Exercise 10: Networks 2

Kristian Urup Olesen Larsen[1]

[1]kuol@econ.ku.dk

April 21, 2020

Problem 10.1.1. - Null models

To describe networks in a statistically rigorous way we need a comparison network - this is exactly like in the regular econometrics where we often compare parameter estimates $\hat{\beta}$ to the exact value 0 and ask if we can confidently say that our data has $\beta \neq 0$. The null of $\beta = 0$ is equivalent to assuming that y and x are unrelated, i.e. y and x are relatively random. In networks the relevant comparison is more complicated but essentially the same thing. A null model is a way to permute an original network to form random comparison networks. Comparing to a null model can then teach us if specific features of our network are significantly non-random. Unlike in linear regression there are many ways in which a network can be random and the exact choice of null model must match the kinds of questions we want to answer.

To measure if the diameter of a network is statisticially different from random under a null with random degree-preserving edge swaps you would compute the difference in diameter between your real network and the diameter in k permuted null models. If this difference is significant our network has a significantly non-random diameter.

Problem 10.1.2. - Degree preserving edge swaps

The double edge swap preserves node degrees by matching pairs of nodes when edges are swapped, so that all nodes that loose an edge also gains a new one. This will render a network in which each node has the same degree as the original, but where edges are randomly shuffled. Note that sometimes a swap is not possible, e.g. if $u - v$, $x - y$ and $u - x$. In this case swapping $u - v$ and $x - y$ will remove an edge. To avoid getting stuck in these cases the algorithm will try a maximum of `max_tries` times before terminating.

Problem 10.1.3. - Facebook data

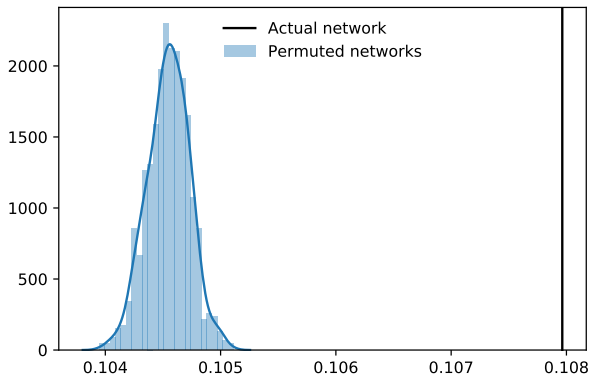

Fig 1. Average clustering coefficients in permuted networks.

The first code for this question can be taken directly from last weeks exercises. This time create an undirected graph instead of the directed one we made last time. After loading the network data I create a networkx graph G. The ALCC of the real facebook network can then be computed via

```
truth = nx.average_clustering(G)
```

which comes out to be ≈ 0.108. Remember that the edge swaps are made in place so G will change as you permute the edges and you will have to reload the data to get back this result unless you copy G. To keep things structured let us begin by writing a simple worker function that does all of the heavy computations for us

```
def worker(G):
    G_ = G.copy()
    G_ = nx.double_edge_swap(G_, nswap=1000, max_tries = 5000)
    return nx.average_clustering(G_)
```

We can then easily write a loop that calls this worker 1000 times. To keep our impatience at bay we can use tqdm to add a progress bar to the loop.

```
from tqdm import tqdm
alcc = []
for _ in tqdm(range(1000)):
    alcc.append(worker(G))
```

Notice we could have written this code as a list-comprehension, but in this case running worker takes far longer than managing the loop. For that reason a list-comprehension will take almost as long time to run as the loop. Once the loop completes we can compute the p-value by computing the share of permuted networks that has an ALCC at least as large as the original network. This should be straight forward to do, and looking at figure 1 clearly suggests that clustering in the facebook network is higher than expected under random links.