Autonomous Robots
Graph Search – A* algorithm

This document will guide you through the practical work related to path planning algorithms for searching a graph efficiently. In particular, this practical exercise consists on programming the A* algorithm to solve a Visibility Graph. All the code has to be programmed in Matlab.

1. Visibility Graph

This programming exercise will use the output of the previous laboratory exercise in which a visibility graph is represented using 2 Matlab variables:

→ "vertices" variable, which uses the following format:

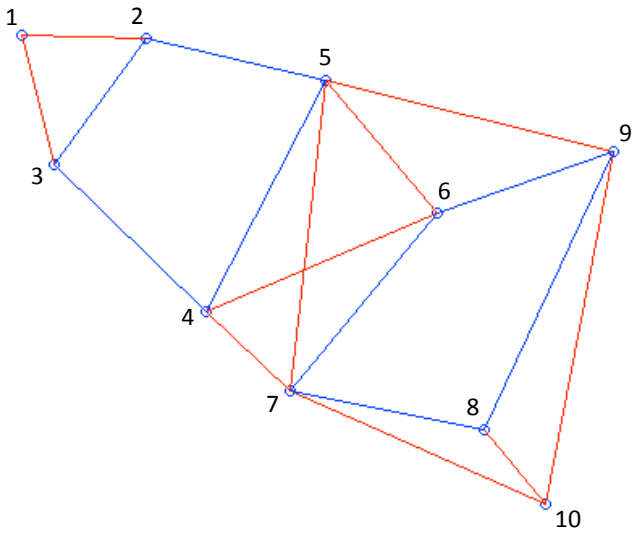
- Columns:
  - First column: x coordinate
  - Second column: y coordinate
  - Third column: object number
- Rows:
  - Each row represents a vertex
  - The first row will ALLWAYS represent the START vertex
  - The last row will ALLWAYS represent the GOAL vertex
  - Vertices from the same object will be inserted sequentially. The object can be plotted by joining with a straight segment each vertex with the next one, and closing at the end the object from the last vertex till the first one.

→ "edges" variable, which uses the following format:

- Columns:
  - First column: indicates one of the vertices of the edge, pointing to one row from the "vertices" variable.
  - Second column: indicates the second vertex of the edge, pointing to one row from the "vertices" variable.
- Rows:
  - Each row represents a different edge of the visibility graph. The same edge will not be repeated by changing the order of the vertices. Each edge will only appear once in the list. There is not any rule in the order of the edges in the list. They are listed without any particular rule.

Here you can see an example of a visibility graph represented by the two variables and plotted in the following figure:

```
>> vertices

vertices =

    0.7807    9.0497         0
    3.0322    8.9912    1.0000
    1.3655    6.7105    1.0000
    4.1140    4.0497    1.0000
    6.2778    8.2310    1.0000
    8.2953    5.8333    2.0000
    5.6345    2.6170    2.0000
    9.1433    1.9152    2.0000
   11.4825    6.9444    2.0000
   10.2544    0.5702    3.0000


>> edges=RPS(vertices)

edges =

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



2. A* algorithm

WORK TO DO: Program the A* algorithm to solve the visibility graph, going from the start vertex to the goal vertex following the minimum path. You can use the algorithm description explained in class or the next pseudocode.

```
function A*(start,goal)
     closedset := the empty set    // The set of nodes already evaluated.
     openset := {start}    // The set of tentative nodes to be evaluated, initially containing the start node
     came_from := the empty map    // The map of navigated nodes.

     g_score[start] := 0    // Cost from start along best known path.
     // Estimated total cost from start to goal through y.
     f_score[start] := g_score[start] + heuristic_cost_estimate(start, goal)

     while openset is not empty
         current := the node in openset having the lowest f_score[] value
         if current = goal
             return reconstruct_path(came_from, goal)

         remove current from openset
         add current to closedset
         for each neighbor in neighbor_nodes(current)
             tentative_g_score := g_score[current] + dist_between(current,neighbor)
             if neighbor in closedset
                 if tentative_g_score >= g_score[neighbor]
                     continue

             if neighbor not in openset or tentative_g_score < g_score[neighbor]
                 came_from[neighbor] := current
                 g_score[neighbor] := tentative_g_score
                 f_score[neighbor] := g_score[neighbor] + heuristic_cost_estimate(neighbor, goal)
                 if neighbor not in openset
                     add neighbor to openset

