TFL Random Walk Model

Jacob Delveaux

June 26, 2020

1 Review

Over the last several weeks we have investigated the TFL transport network in a random walk model with simple dynamics. This was an effort to see which features of the real world transport network were captured by just the most basic functioning. This short text is an outline of how the model was made, its strengths, and its limitations.

2 Data

The data in this model differs somewhat from the data used in the stochastic model.

2.1 The Graph

In this model, we have devised a new way to create the TFL tube / overground graph using the unified API (from api.tfl.gov.uk) (API_graph.py). Here are the steps involved:

1. Create functions for data requests: get lines from travel mode, get routes from line, and get stoppoint from id.
2. For each mode (tube, dlr, overground, tflrail), get lines in that mode (i.e. Picadilly line)
3. For each line, get its routes (i.e. Piccadilly has Cockfosters -> Uxbridge, or Cockfosters -> Heathrow)

4. For each route, add all stations inside of the London area (checked by simple function) to the graph and add connections (with distance attribute) between them.

Which outputs a python NetworkX diGraph containing all stations from all lines connected to one aother.

There are two problems with the above approach. The first problem is that the programme itself makes more than 500 req/min to the TFL server, which is over its service limit (getting names and locations for all stoppoints), so the programme must be throttled manually by a 'wait' function. Second, national rail data is not included. TFL requests for national rail data extracts lines from all across the UK, creating pages and pages of stations to be sorted through. We are currently investigating a way to pull data within a certain GPS boundary, which would alleviate the issue.

After the graph is built, it undergoes further processing (build_API_graph.py). First, stations which are walk able to one another (within 0.5km) are added with a distance "penalty" to account for the fact that walking routes are often not straight lines (i.e. two stations with a walking distance of 0.5km which were previously unconnected are now connected with an edge of weight 1km). Second, duplicate stations are found and removed. For example, Stratford station appears in the overground, tube, and dlr modes, which each have their own unique ID for the station ('940GZZLUSTD', '940GZZDLSTD', '910GSTFD'). These three ids (nodes) with their associated links are merged into a single id (node). Third, National Location Codes (NLCs) are added to the stations, as most TFL movement data (oyster, statistics) are linked to the station NLC rather than its NAPTAN id. Finally, any self-loops (artifacts from the removal of duplicate stations) are removed.

In the end, we are left with a diGraph with 382 nodes (stations) with their colloquial name, GPS coordinates, and NLC code; and 940 edges (links), with the distance between them.

3 The Random Walk Model

Description of (random_walk.py)

3.1 Data

There are only two pieces of data we need to run the random walk model. The first is the tube graph, as described in the prior section. The second is the link load obtained from the 2018 MTT (Monday-through-Thursday) Link Load data file from TFL NUMBAT. This file contains the number of passengers travelling (on average) from station A to station B (i.e. Oxford Circus to Warren Street is 203606 total, 37232 of those during the AM peak and 65989 during the PM peak).

The pre-processing stage involves extracting the Link Load data and adding it as an attribute (weight) to the edges, then removing the edges that had no associated weight in the data file (left with 381 nodes and 852 edges).

3.2 The Walk

The random walk model we have implemented is very simple, relying only on the transition matrix. The general procedure is as follows:

1. The transition matrix is obtained (described below)
2. The walker is given a random initial location
3. At each time-step, the walker makes a move.
   (a) Random choice weighed by the transition matrix. E.g. At Holborn, 25% chance to go to Chancery Lane, 40% to go to Covent Garden, 25% to go to Green Park, and 10% chance to go to Tottenham Court Road; selected randomly.
4. After each move, record the location
5. Continue to make moves until the step limit (predefined) has been reached

3.2.1 Transition Matrix

The transition matrix is obtained via $T = D^{-1}A$ where D is the diagonal degree matrix and A is the (weighted) adjacency matrix. The adjacency matrix weight is defaulted to the link load, but can also be the distance between stations or random weights. By weighted, we mean that wherever a '1' would be in the original adjacency matrix (indicating a link between i and j), there is instead a floating point number representing the weight of that edge.

3.3 Recorded Data

From each run of the random walk model, we obtain three data files.

1. The equilibrium probability (occupation probability for nodes (stations))
2. The mean first passage times (MFPTs) for each node pair
3. The edge counts, or, number of times the agent passed through the given edge during the simulation

The script file_io.py helps to save runs without overwriting older data (and load into post-processing scripts).

4 Results

Below are the currently obtained results from this model.

4.1 Visualizations

There are currently (4) ways of visualizing the results obtained from the model.

1. Colored edges
2. Colored nodes
3. Spatial-denisty plots
4. Edge-count histograms

The edge count data, weights, node properties, and other networkx measures to produce images that distinguish the spatial features of the tube graph simulation results. print_graph.py contains the methods for coloured edge plots, coloured node plots, and spatial density plots. visualize_results.py (discussed below) contains the method for generating edge-count histograms.

