Homework 3

Aozhu Chen 004773895
Ruoxi Zhang 404753334
Ning Xin 704775693

In this assignment, we will study a real network. We are going to study the properties of this real network. 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.

P1. Is this network connected? If not, find out the giant connected component (strongly connected). And in the following, we will deal with this giant connected component.

Not connected.

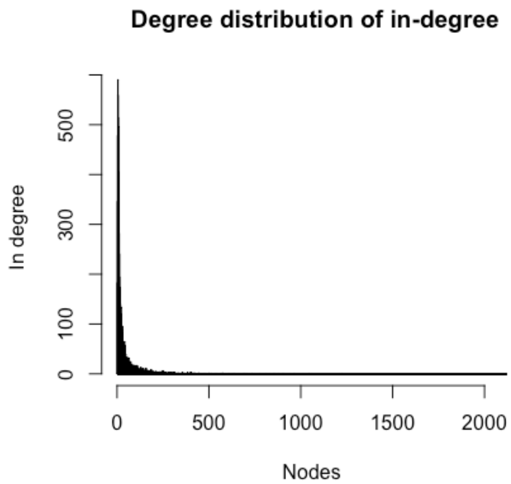
Firstly, we are going to load the data from sorted_directed_net.txt file. By using scan() function we can get all the data from the file. The result shows reading 427486 records.

Next, we used the is.connected() method which is already exist in the igraph library. By doing this step we can get the connectivity of our network. The result shows false. Thus, we know the network is not connected.

Finally, we need to find out the giant connected component (Strong connected). Clusters() method was used to collect all the clusters. We used which.max() method to find the giant connected component index. Then we use a for loop to assign the giant connected value to our giant connected node. The result shows the number of vertices in giant connected component is 10487.

P2. Measure the degree distribution of in-degree and out-degree of the nodes. (plot and briefly analyze)

After briefly analyzing the graph below, we can figure out that the degree distribution of in-degree and out-degree are almost same. When the number of nodes is small, the degree density is larger. When the number of nodes larger than 500, the density of in-degree and out-degree are all almost same which means no dramatic changes.



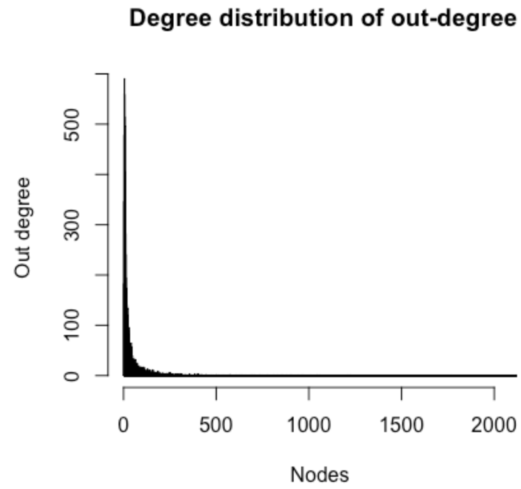


Figure 2.1 In-degree degree distribution

Figure 2.2 Out-degree degree distribution

P3. We would like to measure the community structure of the network. First, we need to convert it into an undirected network. Then we use fastgreedy.community and label.propagation.community to measure the community structure. Are the results of these two methods similar or not?

According to the answer of question 1, we know that the graph is not connected, then we compute the community structure in 2 different ways, label.propagation and fastgreedy. The results are listed below:

Since the original graph is directed, so we need to first convert it into undirected one. We have 2 options:

Option 1: Simply remove the directions

In this way, we didn't modify the number of nodes nor the weight of each edge, we simply removed the direction of each edge.

Label Propagation Community

In igraph, we call *label.propagation.label* function on the undirected function we got, the result are listed below:

Modularity of Community = 0.0001290503

Index	1	2	3	4	5
Size	10473	3	3	3	5

Table 3.1 Community sizes using label propagation (option 1)

Option 2: Merge edges

In this way, we will merge the edges of single nodes and then use the square root of product of all edge as the new weight. In igraph library, we can use function *as.directed(mode="collapse" edge.attr.comb = sqrt_wt)* to commute, where the sqrt_wt function is defined previously.

In this option, we applied both label propagation method and fast greedy method.

