Homework 3
EE232E - Graphs and Network Flows

UID: 404395468 Name: Lele Gao
UID: 704406413 Name: Qinyi Yan
UID: 704588291 Name: Dijia Fan

In homework 3, we are going to study a real network which is stored in a .txt file. In this network, there are 10501 vertices and 427486 edges.

Question 1.

In Question 1, we will carry out corresponding experiments to probe into the properties of this network. Firstly, we have to construct the graph using the dataset and judge whether it is connected. From the .txt file we can find that there are 10501 vertices and 427486 edges in the network and it is a directed & weighed network. After constructing the graph, we find that the network is not connected. We clustered the network and find the giant connected component which contains almost all the nodes in the graph. Its size is 10487.

```
> is.connected(g)
[1] FALSE
> vcount(gcc)
[1] 10487
```

Question 2.

In Question 2, we will try to find the giant connected component in this graph if it is not connected. Also the degree distribution of in-degree and out-degree of the nodes will be measured.

Firstly, we plot the degree distribution of the giant connected component (we call it gcc) which is shown below. It is obvious that the distribution obeys power law which means a vast majority of the nodes have small number of degrees. Then, we plot the in-degree and out-degree distribution of the gcc. From the figures below, we find that these two distributions are almost the same.

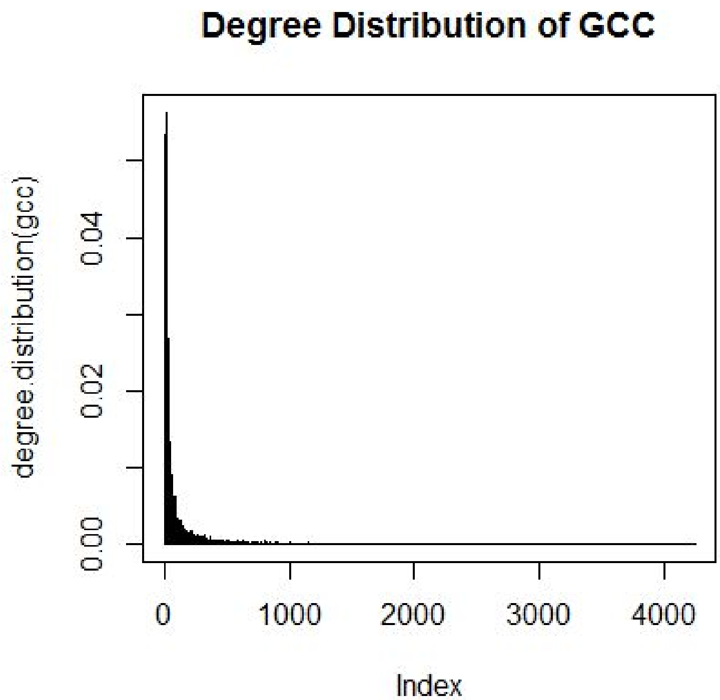


Figure 1: Degree Distribution of GCC

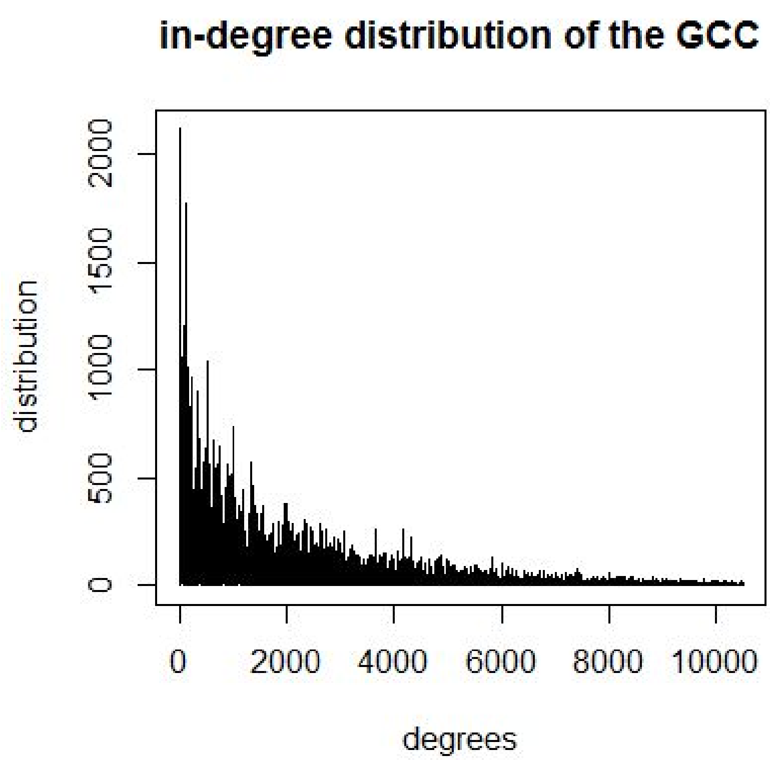


Figure 2: In-degree Distribution of GCC

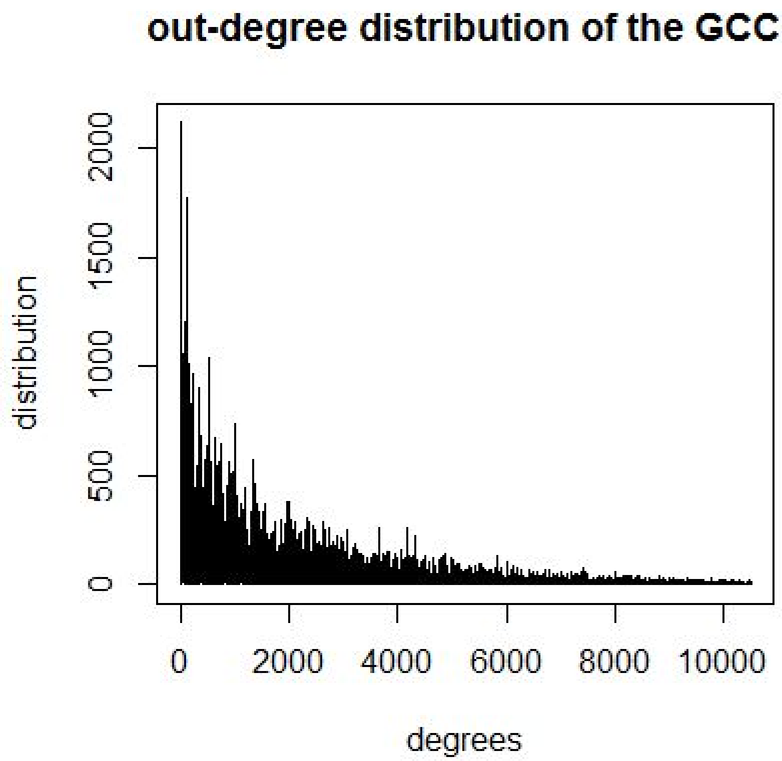


Figure 3: Out-degree Distribution of GCC

Question 3.

In Question 3, we will measure the community structure of the network. This project will help us understand the inner mechanism of complex networks.
