資工四 B00902064 宋昊恩

SNA Hw0

1. Part A:

(a) partA_egofb

I. $\alpha = 1.31533381688$ (directly use min degree of nodes)

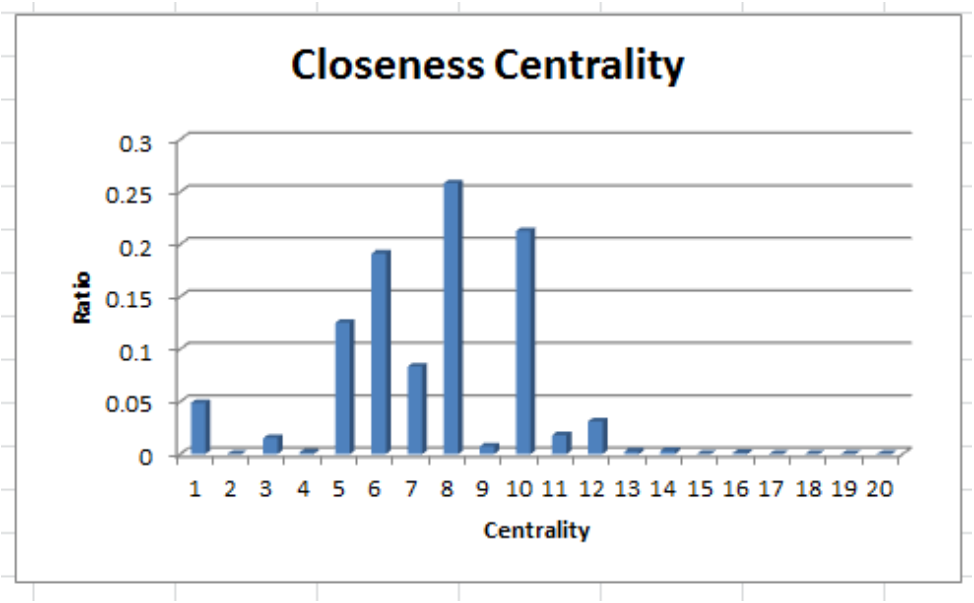
II. Pair rate: 1.00000000000; Average shortest path length = 3.6925068497

III. Closeness centrality:

i. Top 10 nodes:

NodeID	107	58	428	563	1684	171	348	483	414	376
Centrality	0.459699	0.397402	0.394837	0.393913	0.393606	0.370493	0.369916	0.369848	0.369543	0.366558

ii. Distribution graph:



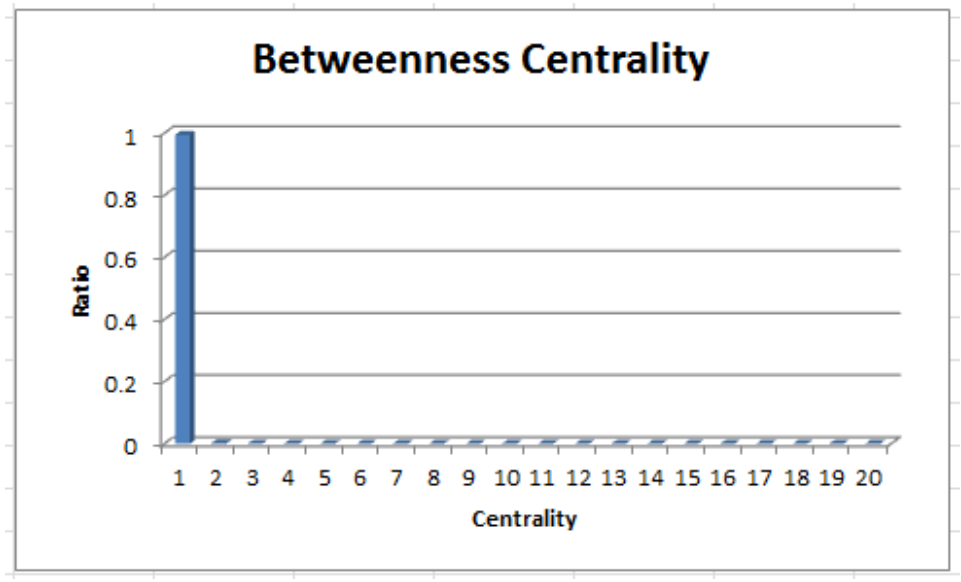
iii. It can be observed from the graph that it does not follow power law distribution.

IV. Betweenness centrality:

i. Top 10 nodes:

NodeID	107	1684	3437	1912	1085	0	698	567	58	428
Centrality	0.480518	0.337797	0.236115	0.229295	0.149015	0.146306	0.11533	0.09631	0.08436	0.064309

ii. Distribution graph:



iii. It can be observed from the graph that it follows power law distribution.

