ECE9047: Laboratory 3

John McLeod

Due Date: 2024 03 29

Objective

This lab is about evaluating the performance of a pseudo-randomly generated graph, as an abstraction of a wireless sensor network.

- This analysis can be done easily and accurately using Scientific Python (SciPy) or Matlab.
  - Everything you need for SciPy, along with a good IDE, are available, for free, as part of the open-source Anaconda distribution. See: https://www.anaconda.com/products/individual. See the *Resources* section for some tips on using Scipy.
  - Matlab should be available as part of Western's site license. See: https://www.mathworks.com/academia/tah-portal/western-university-964054.html. See the *Resources* section for some tips on using Matlab.
  - There are also some free browser-based shells for SciPy that might work, if you don't want to download any program. But I am not providing support for that.
  - You can run Matlab in your browser as well using the Western site licence (it is one of the options from the site listed above). That should work, but I am not providing support for that either. I am assuming you downloaded the Matlab client.
- The graph for this lab doesn't need that many nodes, so you can do the analysis "by hand" or with a spreadsheet if you don't like SciPy or Matlab.
- There are probably lots of other tools available as well. It doesn't matter what tool you use for analyzing the graph.

The goal of this lab is the following:

> **Complete the pseudo-randomly generated graph to ensure full connectivity. Quantitatively discuss the connectivity and coverage of this graph. Accurately complete a Voronoi tessellation diagram and a Delauney triangulation diagram for the graph. Assess the *maximal breach distance* and the *maximal support distance* of the graph.**

As the nodes are generated pseudo-randomly, many of you may have graphs that are not fully connected, so you will need to add nodes to ensure full connectivity. Some of you may be lucky enough to have a fully-connected graph from the beginning.

- Everyone must present a fully-connected graph.
- Everyone is allowed to add extra nodes to improve coverage and connectivity, at their own discretion. You may add any number of extra nodes.
- Adding lots of extra nodes to improve coverage and connectivity can complicate analysis (especially the Voronoi and Delauney diagrams). On the other hand, carefully adding a node to the right place can sometimes simplify the analysis.
- Adding more nodes for better connectivity and coverage doesn't directly translate into a higher grade (as long as the graph is fully connected, as described above), so there is no need to ensure full coverage or robust connectivity.

Data

Each of you are provided with a list of 25 pseudo-randomly generated nodes. These data are available in you *Drop Box* on OWL (the file called `Graph_123456789.txt`, but with your student number instead of "123456789").

- This is a plain-text file where each line lists the x and y coordinates of a node, comma-separated.
- The coordinates of each node are expressed as fractions of the environment size L, the environment is a square area $L \times L$.
- The communications radius for each node is $R_C = 0.25L$.
- The sensing radius for each node is $R_S = 0.2L$.

Deliverables

You must submit your lab report online to the course website on OWL. Your lab report must include:

1. A statement of the problem: basically, that you are analyzing a pseudo-random network. You can reword the objective given above.
2. Describe the *connectivity* of the network, in terms of K-connectivity, and identifying any areas of the network that might "go dark" if a particular node fails.
3. An accurate diagram of the network, using dots for the nodes and lines connecting nodes that are within communication distance.
4. Describe the *coverage* of the network, in terms of K-coverage, and identifying which area(s) has/have the best coverage and which area(s) has/have the worst coverage. If you needed to add nodes to the network, explain where and why. Note: Unless you decided to add lots of extra nodes, it is quite likely that there are areas that have absolutely no coverage, so the K-coverage of the environment area is zero. This isn't a problem (in terms of completing the assignment, anyway), as long as you describe the situation in your report.
5. An accurate diagram of the network, again using dots for the nodes and circles or coloured areas showing what is covered. You may use semi-transparent colours to help visually identify which areas have the best coverage if you want. You may combine this diagram with the above connectivity diagram if it is not too cluttered.
6. Calculate the *maximal breach distance* of the network, travelling from the coordinate $(0, 0)$ to the diagonal coordinate (L, L).
   - The path followed should go along the boundary of the network area, and along the edges of the Voronoi tessellation.
   - The cost of moving along the boundary of the network area is zero, the cost of moving along the edge of a Voronoi tessellation is the *minimum distance* between that edge and a node.
   - The *maximal breach path* between $(0, 0)$ and (L, L) will maximize the total cost.
   - You can program some slick binary search method for finding the *maximal breach distance* if you want (it isn't that hard), but for a network this simple you can probably figure this out by hand.
7. An accurate diagram of the Voronoi tessellation for the network. Use dots for nodes and lines for the edges of the Voronoi tessellation. Probably it is best to keep this as a separate plot rather than adding it to one of the previous plots. Using SciPy or Matlab will probably be particularly useful here. The *maximal breach path* should be shown as well.
8. Calculate the *maximal support distance* of the network, travelling from the node closest to coordinate $(0, 0)$ to the node closest to coordinate (L, L).
   - The path followed should go along the edges of the Delaunay triangles.
   - The cost of moving along the edge of a Delaunay triangle is the *maximum distance* any point on that edge and a node (i.e., this is half the length of the edge).