We firstly converted the directed graph into an undirected network. Then, we detected the community structure of the network using two methods respectively. The result shows that the two algorithms performed differently and fastgreedy.community algorithm has a better performance in terms of modularity. We conclude the results as below, the size of each community is shown in below tables.

Table1: Comparison of label.propagation and fastgreedy algorithms

	modularity	**community number**
label.propagation.community	0.000217338	13
fastgreedy.community	0.2622572	15

Table2: Membership details for label.propagation algorithm

Community ID	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**

Node numbers	10,469	2	4	2	2	3	2	3	2	3	5	2	2

Table3: Membership details for fastgreedy algorithm

Community ID	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Node numbers	1810	787	1703	1191	1017	2379	960	640
Community ID	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**-**
Node numbers	2	2	2	2	2	2	2	-

As can be seen in the results above, the two algorithms performed differently in which label.propagation algorithm divides the graph into fewer communities with a very small modularity whereas fastgreedy algorithm results in larger number of communities with a comparatively larger modularity.

We try to visually display the community detection results of the two algorithms. Figure 4 shows the community structure detected using label.propagation algorithm.

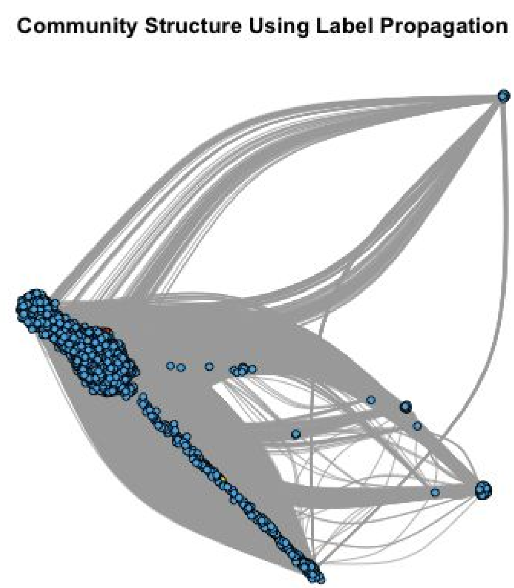


Figure 4: Communities detected using label.propagation of option 1

In option 2, we redefine the weights to form an undirected graph where the new weight is $\sqrt{w_{ij}.w_{ji}}$. Then, we use both the algorithms to detect the community structures and display the results below.

