Experimenting with Spatial Networks

To save time when experimenting, it can be useful to create a subset of the airport dataset, for instance keeping only the airports of larger degree. Note also that when computing deterrence function, it is sometimes useful to work with random samples instead of computing distances between all pairs of nodes. This dataset is small enough however to compute all values in less than 1 minute on an average personal computer.

To the best of my knowledge, there is no good library to work with spatial networks. `networkx` has functions to generate random spatial graphs (called geometric graphs https://networkx.org/documentation/stable/reference/generators.html), but they are not designed to fit deterrence functions, and are not adapted to work with geographical coordinates.

1. Deterrence Function on the airport dataset

   When studying a network with spatial information, a first step to check if it can indeed be considered as a spatial network is to compute its deterrence function.

   (a) We will need to discretize distances using bins(similar to a histogram). Choose b bins, for instance every 500 km from 0km to 10,000km. Write a function which, provided a list of distances, return a list with b values, the number of values in each bin. A convenient way is to use `numpy.digitize` and `collections.Counter`.

   (b) Write a function which compute the distance between two positions on Earth in km. You can use function `haversine` from the package of the same name(`pip install haversine`). To obtain standard latitude and longitude from the data in the airport dataset, you need to divide latitude and longitude values by 3600.

   (c) From the airport dataset, compute the distribution of distances of edges. Plot it.

   (d) Compute the deterrence function. You need to take into account the position of nodes, i.e., to normalize the observed fraction of edges at a given distance by the fraction of pairs of nodes at this distance.

2. Soft RGG

   (a) Generate an ER random graph with the same nodes and the same number of edges as the original graph. Plot this graph using nodes longitude and latitude to position nodes. Compare visually with the original network (You can use networkx `draw_networkx` function using the *pos* parameter, or Gephi using the Geo Layout plug-in).

   (b) Check that the deterrence function of this graph is flat

   (c) Using the deterrence function that you computed prviously, generate a Soft RGG with the same nodes, the same number of edges, and the same deterrence function as the original graph. You need to compute the probability of observing an edge for each pair of nodes according to its distance, and you can then pick the desired number of edges with the desired bias using `np.random.choice` (check the option).

   (d) Check that the generated graph has the same number of edges and the same deterrence function as the original one.

   (e) Plot your graph and compare visually with the previous ones.

(f) Compare the clustering coefficient, the average shortest path, and the degree distribution of your network compared with the original and the ER ones.

(g) By subtracting from the observed network the probability of observing each edge according to the null model, and keeping only edges with a value above 0, you can create a new network in which the weight of edges define how exceptional they are. Are there nodes with more exceptional edges than others?

3. Going further: Gravity Model

(a) Compute a randomized version of the network using the Gravity Model. Don't forget to check that the resulting network has all the desired properties: same node positions, (approximately) similar degrees, (approximately) similar deterrence function.

(b) Compare network properties with the previous networks

(c) Using the same strategy of subtracting the model from the observed network, you can now obtain a much richer view of your network: to each pair of node is associated a value that can be interpreted as the exceptional character of its existence of non-existence. What are the most surprising missing edges, and most suprising edges?