CNSCC.361 Artificial Intelligence

Lab Session 4 A* Search

This lab is intended for you to learn more about search and planning. You will see a MATLAB implementation of the A* search method in a 2D grid world. Even though the 2D grid world is simple, you can see in more detail how A* search works.

You are not expected to go through everything in this document during the lab session. It will take you longer to understand the details of code, and you can do this in your own time.

Starting Code

The code is available on Moodle. The "Lab 04" folder contains some .m files and some .txt files. "A_star_search.m" is the main file that implements A* search. "add_element_priority_queue.m" is a function that is used multiple times in the main A* search code, so I have separated that out as a function to make the code easier to read. "2D_world_00.txt" describes a 5x5 grid world, and "2D_world_01.txt" describes a 10x10 grid world. In these grid worlds, "0" stands for empty space and "1" stands for occupied space (or obstacles).

Open MATLAB and open the "A_star_search.m" file. At the beginning of that file, type the following (the first time it says "put code here…"):

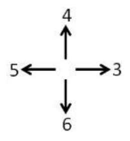
```
world_grid = dlmread('2D_world_00.txt',' ');
[r_world_grid, c_world_grid] = size(world_grid);
start_pos = [4 3];
goal_pos = [1 5];
```

Also, try to execute these statements directly from the MATLAB prompt, and print out these variables to see what values they have.

Type the following (the second time it says "put code here…"):

```
temp_dist = (goal_pos-start_pos) .* (goal_pos-start_pos);
priority_queue(1,:) = [start_pos 2 0 sqrt(sum(temp_dist))];
```

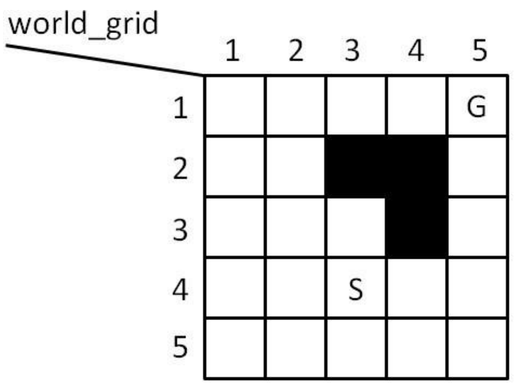
The `priority_queue` variable is a 2D array, where each row is an element of the priority queue and there are five values in each row/element. The first two values specify the (x,y) position in the `world_grid` variable. The third value specifies the direction, which is used later to trace the solution path in reverse from the goal to the start. The directions are:



Note that the integers (2,3,4,5,6) are also used in `world_grid` to represent that a cell in the grid has already been expanded (with the integer 2 indicating no direction). The fourth value in each row/element of `priority_queue` is the distance of the actual path so far, from the start to that cell (g). The fifth value is the total path cost (f=g+h). We saw in the lecture that the total path cost (f) is equal to the actual path so far (g) + the estimate of the future path to goal (h). The rows or elements in the `priority_queue` are sorted according to this fifth value.

Each element in the priority queue is a node that may be expanded. The idea of the priority queue is to sort these elements in order, so that the least cost element can be expanded each time we want to expand a node.

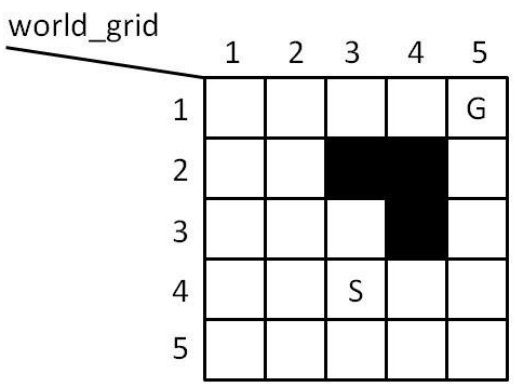
Take a quick look at the rest of the code to get a sense of what each part is doing. Then, you can run the code directly by pressing "Run". You will see a solution path in the form of a list of (x,y) points in reverse, from the goal back to the start. You can draw this solution path here to better visualise it:



Try to write down the cells that were expanded and indicate these in the above figure. Try printing the `world_grid` and `priority_queue` variables at the MATLAB prompt to see their values. In order to get a better understanding of the code, you can manually go through the code to see what each line of the code does for this simple 2D world (try this yourself later).

Some Questions about the Code

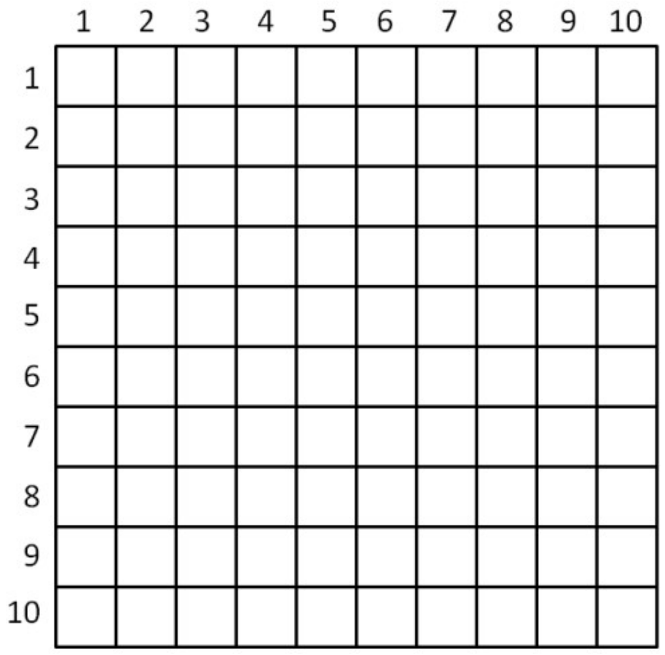
(1) Which part of the code implements the total path cost formula of f = g + h? What happens if we change this to just f = g? What kind of search would this become? Try this on the 5x5 grid world and the same start and goal given above. You will see that the solution path is the same. But the expanded cells are different. Print the `world_grid` variable and draw it here:



(2) What happens if we now change the total path cost to just f = h? What kind of search would this become? The `world_grid` is the same as before this time, but the `priority_queue` is different.

(3) Try the “2D_world_00.txt” file with different start/goal positions, and see the results. You can go back to using f = g + h.

(5) Now, try the “2D_world_01.txt” file (with different start/goal positions). Draw the result here:



(6) Create your own 2D world (10x10 grid) with more obstacles, and find the solution path with the code. You can draw the grid and the solution here:

	1	2	3	4	5	6	7	8	9	10
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										

(7) In the below empty 10x10 grid world, try to come up with an example where there is no solution, and run it with the code. You can draw the grid here:

	1	2	3	4	5	6	7	8	9	10
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										