ECE9047: Notes for Laboratory 3

John McLeod

Due Date: 2021 04 04

k-Connectivity

To find the k-connectivity of your network, find the **node** with the **fewest** communication channels. That number of channels is k.

To plot the communication channels in Matlab, you can use the following code (this is probably not the best way, but it is straightforward to understand):

```
% get the data, obviously use the file name and path
% appropriate for you
points = readtable('Data Files/Graph_SAMPLE.txt');

% plot the nodes as blue circles
plot( points.Var1, points.Var2, 'bo' );
% keep the plot active
hold on;

%this is the number of nodes
N = 20;
% this is the communications radius
RC = 0.25;

%loop through all nodes
for p=1:N
        %loop through rest of nodes again
    for q=p+1:N
                %calculate distance between two nodes
        d = sqrt((points.Var1(p)-points.Var1(q))^2 +
                 (points.Var2(p)-points.Var2(q))^2)
        % if distance is less than communications radius, draw a line
        if (d<RC)
            plot([points.Var1(p), points.Var1(q)],
                 [points.Var2(p), points.Var2(q)], 'r-');
        end
    end
end
```

The `hold on;` line keeps the plot active so you can add more lines to the same figure. Feel free to change the colours/symbols if you like. As noted, `'bo'` is for blue circles and `'r-'` is for a red line. See here: link to Matlab website for more options.

k-Coverage

To find the k-coverage of your network, find the **point in the area** with the **fewest** sensors covering it. Typically, but not always, this will be near a corner.

You can draw circles on a plot in Matlab using `viscircles`. This requires the image processing toolbox, which is an add-on. I was too lazy to log in to the Matlab website to download it, so I drew circles by hand:

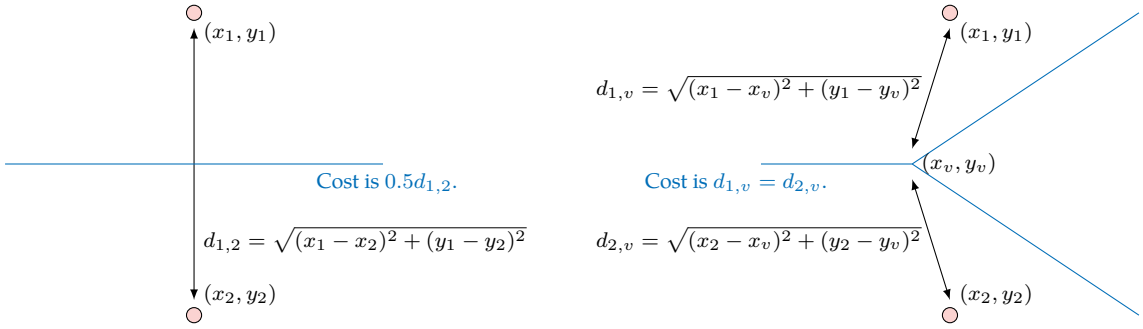
```
%this is sensor radius
RS = 0.20;
%this is list of angles for drawing sensor circle
theta = 0:pi/100:2*pi;

%loop through all nodes
for p=1:N
        %draw a circle using complex math and offset that circle
        %to center on the node
        % note: 1.j is the imaginary number sqrt(-1)
    f = RS * exp(1.j*theta) + points.Var1(p)+1.j*points.Var2(p);
    % add circle to plot
    plot(real(f), imag(f), 'g:');
end
```

This plots the circles using a dotted green circle. If you want filled circles you may need to use `viscircles` or Google some better solution.

Maximal Breach Path

The maximal breach path is the path that stays the **furthest** away from any node: it passes through the area by **maximizing** the distance to the *closest node at any point in the path*. The maximal breach path should follow the lines on the Voronoi tessellation.
The **maximal breach distance** is the *smallest* cost of any line followed on the maximal breach path.

- Each line in the Voronoi tessellation passes halfway between two nodes. Either of these nodes (it doesn't matter which) can be used to calculate the cost.
- If the Voronoi line passes all the way between the two nodes, the cost is **half the distance between those nodes**.
- If the Voronoi line does not pass all the way between the two nodes (because of an intersection with other Voronoi lines), then the cost is **the distance between one of the end-points of the line and one of the nodes**.



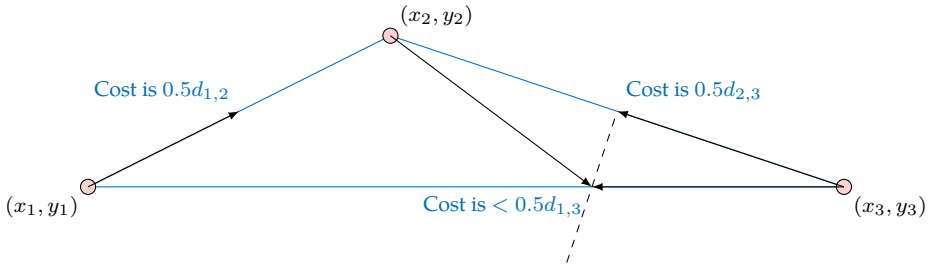
In Matlab, `vx,vy=Voronoi(points.Var1, points.Var2);` will return the line segments of the Voronoi plot. You are welcome to write some script that calculates the breach cost of all line segments, but it probably isn't necessary.

1. You need to draw the maximal breach path from the lower left corner to the upper left corner. Most, possibly all, of this path can probably be found just by inspection, as often some lines are clearly further from nodes than others.
2. You only need to calculate the cost of a line segment when there is doubt which path to take, and to find the maximal breach distance.

Maximal Support Path

The maximal support path is the path that stays **closest** to the nodes: it passes through the area by **minimizing** the distance from the *farthest node at any point in the path*. The maximal support path should follow the lines in the Delaunay triangulation.
The **maximal support distance** is the *smallest* cost of any line followed on the maximal support path.

- For all interior Delaunay triangles, the cost of any line is half the length of that line — as the midpoint on the line is the furthest point from any node, and the closest nodes are the ones on the end point of the line.
- Some Delaunay triangles near the edge of the network may be very long and skinny, and then the furthest point from a node may be different than the midpoint. However in that case the line has a higher cost than other lines, so probably isn't part of the maximal support path.



In the above figure, because the triangle is "long and skinny", the cost of the line between point 1 and 3 less less than the halfway distance. Some simple geometry will allow you to calculate the cost, but in this lab you don't need to — it is clear the maximal support path should pass through node 2 rather than going directly between nodes 1 and 3, as the cost to node 2 is less.

So again, you are welcome to write some script that calculates the support cost of all line segments, but it probably isn't necessary.

1. You need to draw the maximal support path from the lower left corner to the upper left corner. Most, possibly all, of this path can probably be found just by inspection, as often some lines are clearly further from nodes than others.
2. You only need to calculate the cost of a line segment when there is doubt which path to take, and to find the maximal support distance.