Universitat de Girona

Escola Politécnica Superior

Intelligent Field Robotic Systems

AUTONOMOUS SYSTEMS

LAB4 - Graph Search - A* Algorithm

Prepared by

Esra Hicret Uzer

Jesús Enrique Alemán Gallegos

Girona, December, 2021

INTRODUCTION

The present report describes the development of the fourth laboratory practice which is about implementing a graph search algorithm for finding the shortest path from a starting position to a goal position. The algorithm to be implemented is the A* pathfinding algorithm, which is an informed graph search algorithm that allows to efficiently compute the shortest path from a start node to a goal node.

The A* algorithm is a variation of the Dijkstra algorithm, which has the same purpose of finding the shortest path, however A* uses an heuristic to focus the search on the direction of the desired optimality (which in most cases is to minimize the total distance traveled). The heuristic can be a metric such as energy, time, safety, etc. In the robot path planning case a common heuristic to be considered is the euclidean distance from a starting node to the goal node as shown in Figure 1, this heuristic is an informed hypothesis of the cost to reach the goal node. If the heuristic is optimistic, the search will be efficient and the resulting path will be efficient with respect to the heuristic criteria, otherwise if the heuristic is non-optimistic the search will not be efficient but a feasible path could still be found.

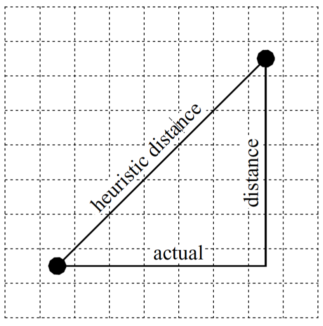

Figure 1. Euclidean distance as heuristic

The A* algorithm can be applied to any kind of graph structure, which in robotics it is very common to have once the environment where the robot moves is defined by a Visibility Graph, a Voronoi Diagram, a Probabilistic Roadmap or a grid. The algorithm's pseudocode proposed by [1] is shown below, however in the current implementation few changes took place as it will be explained in the following section.

```
Input: A graph
Output: A path between start and goal nodes
repeat
    Pick $n_{best}$ from $O$ such that $f(n_{best}) \leq f(n), \forall n \in O$.
    Remove $n_{best}$ from $O$ and add to $C$.
    If $n_{best} = q_{goal}$, EXIT.
    Expand $n_{best}$: for all $x \in$ Star($n_{best}$) that are not in $C$.
    if $x \notin O$ then
        add $x$ to $O$.
    else if $g(n_{best}) + c(n_{best}, x) < g(x)$ then
        update $x$'s backpointer to point to $n_{best}$
    end if
until $O$ is empty
```

Figure 2. A* Pseudocode as proposed by [1]

It is important to mention that the computational complexity of the A* (considering an optimistic heuristic) is $O(b^d)$ where d is the depth to the shortest path and b is the branching factor. Computational complexity of similar algorithms is shown in Table 1. Also, Figure 3 shows the expansion of the algorithms mentioned in Table 1 for visualization and comparison purposes.

Table 1. Common path planning algorithms complexities

Algorithm	Computational Complexity
A*	$O(b^d)$
Dijkstra	$O(\lvert V\rvert log\lvert V\rvert + \lvert E\rvert)$
Breadth First	$O(\lvert V\rvert + \lvert E\rvert)$
Depth First	$O(\lvert V\rvert + \lvert E\rvert)$

* V is the number of vertices in the graph and E is the number of edges

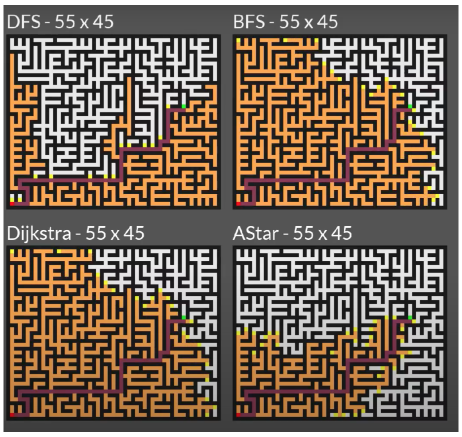

Figure 3. Expansion comparison between algorithms shown in Table 1

Development

The implementation of the A* algorithm was straight forward and took place in the visibility graph built for the environments in the laboratory practice number 3. The input to the program are 2 csv files, the first one defines the vertices in the environment and the second one defines the visibility edges between vertices (that are visible from one to another). Given the visibility edges it is easy to get the adjacent (children) nodes from each edge and find a path from the start vertex (first vertex in the vertexes csv file) to the goal vertex (last vertex in the vertexes csv file). It is important to mention that this algorithm can also be implemented for an environment defined with a grid. The following pseudocode describes the step by step implementation of the A* algorithm.

```
$A*(vertexes, start, goal)$ :
  0.Initialize open and closed lists as empty lists
  1.The start vertex becomes the current vertex
  2.Calculate the heuristic $h$ of the start vertex the goal vertex
  3.Calculate the $f$ value of the start vertex, considering $g$ (start) = 0
  4.While the current vertex is not the goa vertex do:
      5.For every child vertex of the current vertex do:
          6.If child vertex is not in the closed list and is not in the open list then:
              7.Add child vertex to open list
              8.Calculate $g$ (child vertex)
              9.Calculate $h$ (child vertex)
              10.Calculate $f$ (child vertex)
              11.If new $f$ (child vertex) value < existing $f$ (child vertex) or no $f$ value assigned to child vertex then:
                  12.Update $f$ value of the child vertex
                  13.For the child vertex set current vertex as parent vertex
      End For
      14.Add current vertex to the closed list
      15.Sort open list in ascending order with respect the $f$ values of each vertex in the list
      16.Remove the first element of the sorted open list and set it as the current vertex
  17.Return Closed list with backpointers showing the shortest path from the start vertex to the goal vertex.
```

