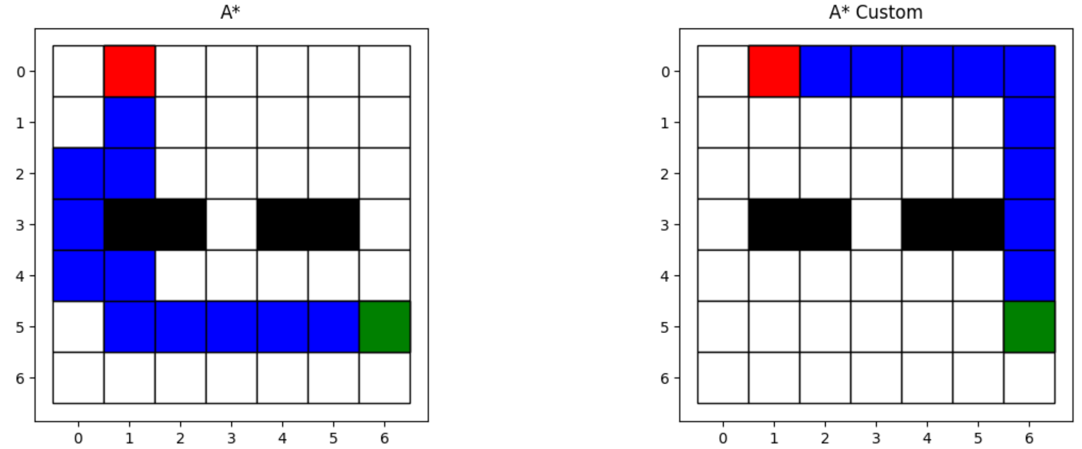
Solution 4 (Bonus Question)

As we know for given the allowed movement of 4 directions (manhattan distance) is the optimal heuristic as it is the true cost between two points on the grid.

So to create a better heuristic we need to have some type of bias in heuristic related to the map/grid structure.

1. $\mathbf{h(n) = (c_y - g_y)}$

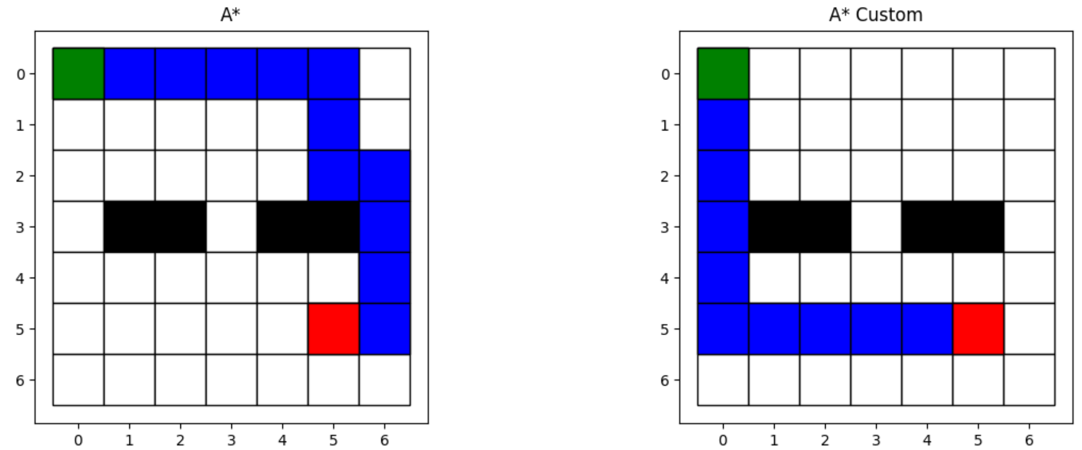
   This Heuristic contain a bias where it rewards movement in columns by reducing the cost (negative because row index of goal is larger than row index of start). Similar, heuristic can be developed for favorable column movement (i.e. $h(n) = (c_x - g_x)$)



(a) A^* with Manhattan Heuristic
Path 13 - steps 14

(b) A^* with Heuristic 1
Path 13 - Steps 11

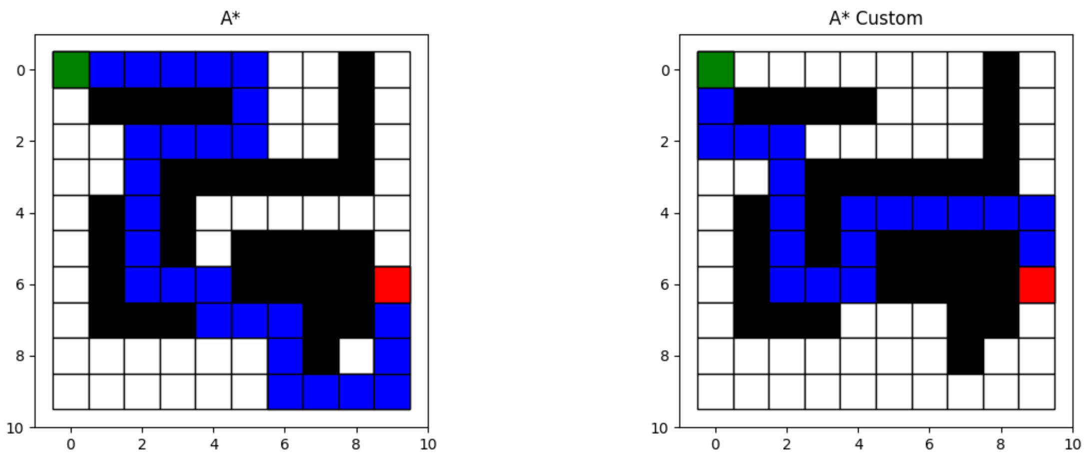
Figure 2: Manhattan VS Custom 1 (Custom Map)