4.2 Comparison with TFL data

Currently, there are (2) methods of comparing simulated random-walk data to TFL data.

1. Spatial density plot distributions

2. Edge-count densities

4.2.1 Edge-count densities

Edge-count density plots are comparisons between the edge-counts obtained via the random walk model and the TFL Link-Load (2018 MTT) NUMBAT data. The histograms show the density of edges with a certain number of travellers along that edge. i.e. 100 edges have 1000 passengers travelling across it, 10 links have 100000 passengers, etc.

For some reason, I have not yet bothered to fit the edge-count distribution to any sort of empirical distribution (be it power law, exponential, etc.).

Anywho, the process is simple. First the numbers of travellers along each edge (station-to-station link) is placed into an array (data1 for TFL data, data2 for simulated data). These arrays are normalized so that the largest value for each is unity i.e. the histogram shows the density of edges (y-axis) that have this fraction of the largest edge-count (x-axis) as their edge-count.

4.3 Hub Identification

An important part of disaster analysis in a network is the identification of hubs. The most common measure is betweenness centrality, or the number of shortest paths through a given node. In hub_identifcation.py, we consider (4) measures to identify hubs:

1. Betweenness centrality

2. Occupation probability / RW stead-state distribution

3. Entrance probability (from TFL data)

4. Swap probability (from TFL data)

Whenever permitted, we use (4) different edge weights for different calculations.

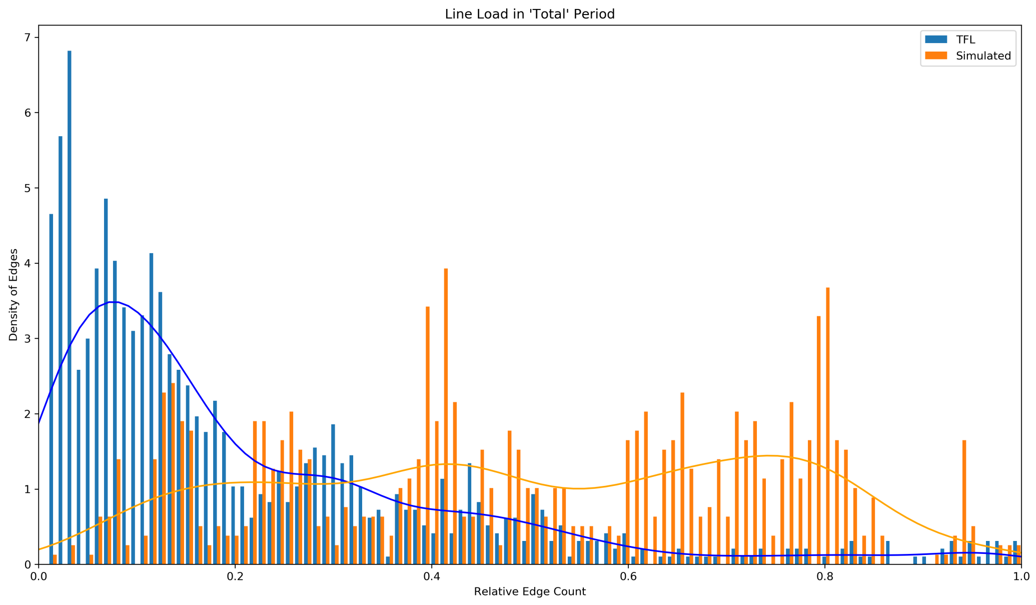

Figure 1: Edge-count histogram for Random-Walk model and TFL data

1. Unweighted
2. Weighted by distance between stations
3. Weighted by Link Load data (TFL, NUMBAT MTT 2018)
4. Weighted by random weights

4.3.1 Occupation probability

By calculating $p = Tp_{old}$ with transition matrix T and probability vector p until convergence of p and p_{old} (starting from a random state) we obtain the steady-state probability distribution or 'probability for an agent to be occupying a station'/'occupation probability'.

4.3.2 Entrance probability

Entrance probabilities are determined via the TFL NUMBAT MTT 2018 Station Link Flows dataset. This data provides the number of entrances/exits into a station (on average). The entrance probability is obtained by dividing

