TMM Lab 02: Network Modelling

RE.Wilson@bristol.ac.uk

03 October 2025

I have broken this worksheet down into three rough learning areas and you should try to sample each in the lab before going into more depth (with the starred subquestions):

- Learning to handle Matlab's graph tool box using square grid 'Manhattan' network examples.
- Generating synthetic urban networks using random planar graph models — for example, beta skeletons.
- Importing real-world map data from OpenStreetMap.

In each case, you can experiment with computing network metrics such as degree distributions, clustering coefficients etc., and compute shortest paths in the networks, and for example, study how shortest path lengths vary if we add or delete edges (build new roads / close old ones).

Getting used to working with graphs: square grid networks

1. Study, understand, and implement the following code snippet for building and plotting a Manhattan-style square grid street network.

   ```
   n = 10; % the number of nodes in each direction - experiment with this
   L = 0.1; % the grid "pitch" in kilometres (for example)
   A = -L*delsq(numgrid('S',n+2));
   G = graph(A,'omitselfloops');
   h = plot(G);
   h.XData = L*floor((0:99)/10);
   h.YData = L*rem((0:99), 10);
   axis equal
   ```

   In particular — inspect `G` and note that the edge weights (effectively lengths) have been set to `L`. Otherwise no especial properties for either edges or nodes have been set. Now to gain familiarity, experiment with some of the methods and functions that Matlab has implemented for working with its graph structure.

   (a) For example - neighbours of a node called either via `G.neighbor(...)` or the object-oriented syntax `neighbor(G,...)`. Likewise, investigate `degree` of different nodes. Make sure it is as you expect. If you type `G.` then tab, you will see a wide range of possible method autocompletions.

   (b) Experiment with shortest paths with `shortestpath`. For example, find the shortest path from node 1 to 100. Note only one of the (many possible equally shortest) paths is returned — as a sequence of nodes. Try it with more return arguments, e.g., `[path,d,edgepath]=` to return both the length of the shortest path, and the path expressed as a list of edges rather than nodes visited. Note the index for an edge refers to its position in the edge structure — e.g., check out `G.edges(171,:)`. Experiment with the use of `highlight` to indicate shortest paths on graph plots.

(c) Write code (e.g., use loops — or research better ways) to compute all the shortest paths between a set of origin nodes and a set of origin nodes — start small, but aim for all 10,000 shortest paths between all pairings of the 100 nodes in the setup with `n`=10. Use `tic` and `toc` to monitor the computational cost. Add up how many times each edge is used in a shortest path. Note under a uniform all-to-all demand scenario, this essentially gives the flow on each link if everyone took the shortest route — except noting that the shortest routes are not unique, so it will be a bit screwed up.

(d)* Develop methods for plotting the 'hotness' of each link.

(e)* Look at the coursework assignment question from 2024/25, which asks you to consider the problem of modifying a square grid network so as to implement Low Traffic Neighbourhoods (LTNs).

2. We are going to perturb the square grid network to make shortest paths more interesting / unique. There are several approaches we can try. (Try these, then run through your shortest path framework from 1(c).)

   (a) Randomise the edge weights a bit. For example, modify each to be a uniform random number between 0 and 2`L`.

   (b)* Perhaps more realistically: perturb the nodes so that they are not exactly at square grid coordinates — e.g., add onto each x and y coordinate a uniform random number between -`L`/2 and +`L`/2. Note — you will need in the above code to compute a new input matrix `A` which reflects that the edge lengths have changed.

   (c) Close some streets? Maybe by using the **`rmedge`** functionality? Just how many streets can you either close or make one way and still have it that every node can be accessed from every other node? (* You could, for example, compute the *minimum spanning tree* for the square grid network and protect the edges which are in it — whilst randomly removing others.)

   (d) It would be really nice to have some plots which distinguish between one- and two-way streets.

   (e)* What do the flows look like on each edge if instead of using uniform all-to-all demand, we use an unconstrained gravity model?