The following notation is considered in the pseudocode for the A* algorithm:

h =Heuristic value from a vertex to the goal vertex, for this implementation the heuristic is the euclidean distance as shown in Figure.

g =Distance from start vertex to another vertex

$f = g + h$

The open and closed lists play a fundamental role for the algorithm to work correctly, the open list stores all the vertices to be explored and is sorted according to their *f* score in ascending order. It is also important to notice that the vertices in the open list can be updated in case a lower *f* score is found following another trayectory. The closed list stores the already explored nodes and it is used to retrieve the shortest path once the algorithm has terminated its execution.

RESULTS

The present Figures 4 to 9 show the results of implementing the A* algorithm on the visibility graphs of each environment. The shortest path will be generated from every starting vertex 0 to every goal vertex (which for these cases are the vertices with the largest index number) . Also every result comes with a list of vertices to be followed to reach the goal with the minimum distance and the total distance to be traveled.

Env 0

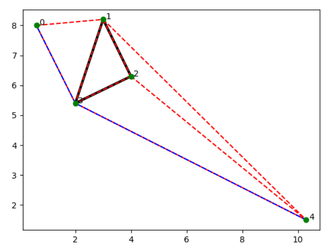

Figure 4. Env 0 shortest path

Path: [0, 3, 4]

Distance: 12.12356982653498

Env 1

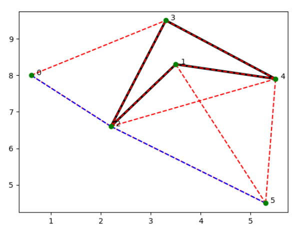

Figure 5. Env 1 shortest path

Path: [0, 2, 5]

Distance: 5.870358207916741

Env 2

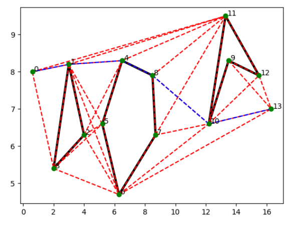

Figure 6. Env 2 shortest path

Path: [0, 1, 4, 8, 10, 13]

Distance: 15.990555296232605

Env 6 (Env 2 with inverted start and goal vertices)

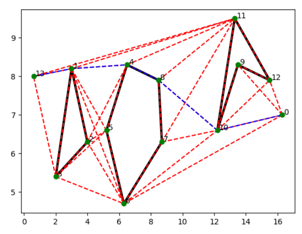

Figure 7. Env 6 shortest path

Path: [0, 10, 8, 4, 1, 13]

Distance: 15.990555296232605

Env 9 (Extra environment)

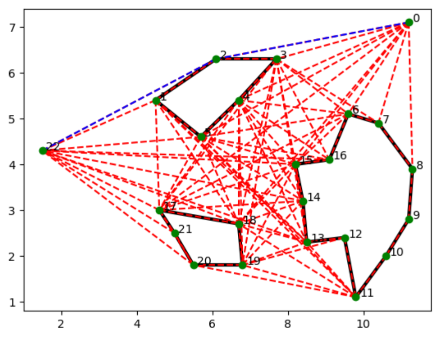

Figure 8. Env 9 shortest path

Path: [0, 2, 22]

Distance: 10.178338281249903

Env 11 (Mexico from the Pacific to the Atlantic ocean)

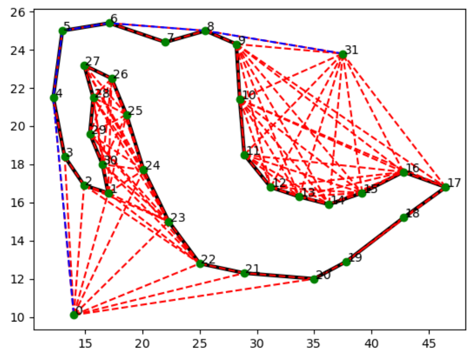

Figure 9. Env 11 shortest path

Path: [0, 4, 5, 6, 8, 31]

Distance: 39.7203301980117

It was possible to retrieve the shortest path in every environment, even when inverting the position of the start and goal vertices such as shown in Figures 6 and 7, having found exactly the same path trajectory and the same distance (as expected). The algorithm works efficiently even in complex environments with multiple vertices and edges such as in Figures 8 and 9, computing the shortest path from the start vertex to the goal vertex.

Conclusion

The present laboratory practice allowed us to implement one of the most popular path planning algorithms to solve the planning problem for robot motion. It was very interesting to see how the topic of visibility graphs, roadmaps and path planning relate to each other in order to create a plan on how to move from a point to another. Since A* has had many implementations in different fields and it is a very studied algorithm, it was relatively easy to find resources to learn about it and implemented correctly. At the end, the learning objectives were achieved and the algorithm was robust given different types of environments and constraints.

References

[1] Choset, H. M. (2005). Principles of robot motion: Theory, algorithms, and implementation. Cambridge, Mass: MIT Press.