(a) A^* with Manhattan Heuristic
Path 13 - steps 13

(b) A^* with Heuristic 2
Path 11 - Steps 37

Figure 3: Manhattan VS Custom 2(Custom Map)



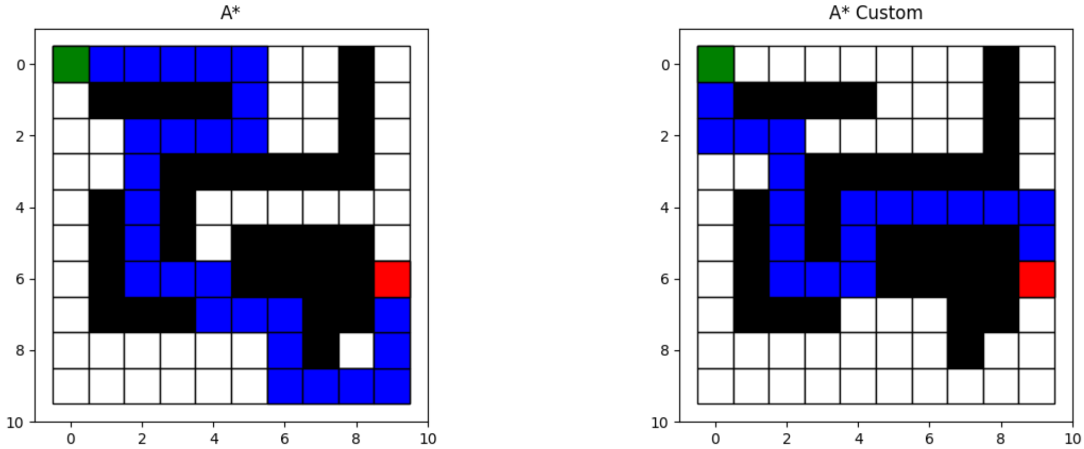
(a) A^* with Manhattan Heuristic
Path 28 - steps 40

(b) A^* with Heuristic 2
Path 20 - Steps 46

Figure 4: Manhattan VS Custom 2 (Large Map)

2. **h(n) = 0 (Dijkstra's Algorithm)**

   This turns A^* into Dijkstra's Algorithm which explores and accounts only the true cost into the computation. It increases the number of nodes visited (i.e. steps in our example) but ensures that we get a optimal path length. As heuristics introduces a bias which can lead to sub-optimal results in some cases.

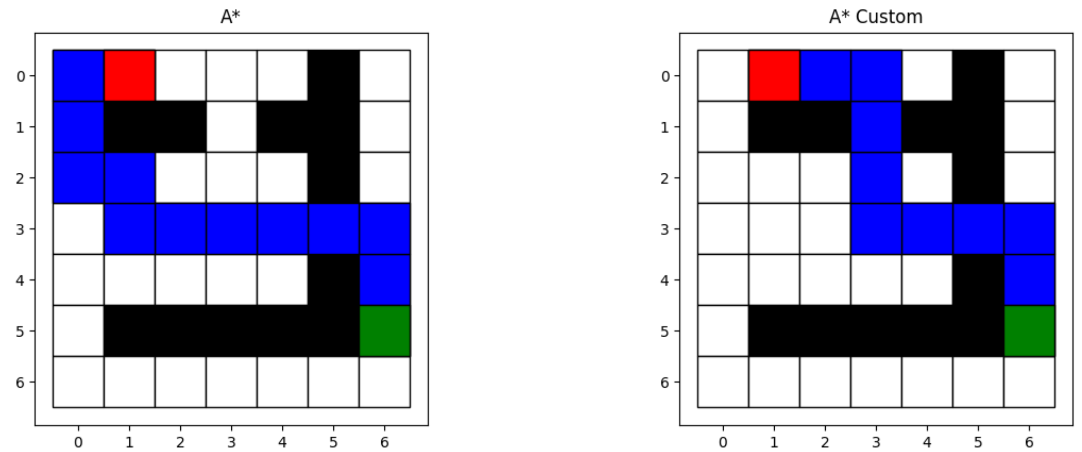


(a) A^* with Manhattan Heuristic
Path 28 - steps 40

(b) A^* with Heuristic 2
Path 20 - Steps 64

Figure 5: Manhattan VS Dijkstra's (Large Map)

In the case of A^* on the left, we move in the left direction due to our preference of that direction in the list, but Dijkstra's algorithm always accounts the true cost thus yields a optimal path as seen on the left.



(a) A^* with Manhattan Heuristic
Path 13 - Steps 14

(b) A^* with Heuristic 2
Path 11 - Steps 36

Figure 6: Manhattan VS Dijkstra's (Custom Map)

In general maps with long zig-zag patterns where it is difficult to find optimal path based on just neighbours is where Dijkstra's algorithm beats the A^* algorithm. But Most of the times A^* is very computationally efficient and give nearly optimal paths.