EE232E Graphs and Network Flows

Spring 2017_HW3

Team Member:
Xiongfeng Hu (304753117)
Younan Liang (504759929)
Dui Lin (504759948)

1. Introduction

In the third assignment, we will continue to explore and analyze a real network with R libraries “igraph” and “netrw”. Particularly, we are going to study the properties of this network such as connectivity, degree distribution, and different ways to compute community of weighted network, and its largest community, sub-community, overlapped community detection. This project serves as a good review of all previous projects and gives a good view of the real stuffs in our life. The data is available on [1]. Please note that the data is given as a directed edge list format, where each line has three items: node1, node2 and the weight of the edge from node1 to node2.

2. Problem Solutions

2.1 Generate giant connected component

We use the provided network edges list to generate the corresponding directed graph, we found that this graph is not connected (is.connected(g) = FALSE). But the giant connected component of this graph is relatively large which comprises 10487 nodes out of 10501 nodes, as shown in the Figure 2.1.1.

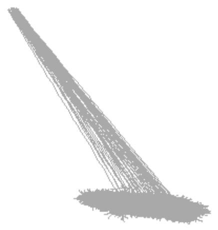


Figure 2.1.1 Giant connected component of the graph

2.2 Measure the degree distribution

Since the graph is directed, the in-degree and out-degree distribution of its giant connected component could be calculated and plotted separately. Figure 2.2.1 and Figure 2.2.2 show the in-degree and out-degree distribution of GCC in a normalized format. It is obvious that with the increase of degree, the percentage of corresponding nodes decrease fast, and most of nodes have the degree between 0 to 2000.

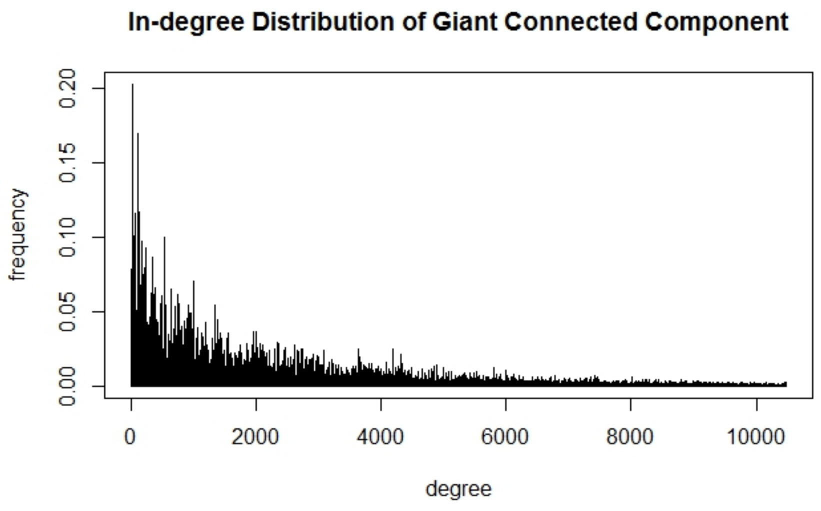


Figure 2.2.1 In-degree distribution

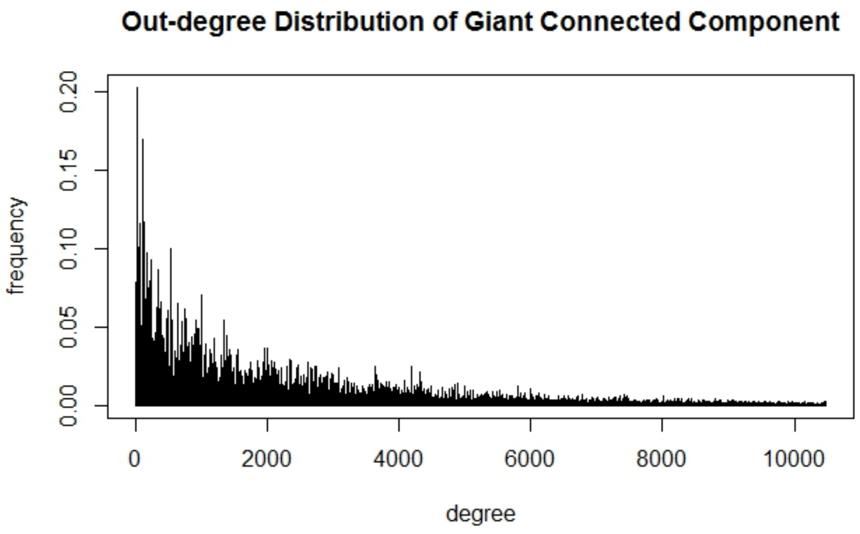


Figure 2.2.2 Out-degree distribution

2.3 Community Structure

Since this network has edge weights, it is not trivial to convert it from directed to undirected, especially if there are two directed edges between node i and node j. We at least have two options.

