EE232E - Graphs and Network Flows
Homework 3

Swati Arora: 404758379
Vishank Bhatia: 304758488
Anshita Mehrotra: 904743371

Question 1: Network Connectivity

Part (a)

For part(a), we read the edge list in a tabular format using the ‘read.table’ command and created a directed graph using **‘graph.data.frame’** of the **igraph** package.

Part (b)

In part (b), we **check for connectivity** for the directed network graph that was created in part(a). We use the ‘**is_connected’** method for this.

Result obtained: The directed graph is **not connected**.

Part (c)

In part (c) we calculate the giant connected component since it is not a strongly connected graph as indicated in part(b).
Steps followed are as follows:

1. Find out all the clusters for the directed graph using ‘**clusters’** method and obtain the maximum size cluster which represents GCC.
2. Obtain the list of vertices which form a part of this GCC by checking the membership of the node.
3. Obtain the subgraph from the graph consisting only the nodes obtained in step 2 using ‘**induced_subgraph’** method.

	Value
Number of vertices in GCC	10487
Number of vertices in network	10501
Number of clusters in network	8

Table 1: Results for number of vertices in GCC and number of vertices and clusters in network

Question 2: Degree Distribution (GCC)

Degree distribution was computed for GCC and following distribution was obtained:

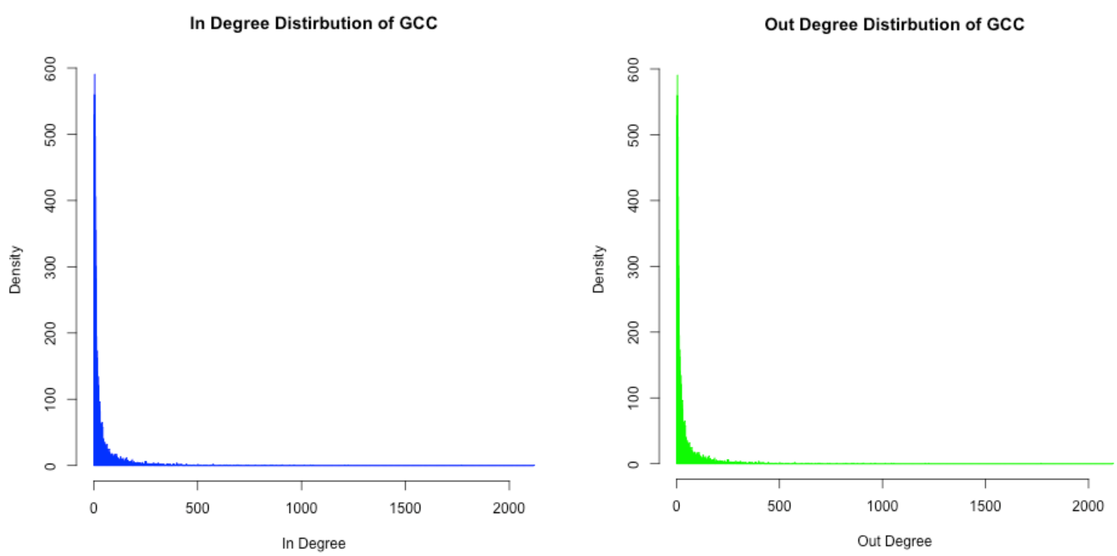

Figure 1 : In degree and Out degree distribution for GCC

Observations:

- It can be clearly seen that **maximum nodes have same in degree and out degree measure.**
- The distribution peaks for initial small values of in-degree and out-degree and drops to 0 for higher values.

Question 3: Community Structure (GCC)

Option (1) Unchanged Edges

In option (1) we obtain undirected network from directed graph keeping the edges unchanged and only removing the directions using **‘as.undirected’** method and method parameter **‘mode=each’**

Community Structure: Label propagation

As the undirected graph obtained by using this option is not simple, only label propagation is used to compute a weighted, non-simple network’s community structure. The method used is **‘label.propagation.community’**.
Modularity and community information obtained is as follows:

Modularity: 0.000205
Number of communities: 5

	Value				
Community	1	2	3	4	5
Number of nodes	10472	4	3	3	5

Table 2: Results for community structure using label propagation

Option (2) Collapsed Edges

In option (1) we obtain undirected network from directed graph keeping the edges unchanged and only removing the directions using **‘as.undirected’** method and method parameter **‘mode=collapsed’**

Community Structure: Label propagation

As the undirected graph obtained by using this option is relatively simple, both label propagation and fast greedy is used to compute a weighted network’s community structure. The method used to obtain community structure by using label propagation is **‘label.propagation.community’**.
Modularity and community information obtained is as follows:

Modularity: 0.000177
Number of communities: 5

	Value				
Community	1	2	3	4	5
Number of nodes	10474	4	3	5	1

Table 3 : Results for community structure using label propagation

Community Structure: Fast greedy

As the undirected graph obtained by using this option is relatively simple, both label propagation and fast greedy is used to compute a weighted network's community structure. The method used to obtain community structure by using label propagation is **'fastgreedy.community'**.