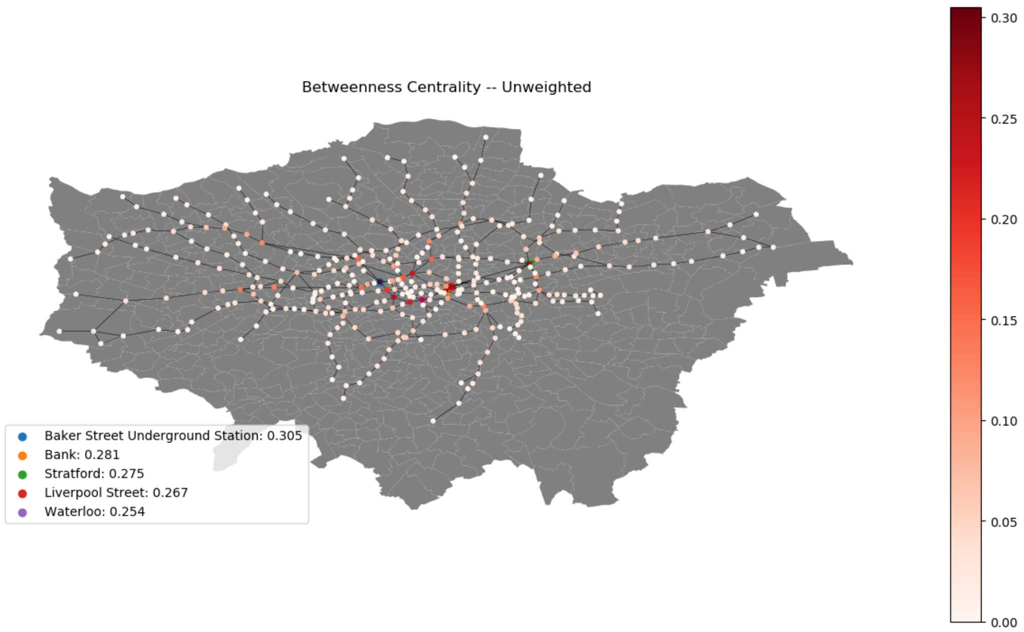
Betweenness Centrality -- Unweighted
● Baker Street Underground Station: 0.305
● Bank: 0.281
● Stratford: 0.275
● Liverpool Street: 0.267
● Waterloo: 0.254
0.30
0.25
0.20
0.15
0.10
0.05
0.00

Figure 2: Hub Identification: Betweenness Centrality

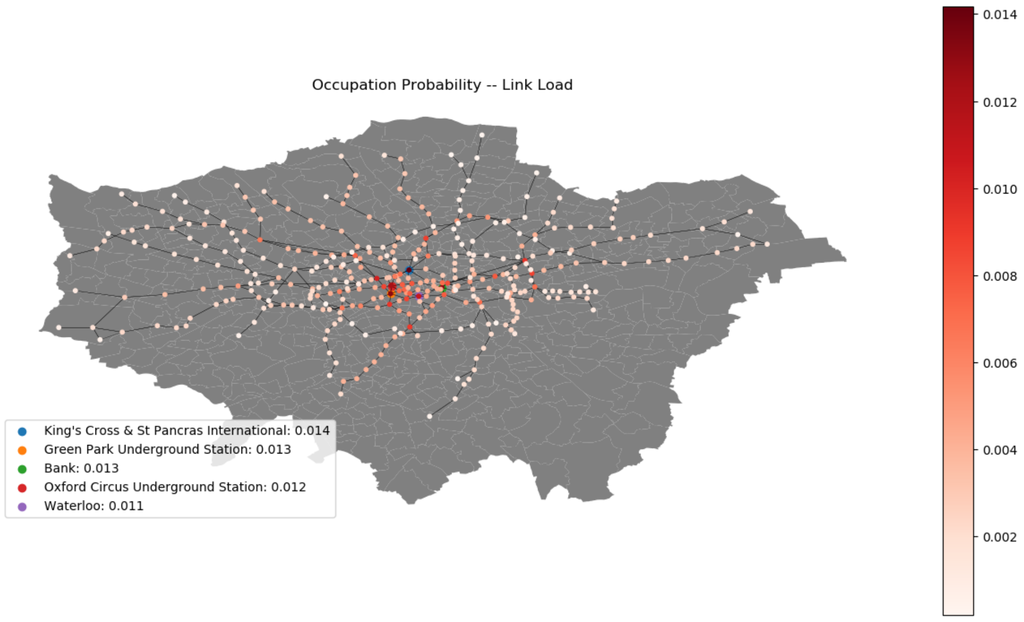
Occupation Probability -- Link Load
● King's Cross & St Pancras International: 0.014
● Green Park Underground Station: 0.013
● Bank: 0.013
● Oxford Circus Underground Station: 0.012
● Waterloo: 0.011
0.014
0.012
0.010
0.008
0.006
0.004
0.002