Option 1- Remove the direction without changing number of edges

In option 1, we keep the number of edges unchanged, and just remove the directions. The resulting undirected network is not simple. label.propagation.community command is used to compute a weighted, undirected, non-simple network's community structure.

Label Propagation Community

Since the resulting graph was simple label.propagation.community was used to obtain the communities of the network. The modularity and sizes of the community structure obtained were as follows,

Modularity of Community = 0.0001290506.

Table 2.3.1 Community Structure using Label Propagation

Community	1	2	3	4	5
Number of Nodes	10475	4	3	3	5

Option 2 - Merge two directed edges between nodes

In option 2, we merge the two directed edges between i and j so that the resulting network is simple. Suppose the weights are and respectively, and we set the weight for the merged undirected edge to be. We can use both fastgreedy.community and label.propagation.community to measure the community structure.

In order to obtain this we used the as.directed (mode = “collapse”, edge.attr.comb = sqrt_weights) where sqrt_weights is an external function that performs the weight function as described above. Since the resulting graph is simple, following community functions are used.

Label Propagation Community

Since the resulting graph was simple label.propagation.community was used to obtain the communities of the network. The modularity and sizes of the community structure obtained were as follows:

Modularity of Community = 0.0001698002.

Table 2.3.2 Community Structure using Label Propagation

Community	1	2	3	4	5
Number of Nodes	10469	4	3	3	5

Fast Greedy Community

The fastgreed.community was also used to obtain the communities the network. The modularity and sizes of the community structure were as follows,

Modularity of Community = 0.328771

Table 2.3.3 Community Structure using Fast Greedy Community

Community	1	2	3	4	5	6	7	8
Number of Nodes	1836	791	1701	1213	2316	634	963	1033

Comparison of results

Based on above results, we can conclude that both the methods produce different results. The label.propagation.community algorithm produces a large-size sub community rather than large number of communities with smaller size while fastgreed.community algorithm is shown to produce more communities with respect to the undirected network.

2.4 Largest Community Structure

We use which.max to select the largest community from community structure of fastfeed.community algorithm. As is shown in table 2.4.1 the largest community is 5 with 2316 vertices. We deleted vertices which were not part of this new giant connected network by checking membership of nodes with this sub GCC.

The isolated community is then again fed to the fastgreed.community algorithm to find its sub-communities. The modularity and structure of sub-communities are as follows,