- The *maximal support path* between $(0, 0)$ and (L, L) will minimize the total cost.
- You can program some slick binary search method for finding the *maximal support distance* if you want (it isn't that hard), but for a network this simple you can probably figure this out by hand.

9. An accurate diagram of the Delaunay triangulation of the network. Use dots for nodes and lines for the edges of the Delaunay triangles. Probably it is best to keep this as a separate plot rather than adding it to one of the previous plots. Using SciPy or Matlab will probably be particularly useful here. The *maximal support path* should be shown as well.

Your report should not be excessively long. Probably two pages is sufficient, perhaps only one if your figures are small (which is ok as long as they are clear).

Grading Scheme

Your report will be graded by the following scheme.

10% Statement of the problem. Marks awarded based on how clearly the problem is stated.

10% Description of the connectivity of the network. Marks awarded based on how clearly this is discussed.

10% Description of the coverage of the network. Marks awarded based on how clearly this is discussed.

10% Description and calculation of the maximal breach distance of the network. Marks awarded based on how clearly this is discussed, and whether or not the calculation is correct.

10% Description and calculation of the maximal support distance of the network. Marks awarded based on how clearly this is discussed, and whether or not the calculation is correct.

10% An accurate diagram of the network's connections. Marks awarded based on clarity and accuracy of this diagram.

10% An accurate diagram of the network's coverage. Marks awarded based on clarity and accuracy of this diagram. [*As mentioned, this can be combined in with the connection diagram if you wish, as one diagram worth 20%.*]

10% An accurate diagram of the Voronoi tessellation showing the maximal breach path. Marks awarded based on clarity and accuracy of this diagram.

10% An accurate diagram of the Delaunay triangulation showing the maximal support path. Marks awarded based on clarity and accuracy of this diagram.

10% Writing quality. Marks awarded based on the extent to which your report is written coherently, in paragraph form, with complete sentences. Correct spelling and grammar are also included. Marks may also be deducted if the report is exhaustively long.

Resources

It is quite probable that most of you have more experience with Matlab or Python than I, and consequently may not need any help in this area. The following information is given to help people who have minimal, or no, experience with these programming languages. The methods given below do not necessarily represent the best or most efficient code for the given task, but hopefully it is reasonably clear.

Note: It is not necessary to use SciPy or Matlab to complete this lab. If you have another preferred tool you can use it if it gets the job done. I am recommending SciPy or Matlab here because they are fairly easy to use and free (well, Matlab isn't free but I think you should have free access with your Western ID).

The data file `Graph_`[*your student number*]`.txt` contains a comma-separated list of coordinates for nodes, one on each line. You can add more coordinates to this list if necessary. You can have this file read in to SciPy or Matlab, or you can copy+paste the data right into your script.

SciPy: If you have downloaded the full Anaconda package, you should be able to write and run Python scripts in Spyder. You can run code line-by-line in the console window, but probably it is easiest to write a script in the main window and run it all at once.
Python is a full-fledge programming language, so you need to load the appropriate modules.