Modularity and community information obtained is as follows:

Modularity: 0.328
Number of communities: 8

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

Table 4 : Results for community structure using Fast Greedy

Part 4: Community structure of the largest community

- In part 4, we find the largest community computed from fastgreedy.community with option 2, in part 3.
- In our case, it is the **community with the number of vertices as 2316.**
- We then isolate this community from other parts of the network to form a new network.
- We then find the community structure of this new network using the fastgreedy.community algorithm. This is the sub-community structure of the largest community. The results are as follows:

Sub Community Structure								
Community	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Nodes	39	378	417	370	32	301	341	438

Modularity of the sub-community: 0.3626

Table 5: Sub-Community structure using Fast-Greedy Algorithm

Part 5: Community structure of all sub communities with size >100

- In part 5, we find the sub-community structures of all communities which have size larger than 100.
- **No of communities with size greater than 100 was found to be 8**
- The sub-community structure was found using the fastgreed.community algorithm. The results are as follows:

Sub Community: 1							
Modularity of Sub-Community = 0.2230							
Sub Communities Structure = 7							
Community	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Nodes	262	454	492	398	88	126	16

Sub Community: 2								
Modularity of Sub-Community = 0.4193								
Sub Communities Structure = 15								
Community	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Nodes	134	67	262	113	65	59	31	15
Community	**9**	**10**	**11**	**12**	**13**	**14**	**15**	
Nodes	13	4	7	4	7	6	4	

Sub Community: 3
Modularity of Sub-Community = 0.3716

Sub Communities Structure = 9									
Community	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Nodes	502	358	346	142	303	32	10	5	3

Sub Community: 4									
Modularity of Sub-Community = 0.3975									
Sub Communities Structure = 9									
Community	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Nodes	279	182	281	88	53	159	69	98	4

Sub Community: 5								
Modularity of Sub-Community = 0.3626								
Sub Communities Structure = 8								
Community	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Nodes	39	378	417	370	32	301	341	438

Sub Community: 6								
Modularity of Sub-Community = 0.4785								
Sub Communities Structure = 15								
Community	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Nodes	170	68	78	156	40	43	33	19
Community	**9**	**10**	**11**	**12**	**13**	**14**	**15**	
Nodes	8	3	3	4	3	3	3	

Sub Community: 7							
Modularity of Sub-Community = 0.5002							
Sub Communities Structure = 14							
Community	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Nodes	296	198	88	169	77	29	65
Community	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Nodes	10	6	3	3	4	8	7

Sub Community: 8							
Modularity of Sub-Community = 0.5053							
Sub Communities Structure = 14							
Community	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Nodes	190	57	248	124	90	72	25
Community	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Nodes	112	83	6	9	6	4	7

Question 6: Personalized PageRank – Overlapping Community Structure

In case of nodes having multiple memberships i.e. if a node belongs to more than one community at the same time, personalized PageRank can come in handy to understand the community structures. The following are the steps:

(a) Calculating the visit probability of all other nodes in the network given a randomly selected node i. The teleportation probability of each start node is 1 and other nodes is 0 i.e. it is a local-pagerank-type random walk. Then, we generate a random network with random walk on node i.

(b) The visit probability of each node on the network is averaged and found using $ave.visit.prob. The largest 30 visit probabilities i.e. the nodes that are mostly likely to be visited are used and for each node, the membership is found using $membership function.

(c) Different threshold values are utilized to remove memberships having small M_i values. If the length(M_i) > 2 is satisfied for a start node, it implies that the node belongs to more than one community and hence comes in the concept of overlapping communities.

(Option 1) Using Label Propagation Community				
Threshold Value = 0.1				
# of Nodes in Overlapping Communities = 23				
3718	4968	7372	9034	9648
4964	4969	7373	9035	9649
4965	6989	8218	9036	9650
4966	7370	8219	9646	
4967	7371	8220	9647	

(Option 2) Using Label Propagation Community				
Threshold Value = 0.1				
# of Nodes in Overlapping Communities = 18				
4964	4969	9034	9648	
4965	7370	9035	9649	
4966	7371	9036	9650	
4967	7372	9646		
4968	7373	9647		

(Option 1) Using Label Propagation Community				
Threshold Value = 0.2				
# of Nodes in Overlapping Communities = 6				
4966	9648			
4967				
7372				
7373				
9647				

(Option 2) Using Label Propagation Community				
Threshold Value = 0.2				
# of Nodes in Overlapping Communities = 7				
4966	9648			
4967	9901			
7372				
7373				
9647				

(Option 1) Using Label Propagation Community				
Threshold Value = 0.3				
# of Nodes in Overlapping Communities = 2				
7372				
7373				

(Option 2) Using Label Propagation Community				
Threshold Value = 0.3				
# of Nodes in Overlapping Communities = 2				
7372				
7373				

(Option 1) Using Label Propagation Community				
Threshold Value = 0.4				
# of Nodes in Overlapping Communities = 2				
7372				
7373				

(Option 2) Using Label Propagation Community				
Threshold Value = 0.4				
# of Nodes in Overlapping Communities = 2				
7372				
7373				

(Option 1) Using Label Propagation Community				
Threshold Value = 0.5				
# of Nodes in Overlapping Communities = 0				

(Option 2) Using Label Propagation Community				
Threshold Value = 0.5				
# of Nodes in Overlapping Communities = 0				

Comments:

1. The threshold can be taken at 0.1 for option 1 and option 2.

2. Higher threshold means lesser overlapping between communities.

3. Closely placed nodes represent spatial continuity between vertices.