

Shortest Path Planning

Computational Intelligence, Lecture 13

by Sergei Savin

Spring 2024

SHORTEST PATH ON A GRAPH

If we want to plan a path on a 2D map, we can represent obstacle-free space regions as a nodes, and possible transitions between the obstacle-free space regions as graph edges.



Figure 1: Path planning as graph search; Credit:
<https://demonstrations.wolfram.com/ProbabilisticRoadmapMethod/>

SHORTEST PATH ON A GRAPH

Consider a directed graph (each edge has a direction assigned to it):

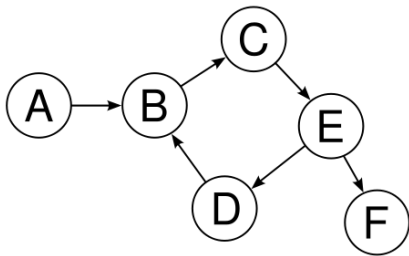


Figure 2: Directed graph; Credit:
<https://github.com/HQarroum/directed-graph>

How can we find a shortest path from a start node to a finish node on it?

SPP as LP

SHORTEST PATH (1)

We assign index variable x_i to i -th edge; each index variable is positive $x_i \geq 0$.

If $x_i = 1$ the edge is part of the path. We assume that otherwise $x_i = 0$ (which will be enforced by the other constraints).

Adding a cost d_i associated with each edge (e.g. Euclidean distance) we get a linear cost $l(\mathbf{x})$:

$$l(\mathbf{x}) = \mathbf{x}^\top \mathbf{d} \tag{1}$$

SHORTEST PATH (2)

Since each edge connects one node (e.g. node u) to another (e.g. node v), we can label all index variables x with superscripts, denoting nodes that they connect - $x^{u,v}$.

Our goal will be to count how many path segments enter and leave each node. For any normal node the number will be equal:

$$-\sum_{\forall i} x^{i,v} + \sum_{\forall j} x^{v,j} = 0 \quad (2)$$

We know that for the starting node s , there will only be one path segment leaving it:

$$-\sum_{\forall i} x^{i,s} + \sum_{\forall j} x^{s,j} = 1 \quad (3)$$

For the finishing node f we have only one path segment entering it:

$$-\sum_{\forall i} x^{i,f} + \sum_{\forall j} x^{f,j} = -1 \quad (4)$$

Together the problem becomes:

$$\begin{aligned} & \underset{\mathbf{x}}{\text{minimize}} && \mathbf{x}^\top \mathbf{d}, \\ & \text{subject to} && \begin{cases} -\sum_{\forall i} x^{i,v} + \sum_{\forall j} x^{v,j} = 0, & \forall v \\ -\sum_{\forall i} x^{i,s} + \sum_{\forall j} x^{s,j} = 1, \\ -\sum_{\forall i} x^{i,f} + \sum_{\forall j} x^{f,j} = -1, \\ \mathbf{x} \geq 0. \end{cases} \end{aligned} \tag{5}$$

And with that, the problem can be solved as an LP.

SPP CODE (1)

```
0 n = 5; V = randn(n, 2);  
  % Connectivity:  
2 C = [1, 2; %edge 1  
      1, 3; %edge 2  
      2, 3; %edge 3  
      2, 4; %edge 4  
      3, 5; %edge 5  
      4, 5]; %edge 6  
8 nc = size(C, 1);  
  d = zeros(nc, 1); %cost - distance  
10 for i = 1:nc  
    d(i) = norm(V(C(i, 2), :) - V(C(i, 1), :));  
12 end
```

SPP CODE (2)

```
0 cvx_begin
  variable x(nc, 1)
2 minimize( dot(d, x) )
  subject to
4 x >= zeros(nc, 1);
  x(1) + x(2) == 1;%v 1
6 -x(5) - x(6) == -1;%v 5

8 -x(1) + x(3) + x(4) == 0;%v 2
  -x(2) - x(3) + x(5) == 0;%v 3
10 -x(4) + x(6) == 0;%v 4
  cvx_end
```

SPP via A^* algorithm

Another popular shortest path planning method for graphs is *A star* (A^*). Unlike the previous method it does not involve optimization, but it requires a *heuristic*.

To study A^* we once more consider a graph whose edges have cost associated with them.

Let p be a node of the graph that the program found a path to. Point p has a predecessor point $a(p)$ - the last node in the path towards p . Since each predecessor knows its predecessor, it means we can recursively reconstruct the path from the point p to the start.

Finding a path from the start to the point p we construct a sequence of edges that we need to travel through - e_1, e_2, \dots, e_n . Each of these edges has a cost associated with them - c_1, c_2, \dots, c_n . So, the cost of reaching a node p is $g(p) = \sum_{i=1}^n c_i$.

If we have a heuristic $h(p)$ that (while more or less accurate) always *underestimates* the cost to reaching goal from the node p , we can use A^* to choose the next node in the path. We choose the node that minimizes the following cost function:

$$p_{next} = \underset{p}{\operatorname{argmin}}(g(p) + h(p)) \quad (6)$$

In practice, when we can compute $g(p)$ much simpler. Given a new node p_{next} and its predecessor p_a , and the cost associated with the edge connecting them c_a , we can assign the value of $g(p_{next})$ as:

$$g(p_{next}) := g(p_a) + c_a \quad (7)$$

Heuristic might be difficult to formulate in general, but as long as each node has coordinates on a plane associated with it, Euclidean distance provides a suitable heuristic.

A STAR SEARCH - IMPLEMENTATION

A grid can easily be seen as a graph, where adjacency implies connection.

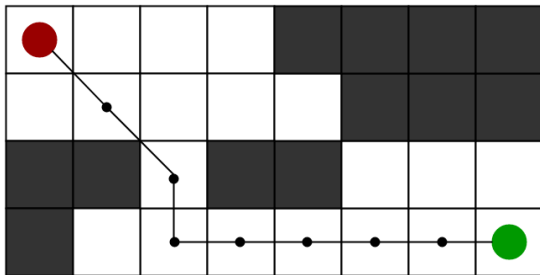


Figure 3: Example of a grid with obstacles. Credit: [geeksforgeeks.org](https://www.geeksforgeeks.org)

Lecture slides are available via Github, links are on Moodle:

github.com/SergeiSa/Computational-Intelligence-2024