Modularity of Sub-Community = 0.3626932

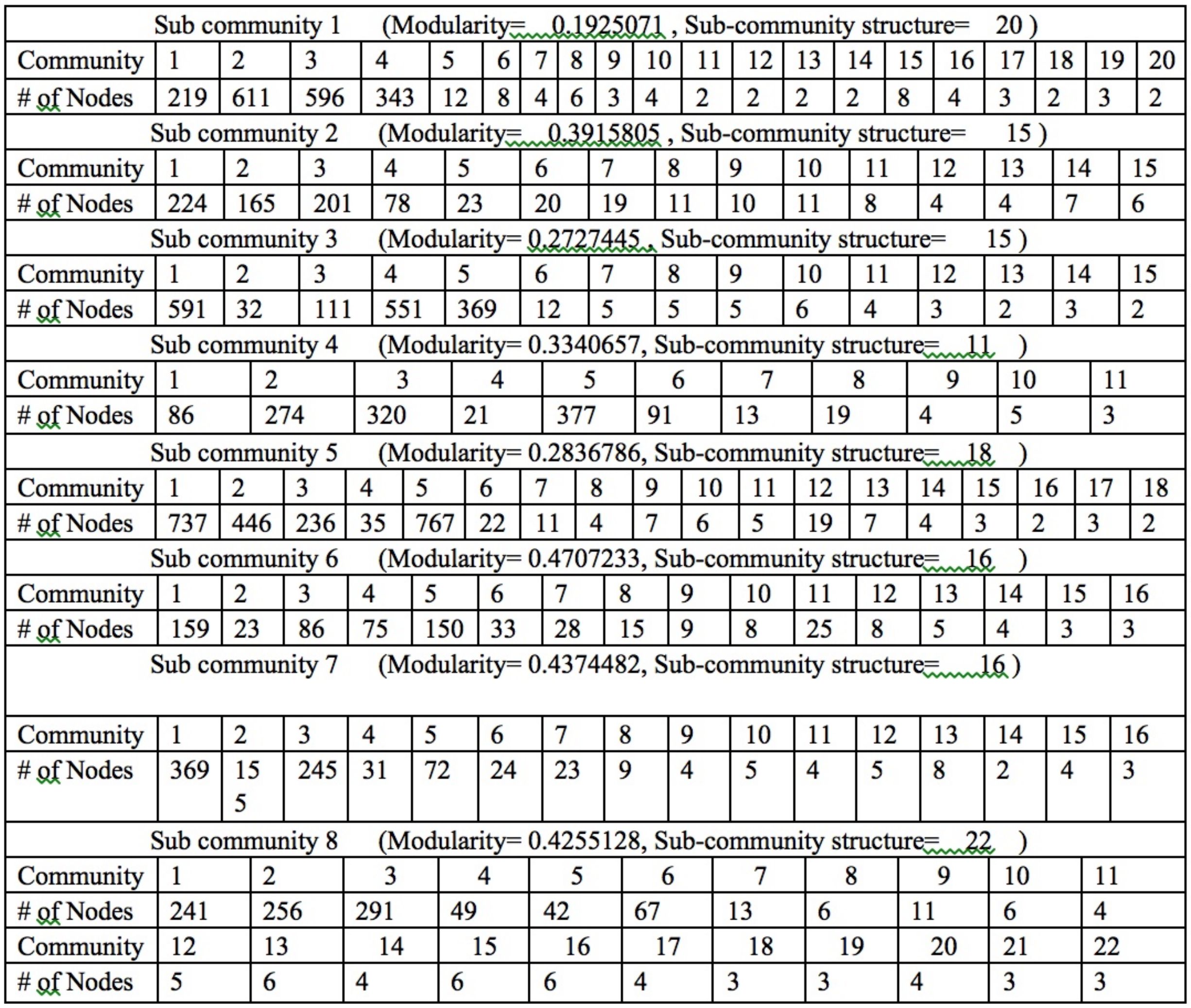
Table 2.4.1 Sub Community Structure by Fast Greedy Community Algorithm

Community	1	2	3	4	5	6	7	8	Total
Number of Nodes	39	378	417	370	32	301	341	438	2316

2.5 Sub-community Structure

In Q5, we find sub communities whose size are larger than 100. The step is similar as previous question, and we use function of *which (size (community)>100)* to inspect all sub communities. Check for both Fastgreedy Community algorithm and Label Propagation algorithm.

For Fastgreedy Community algorithm, we find that Fastgreedy Community algorithm produces 8 large sub communities whose sizes are greater than 100. And their properties are listed as below:



Sub community 1 (Modularity= 0.1925071 , Sub-community structure= 20)

Community	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
# of Nodes	219	611	596	343	12	8	4	6	3	4	2	2	2	2	8	4	3	2	3	2

Sub community 2 (Modularity= 0.3915805 , Sub-community structure= 15)

Community	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
# of Nodes	224	165	201	78	23	20	19	11	10	11	8	4	4	7	6

Sub community 3 (Modularity= 0.2727445 , Sub-community structure= 15)

Community	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
# of Nodes	591	32	111	551	369	12	5	5	5	6	4	3	2	3	2

Sub community 4 (Modularity= 0.3340657, Sub-community structure= 11)

Community	1	2	3	4	5	6	7	8	9	10	11
# of Nodes	86	274	320	21	377	91	13	19	4	5	3

Sub community 5 (Modularity= 0.2836786, Sub-community structure= 18)

Community	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
# of Nodes	737	446	236	35	767	22	11	4	7	6	5	19	7	4	3	2	3	2

Sub community 6 (Modularity= 0.4707233, Sub-community structure= 16)

Community	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
# of Nodes	159	23	86	75	150	33	28	15	9	8	25	8	5	4	3	3

Sub community 7 (Modularity= 0.4374482, Sub-community structure= 16)

