CSN significance of metrics

Pietro Fronte, Carolina Centeio Jorge

October 2018

1 Introduction

In this assignment we are asked to investigate the significance of some metrics of interest related to networks. In particular we are asked to investigate one of these possibilities:

1. (Global/Local) Clustering measure
2. Closeness measure

We chose to develop the Closeness measure for the proposed set of graph. This value can be seen as a measure of importance of each vertex; however the value we are going to consider is the Mean closeness centrality, the average value of all the vertices in the given graph.

$$\mathcal{C}_i = \frac{1}{N-1} \sum_{j=1(i \neq j)}^{N} \frac{1}{d_{i,j}}$$

As suggested, we have implemented everything in C++.

2 Results

We were able to get results only for few given graphs. All of the p-value were 0 even not expected (at least not for all). Our code, due to our implementation, was not able to read graphs bigger than the Greek one. For bigger ones we got repeatedly the same error "std::bad_all in memory position" not allowing us to get any kind of value.

3 Discussion

In this task the null hypothesis are two different random graph obtained starting from the same real network. The ER random graph try to keep the structure of the original network generating a graph having same number of nodes and edges as the original one. However there is a fundamental aspect that is not

Language	N	E	$\langle k \rangle$	δ
Arabic	21532	68767	6.3874	0.0002966
Basque	12207	25558	4.19	0.0003406
Catalan	36865	197318	10.7049	0.0002903
Chinese	40298	181081	8.9870	0.0002230
Czech	69303	257295	7.4252	0.0001071
English	29634	193186	13.0381	0.00044
Greek	13283	43974	6.6210	0.0004985
Hungarian	36126	106716	5.9079	0.0001635
Italian	14726	56042	7.6113	0.0005168
Turkish	20409	45642	4.4727	0.0002191

Language	*Metric*	p-value (binomial)	p-value (switching)
Arabic	Closeness Centrality	/	/
Basque	Closeness Centrality	0	0
Catalan	Closeness Centrality	/	/
Chinese	Closeness Centrality	/	/
Czech	Closeness Centrality	/	/
English	Closeness Centrality	/	/
Greek	Closeness Centrality	0	0
Hungarian	Closeness Centrality	/	/
Italian	Closeness Centrality	0	0
Catalan	Closeness Centrality	/	/
Turkish	Closeness Centrality	0	0

considered: the ER graph doesn't keep track of the degree sequence the original graph has. In this way we are ignoring for example important or rare words that appears in the original network assuming a binomial degree distribution. These aspect is instead considered when building the Switching Model. Looking at our results however we are getting almost everywhere a p-value of 0. This means that, since we are evaluating the

$$p(x_{NH} > x) = \frac{f(x_{NH} > x)}{T} = \frac{\mathbb{1}_{\{\mathbb{x}_{\mathbb{NH}} > \mathbb{x}\}}}{T}$$

with T = number of simulations, we didn't get any randomized version of the real network showing an higher value of closeness centrality measure. We are aware that the number of simulation might be not big enough (21 simulation) but in spite of that we didn't get any interesting results.

However thinking about what we can expect we can formulate some hypotheses even if they are not confirmed by our results. As said before the switching model should be a randomized form of our original graph that keep the number of nodes, number of edges and preserve the degree sequence; for this reason what we can expect is a very close graph with similar closeness value to the original one. If we add noise due to the randomization process we expect closeness value to be around the original value (sometimes greater than the original sometimes smaller) and so $\{\mathbb{1}_{\{\mathbb{x}_{\mathbb{NH}} > \mathbb{x}\}}\} \neq 0$.

On the other hand ER is a random graph that doesn't take care of the degree sequence and generate whose nodes lose their original importance (node degree). For this reason we can expect hubs (important nodes in real network) becoming common nodes in ER losing their centrality and for that a generally lower value of closeness centrality. $\{\mathbb{1}_{\{\mathbb{x}_{\mathbb{NH}} > \mathbb{x}\}}\} \simeq 0$

4 Methods

4.1 Graph Structure

To structure the graph, we created a C++ struct called **Graph**. This structure stores the number of nodes, vertices and an adjacency matrix (vector of different sized vectors) indicating the edges and a map to store the names of the nodes and corresponding id (and position in adjacency matrix). It also had a distance matrix, that we will explain later in this section, and a boolean parameter indicating whether it is to make use of the distance matrix or not.

4.2 Geodesic distance

The geodesic distance algorithm is based on a Breadth-First Search[1] (and makes use of a vector of distances between the starting node to all others). This vector,

[1] adapted from https://www.geeksforgeeks.org/minimum-number-of-edges-between-two-vertices-of-a-graph/

at the end of the algorithm, will contain the distance between the starting node and at least one another. Thus, we believe it is important to store all these calculated distances so that we will not have to compute them if needed in the future. This will improve the global time of our algorithm. So, our graph structure has the distance matrix introduced above. We can then check, at the beginning of geodesic distance algorithm for nodes u and v, if $d_{u,v}$ or $d_{v,u}$ is already assigned and skip the computation part. Plus, we check if one of the nodes has degree bigger than one ($k > 1$). If not, we simply increment the distance of its only neighbor with the other node. In case the degree is 0, the closeness will automatically be 0 and we will not compute geodesic distance.

4.3 Ordering

We tried four ways of ordering the edges: as they are disposed in the file, randomly, crescent order regarding the degree of the nodes and decrescent order. We realized it is better to use crescent order, since it may be faster to converge when calculating the mean closeness (specially with bounds). The mean closeness is the sum of several closeness. If we start by computing the closeness of the nodes with higher degree, we can understand that it will increase a lot at the beginning and from there, every time a bit less. So, when bounding, it may find them faster at the beginning.

4.4 Add Edge function

The function *addEdge* tries to add an Edge to a graph. However, if it will make a loop or a multiedge, the edge will not be added and will return false.

4.5 Null hipothesis

4.5.1 Erdos-Renyi

The algorithm to create an Erdos-Reyi graph was developed in C++., creating the same number of nodes as the real network and randomly creating E possible edges (being E the number of edges in the real network).

4.5.2 Switching Model

We also developed the switching model in C++. We randomly choose two edges. If it is not possible to make to new switching the ends of those edges (addEdge returned false), we continue, increasing the counter but leaving the graph as it was at the beginning the iteration. All nodes will then preserve their degree. Q will be log(E), as for coupon collector's problem) and so QE will be (E*log(E)).

4.6 Experiences

We will run $T = 21$ ER graphs and Switching models (two different null hypothesis) and compare their closeness with the Real Network closeness. At first, we

would use a Monte Carlo simulation and find an approximation for the value of the real network closeness. However, we found this optimization unnecessary, at the end.

To compare the closeness of the null hypothesis, we use a lower and upper bound method. We will set a lower bound, at the beginning, as the closeness of 10% of N (total number of nodes) nodes. The upper bound will then be the lower bound plus closeness of one for the rest of the nodes (in the first iteration 0.9*N). At each iteration, the lower bound will increase 10% of N until it reaches N (worst case).