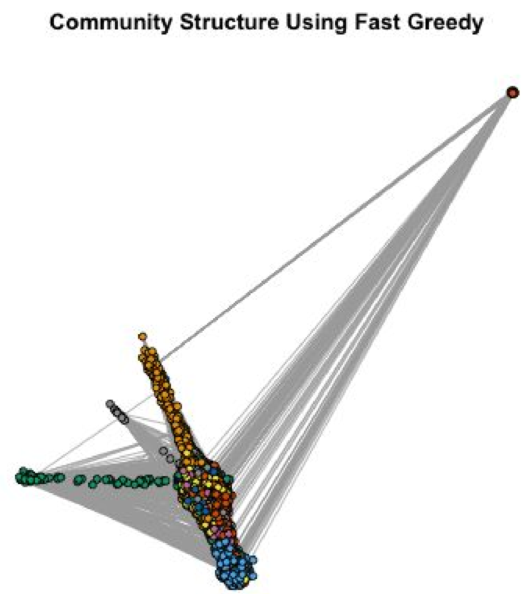


Figure 5: Communities detected using fastgreedy of option 2

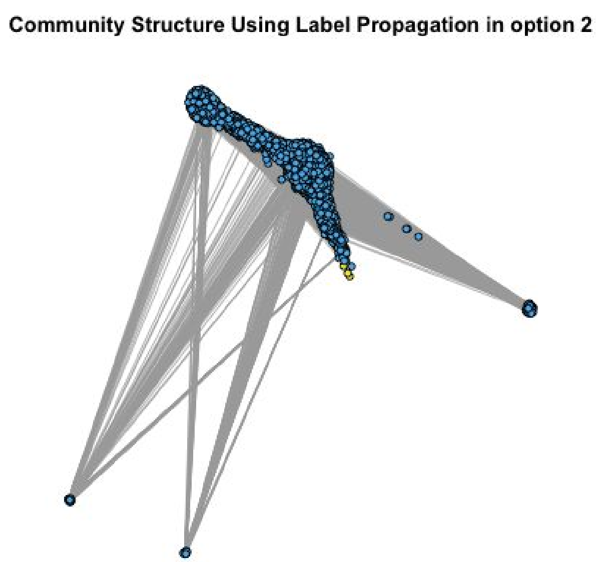
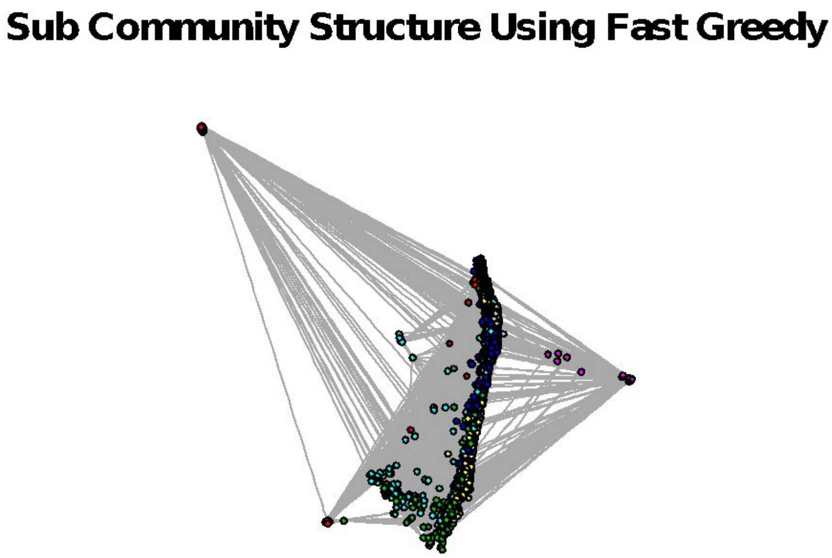


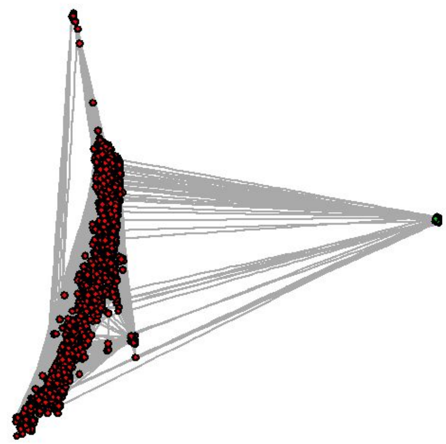
Figure 6: Communities detected using label.propagation of option 2

Question 4:
Find the largest community computed from *fastgreedy.community()*. Isolate the community from other parts of the network 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.
Solution:
We get two sub-communities, one is calculated by *fastgreedy.community()*, and the other is calculated by *label.propagation.community()*.