     return failure


 function reconstruct_path(came_from, current_node)
     if current_node in came_from
         p := reconstruct_path(came_from, came_from[current_node])
         return (p + current_node)
     else
         return current_node
```

Use as a heuristic function the Euclidean distance.

You must program a Matlab function with the following input and output parameters:

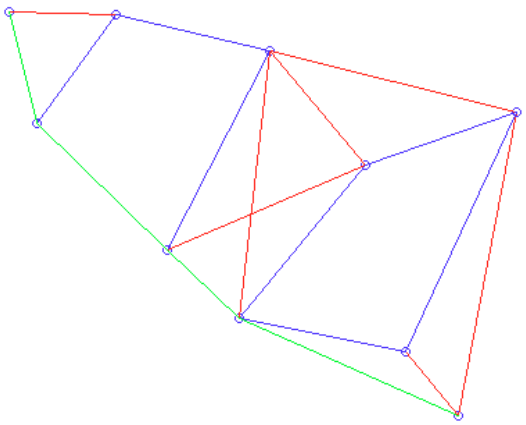
```
function [path,minCost]=Astar(vertices, edges)
```

The “path” variable is a vector variable that contains the ordered list of vertices that are part of the minimum path. Since the first vertex of the “vertices” variable is the start vertex and the last vertex is the goal vertex, the “path” variable will always start by “path = [1, …” and finish by “…, numberRowsVertices]”.

The “minCost” variable is a scalar value that indicates the length of the optimal path.

When Astar function is called with the previous input variables, the next result will be obtained. On the left you have a representation of the environment (blue) the visibility graph (red) and the shortest path (green).

```
path =

     1     3     4     7    10

minCost =

   13.3788
```



It is very important that you use for the “path”, “minCost”, “vertices” and “edges” variables EXACTLY the same format you can see in the previous example. Your “Astar.m” file will be evaluated exclusively based on the results it generates which must follow the previous format. The algorithms will be tested with different environments.

Here you have another example of visibility graph:

vertices =

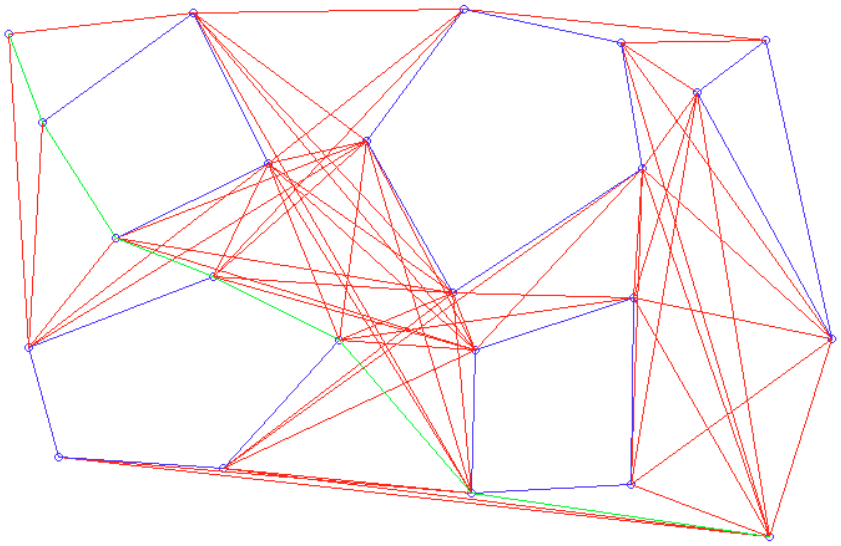
0.6053	7.9971	0
1.0439	6.8567	1.0000
2.9737	8.2602	1.0000
3.9386	6.3304	1.0000
1.9795	5.3655	1.0000
6.4532	8.3187	2.0000
5.1959	6.6228	2.0000
6.3070	4.6637	2.0000
8.7339	6.2719	2.0000
8.4708	7.8801	2.0000
3.2368	4.8684	3.0000
0.8684	3.9620	3.0000
1.2485	2.5585	3.0000
3.3538	2.4123	3.0000
4.8450	4.0497	3.0000
6.5994	3.9327	4.0000
6.5409	2.0906	4.0000
8.5877	2.2076	4.0000
8.6170	4.6053	4.0000
9.4357	7.2368	5.0000
11.1608	4.0789	5.0000
10.3129	7.9094	5.0000
10.3713	1.5351	6.0000

edges =

1	3
1	12
1	2
2	3
2	12
2	5
3	6
3	4
3	17
3	16
3	8
3	7
4	7
4	6
4	5
4	12
4	11
4	15
4	17
4	16
4	8
5	7
5	12
5	11
5	16
5	8
6	11
6	7
6	10
6	22
7	12
7	11
7	15
7	17
7	16
7	8
8	9
8	11
8	15
8	14
8	17
8	16
8	19
9	20
9	10
9	14
9	16
9	19
9	18
9	23
9	21
10	22
10	23
10	21
10	20
11	12
11	15
11	16
12	13
13	23
13	17
13	14
14	16
14	15
14	23
14	17
15	19
15	17
15	16
16	19
16	17
17	18
17	23
18	21
18	20
18	19
18	23
19	20
19	23
19	21
20	22
20	23
20	21
21	22
21	23

```
path =

     1     2     5    11    15    17    23

minCost =

   12.6005
```

And the graphical representation would be:



3. Submission.

WORK TO DO: Submit a report in pdf and the Matlab file “Astar.m”. Explain in detail, in the report, the work done. Show the environment and the shortest path graphically. Explain also the problems you found. You might want to test your algorithm in other environments.

NOTE that only these 2 files are required and will be evaluated. Do not send more files.