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.spatial import Voronoi, voronoi_plot_2d, Delaunay
```

Numpy is "Numeric Python" and provides base functionality for the other modules. Matplotlib is for making plots and figures, and SciPy provides the specific routines that make Voronoi tessellations and Delaunay triangulation really easy.
To load the data points from your file, you can use a handy routine from Numpy:

```
points = np.genfromtxt('Graph_SAMPLE.txt', delimiter = ',')
print(points)
```

Executing a script that contains the above imports and this code to read from the file should dump a list of data points to the console window. (Alternatively, you can enter this code line-by-line into the console window.)
To plot these nodes as points, use `plt` from Matplotlib. This requires a list of x-points and y-points, so we will use slice operators on the 2D array of (x, y) pairs from the data file. The option `'bo'` plots the as blue (b) filled circles (o). Of course, feel free to change the colours and styles as you like.

```
plt.figure(figsize = (5,5))
plt.plot(points[:,0], points[:,1], 'bo')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('x-Coordinate␣(L)')
plt.ylabel('y-Coordinate␣(L)')
```

Here I also set up a square figure (of size 5×5, really the size doesn't matter so much) and make sure the plot limits are $0 \leq x \leq 1$ and $0 \leq y \leq 1$. These lines of code are optional, but make sure that the plot is square and undistorted (there is a tendency for Matplotlib to produce rectangular plots, which is fine, but since the network is a square I like my plots to be undistorted).
This plot will show up in the "Plots" panel on Spyder, usually it is one of the tabs in the upper-right. Once you have a nice plot you can export it:

```
plt.savefig('My_Figure.png')
```

To plot lines showing the connections between these points, you can use the same kind of code.

```
plt.plot([points[4,0], points[7,0]], [points[4,1],points[7,1]], 'g:')
```

This will draw a green dotted line (`'g:'`) between the 5$^{\text{th}}$ and 8$^{\text{th}}$ points in the list (i.e., 4 and 7 starting from 0). There are, of course, other options. I am leaving it up to you to figure this out.

Hint: The distance between points p and q can be calculated by:

```
distance = np.sqrt((points[p][0] - points[q][0])**2
                   + (points[p][1] - points[q][1])**2)
```

Note that Python uses ** for integer exponentials.

SciPy makes determining the Voronoi tessellation and Delaunay triangulation for a sequence of points very easy:

```
vor = Voronoi(points)
tri = Delaunay(points)
```

These return the objects `vor` and `tri` that contain all of the vertices, edges, areas, etc. Plotting the Voronoi tessellation is particularly easy:

```
fig = voronoi_plot_2d(vor)
```

This default plot is acceptable, but you can also make it square and set the x- and y-limits:

```
fig.set_size_inches(6, 6)
plt.xlim([0, 1])
plt.ylim([0, 1])
```

This is a regular Matplotlib plot, so you can draw over it to show the maximal breach path (using `plt.plot(...)`). Alternatively, you can save the figure as an image and use a graphics editor to highlight the edges that correspond to the maximal breach path.
Plotting the Delaunay triangulation is a bit different, but still pretty simple:

```
fig = plt.figure( figsize = (5, 5))
plt.triplot(points[:,0], points[:,1], tri.simplices)
plt.plot(points[:,0], points[:,1], 'ro')
```

The `plt.triplot` part plots the edges of the triangles, and the `plt.plot` part plots the nodes as red filled circles. Again, you can draw over it to shown the maximal support path, or you can save as an image and use a graphics editor to highlight the edges that correspond to the maximal support path.

Matlab: Matlab is rather similar to SciPy. Almost as if SciPy was written as a free, open-source alternative to Matlab! Matlab has all the functionality built-in, so you don't need to load modules as you do in SciPy. To read in the data from the text file and plot:

```
points = readtable('Data␣Files/Graph_SAMPLE.txt');
plot(points.Var1, points.Var2, 'bo')
```

This will create a pop-up window with the plot. Matlab provides more options for modifying plots, so you can make it square, adjust the x- and y-limits, and add axes labels directly in the plot (look for the "Property Inspector" button).
To make a plot of the Voronoi tessellation:

```
vx, vy = voronoi(points.Var1, points.Var2)
```

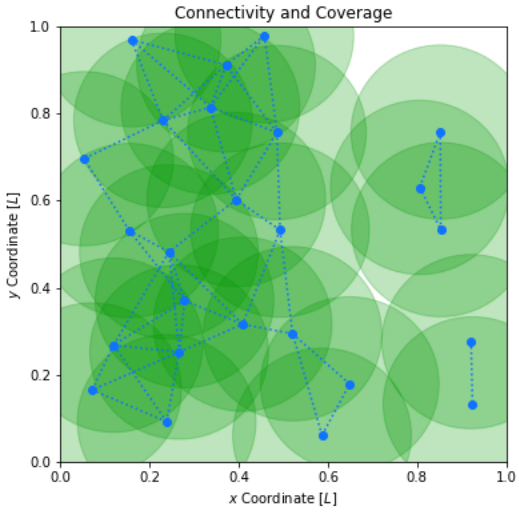
This will create the plot, and `vx, vy` will hold the x- and y-coordinates of the edges in the diagram. These will be needed if you want to use Matlab to calculate and plot the maximal breach distance.
To make a plot of the Delaunay triangulation:

```
tri = delaunay(points.Var1, points.Var2)
triplot(tri, points.Var1, points.Var2)
```

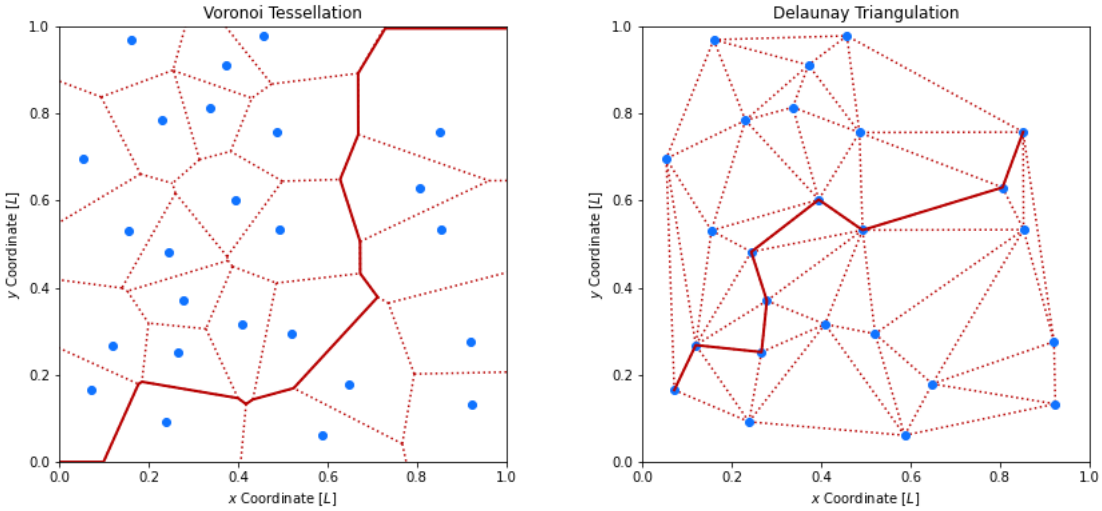
Here `triplot` creates the plot. The variable `tri` has a row for each triangle, and each row has the three indices identifying the nodes from `points` that make the corners of this triangle. These will be needed if you want to use Matlab to calculate and plot the maximal support distance.

Example Plots

Here are some example figures. I used SciPy to make these.



This is a nice combined plot of the connectivity and coverage for a pseudo-random network. This would **not** be acceptable because the network is not fully connected — I need to add extra nodes (at least 2, I think) to get the whole network connected. I used semitransparent circles to help highlight areas with greater or less coverage, this is a nice touch but not necessary. There are a few spots without coverage, but this isn't a problem for the report as long as I discuss it. Of course I could add extra nodes to get full coverage if I want.



Here is the Voronoi tessellation of the network on the left, with the maximal breach path shown in solid red. The Delaunay triangulation of the network is on the right, with the maximal support path shown in solid red. Even though the network is unconnected, we can still draw the maximal support path — it would just have a relatively large cost compared to a fully-connected network.