Figure 3: Hub Identification: Occupation Probability

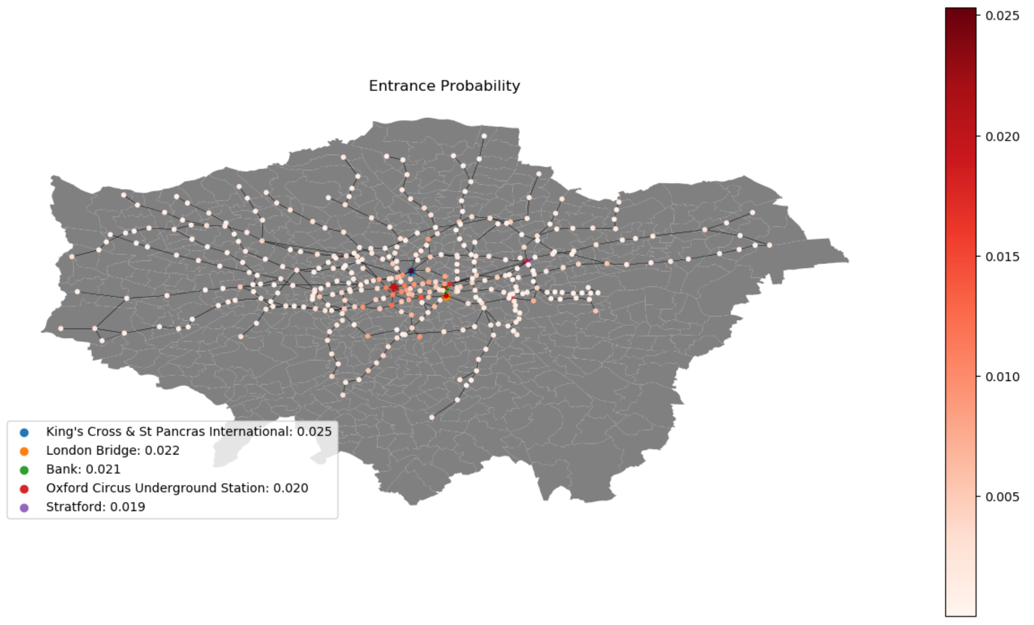

Figure 4: Hub Identification: Entrance probability

the number of entrances into a station by the total number of entrances across the network (during a particular time period). We have opted to just use the time period 'Total', which includes all data from all time periods.

4.3.3 Swap probability

From the TFL NUMBAT MTT 2018 OD (Orgin-Destination) matrix we determined the shortest route from every origin to every destination via dijkstra's algorithm. The swap probability is defined as the number of times a path included a swap divided by the total number of paths.

This is actually a terrible measure. It does not give the probability of swapping at a particular station (a measure of how many people actually need to get out and walk around the station) but rather is the probability that, by starting at this station, you will need to swap at some point during your journey. This measure is biased towards stations with a small number of lines and/or those on shorter lines with less stops overall.

A better measure would be to find the number of passengers that swam AT THIS STATION, i.e. a measure of at which station the swap is actually

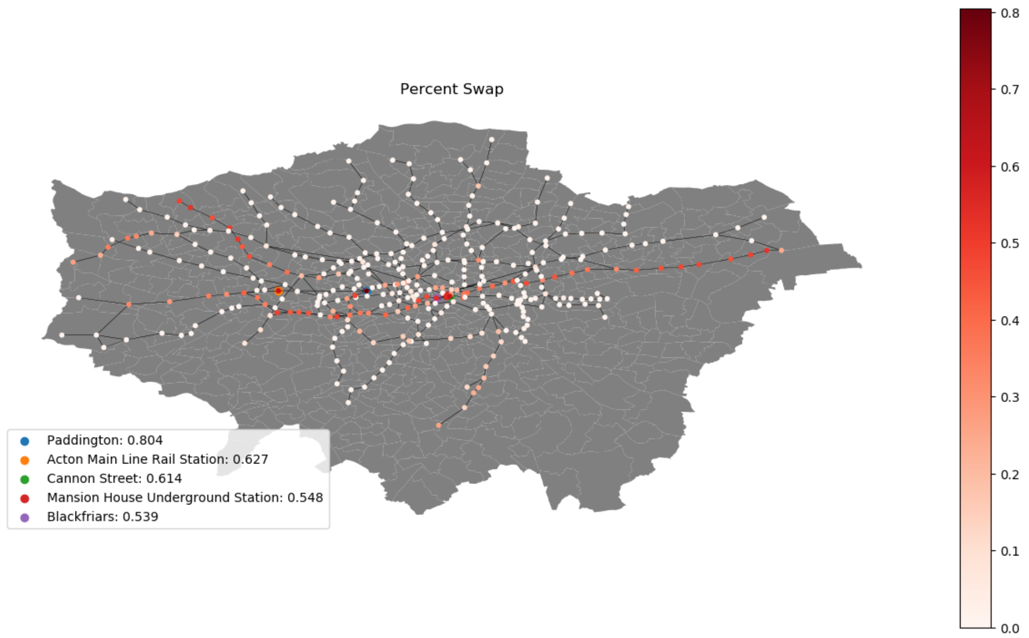

Figure 5: Hub Identification: Percent Swap

occurring, rather than where it originated from.