the modularity of this sub-community is 0.3595153

Sub Community Structure Using Lable Propagation



the modularity of this sub-community is 0

Question 5:
Find all the sub-community structures of the communities with sizes larger than 100.
Solution:
We get 8 sub-community structures with sizes larger than 100.
Structure 1 (fast greedy)

1	2	3	4	5	6	7
262	454	492	398	88	126	16

Modularity: 0.2230858

Structure 1 (label propagation)

1	2	3
1832	3	1

Modularity: 0.0001310914

Structure 2 (fast greedy)

1	2	3	4	5	6	7	8	9	10
134	67	262	113	65	59	31	15	13	4
11	12	13	14	15					
7	4	7	6	4					

Modularity: 0.4193492

Structure 2 (label propagation)

1	2	3	4	5	6	7	8	9	10
607	15	115	6	11	6	6	4	3	7
11	12	13							
2	4	5							

Modularity: 0.3052929

Structure 3 (fast greedy)

1	2	3	4	5	6	7	8	9
502	358	346	142	303	32	10	5	3

Modularity: 0.3716341

Structure 3 (label propagation)

1	2
1698	3

Modularity: 0.0002298961

Structure 4 (fast greedy)

1	2	3	4	5	6	7	8	9
279	182	281	88	53	159	69	98	4

Modularity: 0.3975836

Structure 4 (label propagation)

1	2	3	4	5
1196	5	5	3	4

Modularity: 0.003770803

Structure 5 (fast greedy)

1	2	3	4	5	6	7	8
39	378	417	370	32	301	341	438

Modularity: 0.3626932

Structure 5 (label propagation)

1	2
2304	12

Modularity: 0.004410772

Structure 6 (fast greedy)

1	2	3	4	5	6	7	8	9	10
170	68	78	156	40	43	33	19	8	3
11	12	13	14	15					
3	4	3	3	3					

Modularity: 0.4785346

Structure 6 (label propagation)

1	2	3	4	5	6	7	8	9	10
393	126	35	4	2	4	5	15	13	3
11	12	13	14	15	16	17	18	19	
5	6	3	3	5	3	3	3	3	

Modularity: 0.3699817

Structure 7 (fast greedy)

1	2	3	4	5	6	7	8	9	10
296	198	88	169	77	29	65	10	6	3
11	12	13	14						
3	4	8	7						

Modularity: 0.5002231

Structure 7 (label propagation)

1	2	3	4	5	6	7	8	9	10
394	451	7	23	15	3	6	5	16	4
11	12	13	14	15	16	17	18		
7	6	4	7	4	4	4	3		

Modularity: 0.3928051

Structure 8 (fast greedy)

1	2	3	4	5	6	7	8	9	10
190	57	248	124	90	72	25	112	83	6
11	12	13	14						
9	6	4	7						

Modularity: 0.5053173

Structure 8 (label propagation)

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
40	552	4	184	45	22	40	44	3	6
11	12	13	14	15	16	17	18	19	20
7	28	9	4	5	4	4	1	4	4
21	22	23	24	25	26				
3	3	5	3	6	3				

Modularity: 0.4077668

Question 6 :

For this question, we assume that each node belongs to more than one communities. We are going to use personalized PageRank to find the visiting probability of each node in the connected graph. We picked several nodes that with the largest visiting probability and calculated M parameter for them. The expression for M is given by:

$$\vec{M}_i = \sum_j v_j \vec{m}_j$$

where v_j is the visiting probability of node j and m_j is the community membership of this node. In order to indicate multiple community memberships, we modify the m_j to a n dimensional vector expression with only one expression being 1.
In our case, we use the top 30 nodes with the largest visiting probability to calculate the M parameter for them. The threshold is set to filter out the communities with small value in M.
According to some experimental results, we chose different threshold for the two algorithms. For fast greedy algorithm, we set the threshold to be 0.3, and found out 58 nodes belonging to more than 1 communities. For label propagation algorithm, we set the threshold to be 0.1, which yields 24 nodes belonging to multiple communities.
The nodes belonging to multiple communities for the two algorithms are shown in the following two tables.

nodes with multiple communities - Fast greedy algorithm			
356	868	2084	2266
2851	3730	3828	4298
5180	5244	5339	6205
6328	6474	6591	6608
6623	6696	6697	6706
6803	6804	6897	7141
7158	7159	7182	7573
7587	7864	8249	8601
9060	9231	9282	9283
9427	9462	9503	9512
9593	9675	9686	9696
9697	9734	9784	9833
9946	9984	9987	10151
10260	10311	10366	10367
10397	10459		

nodes with multiple communities -label propagation algorithm			
4687	7733	10015	10176
10177	10178	10179	10348
10349	10350	10351	10352
10401	10402	10403	10464
10465	10466	10475	10476
10477			