Label Propagation Community

In igraph, we call *label.propagation.community* function on the undirected function we got, the result are listed below:

Modularity of Community = 0.0001494257

Index	1	2	3	4	5
Size	10472	4	3	3	5

Table 3.2 Community sizes using label propagation (option 2)

Fast Greedy Community

In igraph, we call *fastgreedy.community* function on the undirected function we got, the result are listed below:

Modularity of Community = 0.328771

Index	1	2	3	4	5	6	7	8
Size	1836	791	1701	1213	2316	634	963	1033

Table 3.3 Community sizes using fast greedy (option 2)

As we can see from the result modularity and the community size matrix, we can get more membership from fast greedy than that from label propagation algorithm. Fast greedy algorithm will also produce a much larger modularity value. From the matrix, we can find that, fast greedy algorithm tends to create multiple clusters with less size difference, whereas label propagation tends to create clusters as big as possible.

P4. Findthelargestcommunitycomputedfromfastgreedy.community with option 2. Isolate the community from other parts of the net- work to form a new network, and then find the community structure of this new network. This is the sub-community structure of the largest community.

From the table we got in table 3.3, the largest community generated by fast greedy algorithm is 5^{th} one, which has 2316 nodes. Then we deleted the vertices that are not part of this giant connected network by checking the membership of nodes with this GCC.

Then we fed the remaining part to fast greedy algorithm again to find the sub-community structure, the result are showed below:

Modularity of Community = 0.1503723

Index	1	2	3
Size	2304	9	3

Table 4.1 Community sizes using fast greedy

P5. Find all the sub-community structures of the communities(found by fastgreedy.community with option 2) whose sizes are larger than 100 .

In this section, we want to find all the sub-community structure of the communities that have size larger than 100. Based on the result from question3 option 2 with fast greedy algorithm, we extract the communities has size larger than 100.

Community Size > 100								
Community	1	2	3	4	5	6	7	8
# of nodes	1836	791	1701	1213	2316	634	963	1033

Then we apply the fast-greedy algorithm and label propagation algorithm with weight to find the sub-community structure. The result is as following:

- Fast greedy algorithm with weight

Sub-community 1							
Sub-Community	1	2	3	4	5	6	7
# of nodes	262	454	492	398	88	126	16
Modularity	0.2231						

Sub-community 2															
Sub-Community	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
# of nodes	134	67	262	113	65	59	31	15	13	4	7	4	7	6	4
Modularity	0.4193														

Sub-community 3									
Sub-Community	1	2	3	4	5	6	7	8	9
# of nodes	502	358	346	142	303	32	10	5	3
Modularity	0.3716								

Sub-community 4									
Sub-Community	1	2	3	4	5	6	7	8	9
# of nodes	279	182	281	88	53	159	69	98	4
Modularity	0.3976								

Sub-community 5								
Sub-Community	1	2	3	4	5	6	7	8
# of nodes	39	378	417	370	32	301	341	438
Modularity	0.3627							

Sub-community 6															
Sub-Community	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
# of nodes	170	68	78	156	40	43	33	19	8	3	3	4	3	3	3
Modularity	0.4785														

Sub-community 7														
Sub-Community	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# of nodes	296	198	88	169	77	29	65	10	6	3	3	4	8	7
Modularity	0.5002													

Sub-community 8														
Sub-Community	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# of nodes	190	57	248	124	90	72	25	112	83	6	9	6	4	7
Modularity	0.5053													

- label propagation algorithm with weight

Sub-community 1		
Sub-Community	1	2
# of nodes	1833	3
Modularity	0.0001	

Sub-community 2													
Sub-Community	1	2	3	4	5	6	7	8	9	10	11	12	13
# of nodes	612	15	113	6	9	6	5	3	6	6	2	3	5
Modularity	0.3027												

Sub-community 3				
Sub-Community	1	2	3	4
# of nodes	1688	7	3	3
Modularity	0.0014			

Sub-community 4									
Sub-Community	1	2	3	4	5	6	7	8	9

# of nodes	1128	56	5	7	3	3	5	3	3
Modularity	0.0526								

Sub-community 5		
Sub-Community	1	2
# of nodes	2304	12
Modularity	0.0044	

Sub-community 6										
Sub-Community	1	2	3	4	5	6	7	8	9	10
# of nodes	453	141	4	5	13	6	3	3	3	3
Modularity	0.2908									

Sub-community 7															
Sub-Community	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
# of nodes	663	215	13	3	9	5	15	8	8	4	5	5	4	3	3
Modularity	0.3817														

Sub-community 8																		
Sub-Community	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
# of nodes	51	827	43	23	26	6	4	13	3	4	5	4	4	7	4	3	3	3
Modularity	0.1975																	

P6. Bothfastgreedy.communityandlabel.propagation.community assume that each node belongs to only one community. In practice, a node can belong to two or more communities at the same time. There is no command in igraph that can detect overlapped communities. Here we are going to use personalized PageRank to study the overlapped communities structures.