Community	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
# of Nodes	369	155	245	31	72	24	23	9	4	5	4	5	8	2	4	3

Sub community 8 (Modularity= 0.4255128, Sub-community structure= 22)

Community	1	2	3	4	5	6	7	8	9	10	11
# of Nodes	241	256	291	49	42	67	13	6	11	6	4
Community	12	13	14	15	16	17	18	19	20	21	22
# of Nodes	5	6	4	6	6	4	3	3	4	3	3

Table 2.5.1 Sub-communities with sizes greater than 100

For Label Propagation algorithm, we find that Label Propagation algorithm produces just only one large community whose sizes are greater than 100. And total sub communities we obtained is 70, as listed below:

Sub-communities of Label Propagation algorithm, Modularity=0.2286573									
2238	4322	3458	135	35	14	15	7	12	9
8	8	6	8	7	9	5	3	6	3
6	3	5	4	3	8	2	4	3	3
3	3	2	3	8	2	3	3	2	3
2	6	2	3	4	4	2	4	2	3
2	3	3	2	2	7	4	3	3	3
2	2	3	3	3	2	2	2	3	2

Table 2.5.2 Sub-communities of Label Propagation algorithm

2.6 Random Walk - Community Structure

To study the overlapped community structures, personalized *PageRank* is used. First we started at node *i* and computed the visit probability of other nodes. The random walk process can be done with *netrw* package, as HW2 did, and the visit probability of other nodes can be realized with function of *$ave.visit.prob* in the *netrw* package. For simplicity, as the assignment requires, only the largest 30 nodes being visited are considered in the calculation. Also, a threshold is needed to remove the memberships that have very small values in M_i, and by back and forth test, we set the threshold range from 0.5 to 0.1, with decrement of 0.1. Try both methods (Label Propagation Community and Fast Greedy Community), lots of nodes that belong to multiple communities were found. If one node satisfies the condition of $length(M_i)>2$, we can assure that this node belongs to different communities, and they are listed as below:

Label Propagation Community, Threshold=0.4	
# of nodes with multiple community	2
Node Number	7272,7273
Label Propagation Community, Threshold=0.3	
# of nodes with multiple community	3
Node Number	7272,7273,4966
Label Propagation Community, Threshold=0.2	
# of nodes with multiple community	6
Node Number	7272,7273,4966,4967,7370,9650
Label Propagation Community, Threshold=0.1	
# of nodes with multiple community	21
Node Number	4964,4965,4966,4967,4968,4969,7370,7371,7372,7373,8218,8219,8220,9034,9035,9036, 9646, 9647,9648, 9649 ,9650
Fast Greedy Community, Threshold=0.4	
# of nodes with multiple community	6
Node Number	6794,6795,8362,8363,10079,10080
Fast Greedy Community, Threshold=0.3	
# of nodes with multiple community	12
Node Number	6794,6795,7378,8348, 8362,8363,8462, 9260,9911,10079,10080,10431

Table 2.6.1 Nodes belong to multiple communities

From table 2.6.1, it is easily found that Fast Greedy method has a higher threshold compared to Label Propagation method. The number of nodes with multiple community grows too large as threshold moves to 0.2 and 0.1, using Fast Greedy Community, so we didn't list all of them. But it is clearly seen that there are more nodes who belong multiple community given the same threshold using Fast Greedy algorithm. As a result, Fast Greedy algorithm generates more nodes that belong to multiple communities. Because in 2.3, Fast Greedy algorithm also generates more communities compared to Label Propagation algorithm, the result from 2.3 and 2.6 are consistent.

3. Summary

This assignment is intent to explore and analyze a realistic network and we have practiced a lot based on the community detection algorithms, converting directed graph to undirected one and measure the community structures. Specifically, the topics of network connectivity, degree distribution, fastgreedy community detection algorithm and label propagation community algorithm are covered in this assignment. Considering the realistic cases that a node in one community can possibly belong to another or even more communities (community overlap). We are enlightened by the idea of personalized PageRank algorithm and use this to find example nodes that belong to multiple communities. In general, this assignment is a good review of previous exercise and many of the operations utilized before, and we also have a good understanding of real network structure for the preparation of ongoing projects.

4. Useful Links

[1] https://www.dropbox.com/s/oybns7fvztfy76q/sorted_directed_net.txt?dl=0