(b) partA_hepth

I. $\alpha = 1.38962352061$ (directly use min degree of nodes)

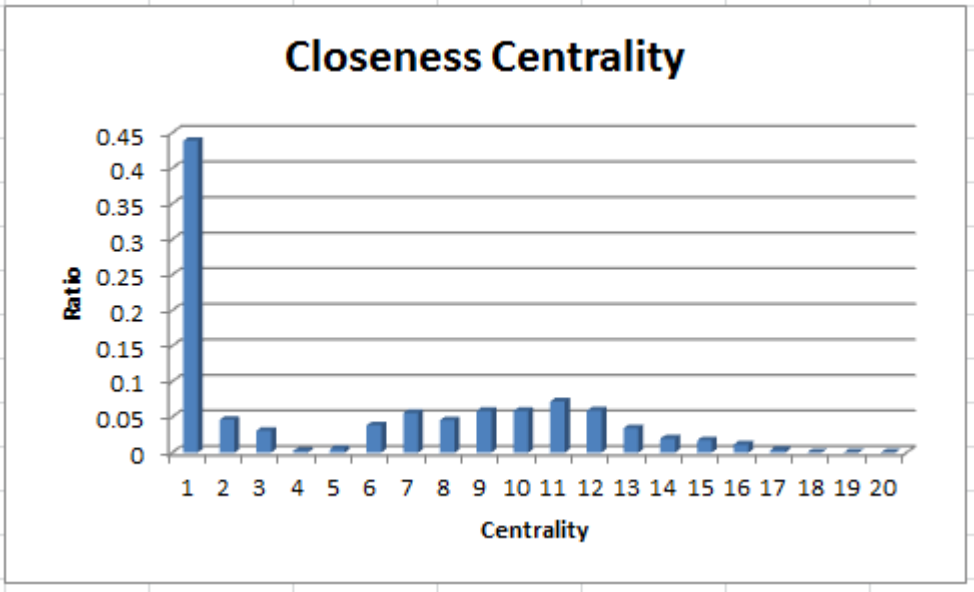
II. Pair rate: 0.299196841858; Average shortest path length = 8.45911497021

III. Closeness centrality:

i. Top 10 nodes:

NodeID	210224	210292	211178	303185	210157	211181	304180	302137	209241	303214
Centrality	0.159061	0.14491	0.143795	0.142011	0.141963	0.140778	0.139355	0.139102	0.138797	0.138255

ii. Distribution graph:



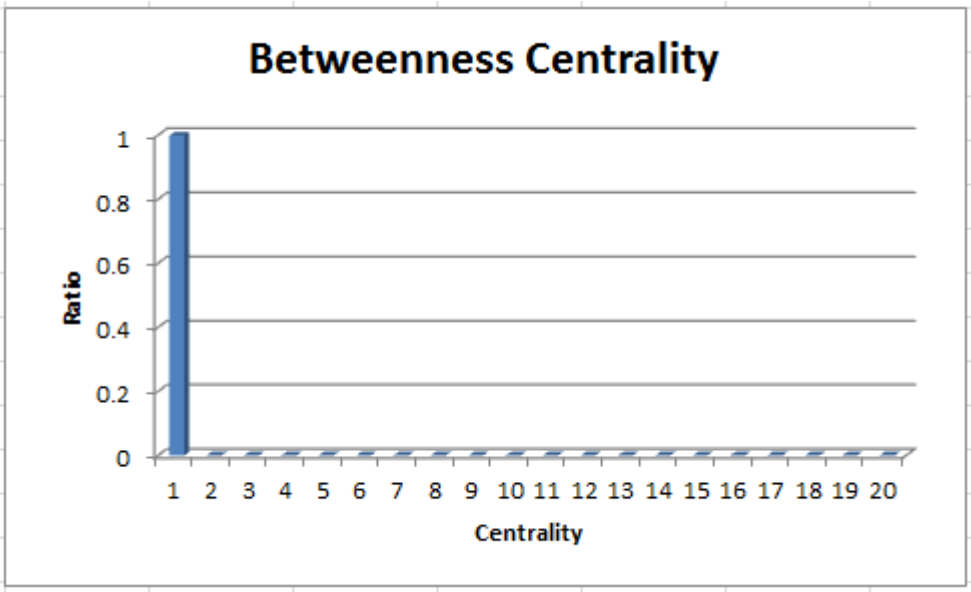
iii. It can be observed from the graph that it almost follows power law distribution.

IV. Betweenness centrality:

i. Top 10 nodes:

NodeID	9905111	9810008	206223	9509140	9803001	9912210	9902121	9607239	206182	9907085
Centrality	0.110153	0.091944	0.091876	0.04576	0.03734	0.028428	0.024149	0.023812	0.018984	0.017359

ii. Distribution graph:



iii. It can be observed from the graph that it follows power law distribution.

2. Part B:

(a) IC model:

I. Number of activated nodes is 870.378 in the average of 10000 tests
(To speed up the program, I applied the multiprocess mechanism in python, which saves considerable of time. Details as following: 3659.900u 2.164s 3:54.96 1558.5%)

3. Implementation:

(a) partA_egofb and partB_egofb:

The alpha value can be calculated based on the formula provided in the slides. I calculated x_{min} value by simply finding the smallest x among the whole give data.

Finding the largest connected component in undirected graph or the largest weekly connected component in directed graph is done with the networkx API.

In undirected graph, the number of pairs is obviously 1.0 according to the undirected graph property, and it is also an API for calculating average shortest path.

However, in directed graph, directly using the API will get a wrong answer because that networkx.average_shortest_path_length() will simply let the distance of non-reachable pairs to be zero.

There is also API from networkx for calculating centrality. After obtaining the answer, I use Microsoft Office Excel to draw the corresponding statistic graphs for me.

(b) IC model

I implement the Independent Cascade Model simply following the algorithm statement.

However, running model for 10000 times to get the average performance is too time-consuming for me. Thus, I study the multi-process mechanism in python and increase the calculation significantly.

It is worth noting that at first, I follow the pseudo code searched by Google, and I get the average answer is 1091.7923, which is larger than other classmates' ones. After reviewing and discussing with others, I found that provided pseudo code is just wrong! In that pseudo code, it tags node "visited" when it is popped out from queue, however, the correct one is to add the tag when pushed into queue.

(c) LT model

The implementation of Linear Threshold Model is quite easy. I spend some time on studying the networkx API to save further information on nodes.

4. Source codes:

(a) cal_undirect.py: for partA_egofb.txt

(b) cal_direct.py: for partA_hepth.txt

(c) ic_model.py: ic model implementation

(d) lt_model.py: lt model implementation