Both fastgreedy.community and label.propagation.community assume that each node belongs to only one community. In this section, we use the idea of personalized PageRank to study the overlap community structures. The basic idea is as following:

a. Set the teleportation probability. Teleportation probability distribution is 1 to the node under examination and 0 to every other node.
b. Start the random walk from a node i with the damping parameter 0.85 in the original directed network to get the visit probability of each node.
c. We only calculate the community score of the top 30 node with largest visit probability. The community score is calculated as following

$$M_i = \sum_j v_j m_j$$

where v_j is the visiting probability of node j and m_j is its community membership. n dimension vector mi has only one element to be 1 which represent the community it belongs to.

d. We tried different threshold value for different algorithm (0.3,0.4,0.45 for fast greedy and 0.2,0.3,0.4,0.5 for label propagation). Since the community structure of original network is very similar for option 1 and option 2 using label propagation algorithm in question 3. Therefore, we only test label propagation method once.

The results are as following:

- Label Propagation method
    - **Threshold = 0.2**

      # of multi-community nodes = 7

Node	
4966	4967
7372	7373
8220	9647
9648	

The corresponding community scores are as following

Node	V1	V2	V3	V4	V5	V6
4966	0.58	0.34	0	0	0	0
4967	0.56	0.22	0	0	0	0
7372	0.46	0	0.54	0	0	0
7373	0.55	0	0.45	0	0	0
8220	0.53	0	0	0.21	0	0
9647	0.45	0	0	0	0	0.27
9648	0.41	0	0	0	0	0.23

- **Threshold = 0.3**

  # of multi-community nodes = 3

Node
4966
7372
7373

The corresponding community scores are as following

Node	V1	V2	V3	V4	V5	V6
4966	0.58	0.34	0	0	0	0
7372	0.46	0	0.54	0	0	0
7373	0.55	0	0.45	0	0	0

- **Threshold = 0.4**

  # of multi-community nodes = 2

Node
7372
7373

The corresponding community scores are as following

Node	V1	V2	V3	V4	V5	V6
7372	0.46	0	0.54	0	0	0
7373	0.55	0	0.45	0	0	0

- **Threshold = 0.5**
  # of multi-community nodes = 0

- Fast greedy method
  - **Threshold = 0.3**
    # of multi-community nodes = 10

Node	
4969	8377
6794	6795
7176	7801
8362	8363
10079	10080

The corresponding community scores are as following

Node	V1	V2	V3	V4	V5	V6	V7	V8
4969	0.02	0	0.37	0.01	0.12	0	0.03	0.31
6795	0	0	0.44	0	0.56	0	0	0
6795	0	0	0.44	0	0.56	0	0	0
7176	0.31	0.02	0.02	0.03	0.33	0	0.01	0
7801	0.32	0	0.05	0.05	0.44	0.01	0.03	0
8362	0.55	0	0	0.45	0	0	0	0
8363	0.45	0	0	0.55	0	0	0	0
8377	0.07	0.01	0.3	0	0.3	0.02	0.01	0.02
10079	0	0.46	0	0	0	0.54	0	0
10080	0	0.53	0	0	0	0.47	0	0

  - **Threshold = 0.4**
    # of multi-community nodes = 6

Node	
6794	6795
8362	8363
10079	10080

The corresponding community scores are as following

Node	V1	V2	V3	V4	V5	V6	V7	V8
6794	0	0	0.55	0	0.45	0	0	0
6795	0	0	0.44	0	0.56	0	0	0
8362	0.55	0	0	0.45	0	0	0	0
8363	0.45	0	0	0.55	0	0	0	0

10079	0	0.46	0	0	0	0.54	0	0
10080	0	0.53	0	0	0	0.47	0	0

- **Threshold = 0.45**
  # of multi-community nodes = 2

Node	
10079	10080

The corresponding community scores are as following

Node	V1	V2	V3	V4	V5	V6	V7	V8
10079	0	0.46	0	0	0	0.54	0	0
10080	0	0.53	0	0	0	0.47	0	0

- Conclusion

From the tables above, we can conclude that when threshold is larger than around 0.45 for both fast greedy algorithm and label propagation algorithm, there will not exists any multi-community nodes.