Synthetic beta-skeleton networks

Recall from the lecture that we can use either lune-based or circle-based beta-skeleton networks to create synthetic cities with realistic-looking disordered street connectivity. These planar networks include other well-known graphs such as the Gabriel and Relative Neighbourhood Graphs in their family. The approach is to position nodes (junctions) either randomly or some other way — then to join them up with 'streets' using the beta-skeleton.

3. Download the proximity graphs toolbox from central file exchange (as shown in the lecture) and study / implement the Matlab livescript `BuildBetaSkeletonsExample.mlx` linked there. Hence explore beta-skeletons, either circle- or lune-based, as the value of β is varied. You may want to consider either (i) placing the nodes uniformly randomly as in the exemplar livescript, or (ii) as perturbations of a uniform grid as in Q2 above. In each case — what we are looking for is measures of how the total street length in the the network trades off against statistics of the shortest path length, and of the congestion on each street.

4.* Read the research paper Espinosa, Connors, and Wilson as linked in the lecture. The paper implements a single OD pair but also places the beta-skeleton in a tile that is wrapped in doubly-periodic boundary conditions, i.e., a ring doughnut, so we can model a city with no boundary effects whatsoever. Implement this network model, but load it with uniform all-to-all demand with shortest routes. Study how the various metrics vary with β.

Working with real-world OpenStreetMap data

As we described in the lecture, OpenStreetMap (OSM) is an amazing source of free data — albeit probably not as sophisticated and detailed as commercial mapping sources (which cost lots of money). No kidding: Python tool-chains are much better for working with OSM. However — in line with the unit policy, we are trying to stay substantially Matlab-based.

5. Use the Matlab mapping toolbox function `readgeotable` to read the provided Bristol OSM file and plot it. Note that unfortunately, the mapping toolbox does not yet appear to preserve the logical relationships between 'nodes' and 'ways' and so we cannot use it to cast OSM data into a network structure.

6. Explore sources of alternate OSM files (and their faster binary equivalent format PBF) and experiment with loading them in.

7. Install and experiment with command line tools for simple manipulations of OSM data (such as `osmium` — which also has associated Python packages).

8. Study the provided livescript code. This takes the Bristol OSM file and uses the Matlab function `readstruct` to extract it — NB a very slow operation — and then further simplify matters to produce tables of 'nodes' and 'ways'. As some of the steps are extremely slow — I have also provided some product data files in `.mat` format which you can load directly into matlab to explore.

9.* Consider how to turn these nodes and ways tables into an adjacency matrix that can be loaded into Matlab's graph structure and which we can then operate on. Note that we only want junctions between different ways to be nodes in the new network — that is nodes which are simply tracing out the curve of a way should not be retained as nodes in the new network. However, it would be nice to calculate the arc length of each way between junctions.

10.* Consider games such as finding shortest paths and seeing how they change as some edges are deleted — or indeed as extra edges are added.

As a special treat at this point Wentao might show you how to use Python to prepare intermediate files for you to import to Matlab and then analyse.

11.* Explore whether beta skeletons are good models for the street networks of each of these cities. To do this for a given network, extract the nodes and compute the edge list for the beta skeleton for a given value of β. What proportion of the edges present in the beta skeleton are also edges in the real network, and vice versa? Scan over β to determine for what value the fit between the skeleton and the real world network is best. Do different cities have significantly different optimal values of β?

Where are we going with all this? Here are the sorts of capabilities a fun (i.e., challenging) coursework assignment might test.

- Take some map snippet for which there is also demand data (eg from the census).
- Allocate demand to the map — note there will need to be tricks because e.g., an MSOA is quite large compared to the street network topology. So you will need to sort-of randomly spread out the demand over an MSOA.
- Compute shortest routes for all the required trips.
- Study how the network is loaded — which edges are busy etc
- Consider if the network is in equilibrium (it won't be if there is much traffic) — i.e., if we modelled the congestion caused by the traffic loaded on the network, where would the new shortest routes be? This leads to next weeks topic — Assignment modelling.