Graph and Network
Homework 3

Mansee Jadhav - 204567818
Ronak Sumbaly - 604591897

May 2, 2016

Ques 1. Network Connectivity & Giant Connected Component

a. Read the data

A **directed network** is constructed using the provided data-set comprising of 3 columns - ($node_1$, $node_2$ and $weight$) where each row signifies a directed edge from $node_1$ to $node_2$ and its corresponding $weight$. We created directed network from the available data using the function,

```
graph.data.frame(graph_data_file, directed=TRUE)
```

b. Check for connectivity

We checked the connectivity of the graph using `is.connected` method of igraph. The directed network is **NOT CONNECTED**.

c. Find GCC

Since the directed network is not connected we calculate it's giant connected component. In order to calculate the GCC of the graph, we followed the following procedure:

1. Get all the clusters from the directed graph using `clusters()` method and then get the cluster with the maximum size which represents GCC.
2. Get nodes which are not present in GCC cluster by checking their membership. Remove nodes from the graph to get only the GCC using `delete.vertices`.
3. The difference in number of vertices in the GCC and the directed network is shown below,

   # of vertices in GCC = 10487

   # of vertices in network = 10501

Thus it can be seen that the GCC is as large as the directed network barring a couple of vertices.

Ques 2. Degree Distribution of GCC

The **in and out degree distribution** of the directed giant connected component was computed and is shown in the following diagram. Visually it can be seen that most of the nodes have the **same in and out degree measure.**

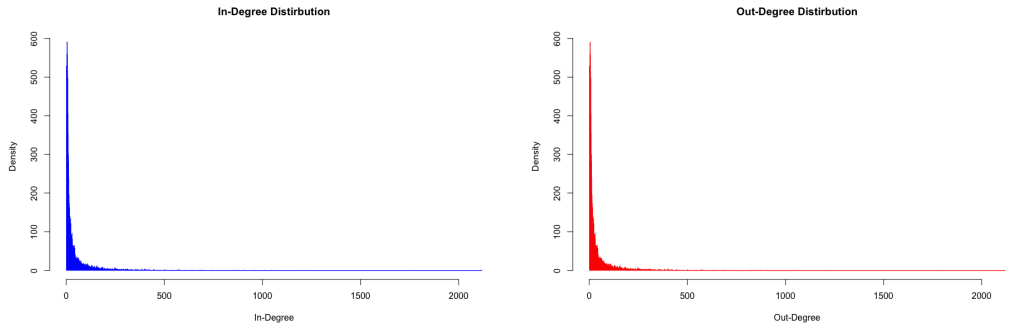

Figure 1: Degree Distribution of GCC - In (Left) Out (Right)

Ques 3. Community Structure

Since the network has edge weights the task of converting it from directed to undirected is performed in two different ways as follows,

Option 1 - Number of edges unchanged & Remove the directions

In this option we converted the directed network to an undirected network by keeping the # of edges unchanged and by removing the directions.

In order to obtain this we used the `as.directed(mode = "each")`.

Label Propagation Community

Since the resulting graph wasn't simple only `label.propagation.community` was used to obtain the communities of the network. The modularity and sizes of the community structure obtained were as follows,

$$\texttt{Modularity of Community} = 0.0001494257$$

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

Table 1: Community Structure using Label Propagation

Option 2 - Merge two directed edges between nodes

In this option we converted the directed network to an undirected network by merging the # of edges. The merged weights are assigned as the function $\sqrt{w_{ij}w_{ji}}$ where the w_{ij}, w_{ji} are the weights of the edges between two nodes.

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 were 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,

$$\texttt{Modularity of Community} = 0.0001494257$$

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

Table 2: Community Structure using Label Propagation

Fast Greedy Community

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

$$\texttt{Modularity of Community} = 0.328771$$

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

Table 3: Community Structure using Fast Greedy Community

Comparison of results

As seen from the results both the methods produce different results. The `fastgreed.community` algorithm is shown to produce more communities with respect to the undirected network while `label.propagation.community` algorithm produces a giant sub community rather than greater # of communities with smaller size.

Ques 4. Largest Community Structure

The largest community from community structure of `fastgreed.community` algorithm, was selected as the sub giant connected component using `which.max`. As seen from *Table* 3 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 (similar to result obtained in *Ques 3. Option 2*) and structure of the sub-communities is presented below,

$$\texttt{Modularity of Sub-Community} = 0.3626932$$

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

Table 4: Sub Community Structure using Fast Greedy Community

Ques 5. Sub-Community Structure

In this question we find the sub-community structures of all communities which have size larger than 100. The procedure is similar to the ones described above but instead of using `which.max` we check for `which(sizes(community)>100)`. The sub-community structure for both options was found using the `fastgreed.community` algorithm. Results for both community algorithms is presented below,

Label Propagation Community

As seen from *Table* 2 there is only one community with $size > 100$, **Community 1**. The sub-community structure for the community is given below, (each value in the table corresponds to the number of vertices in a sub-community). **Total sub-communities obtained = 70.**

$$\texttt{Modularity of Sub-Community 1} = 0.2286573$$

2,238	8	6	3	2	2	2
4,322	8	3	3	6	3	2
3,458	6	5	2	2	3	3
135	8	4	3	3	2	3
35	7	3	8	4	2	3
14	9	8	2	4	7	2
15	5	2	3	2	4	2
7	3	4	3	4	3	2
12	6	3	2	2	3	3
9	3	3	3	3	3	2

Table 5: Sub Community for Label Propagation

Fast Greedy Community

As seen from *Table* 3 there are 8 communities with $size > 100$. Each sub-community structure for the community is given below. Note : Each value in the table corresponds to the number of vertices in a sub-community

Sub-Community 1

`Modularity of Sub-Community` = 0.1925071
`Sub-Communities Structure` = 20

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

Table 6: Sub Community Structure 1

Sub-Community 2

`Modularity of Sub-Community` = 0.3915805
`Sub-Communities Structure` = 15

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

Table 7: Sub Community Structure 2

Sub-Community 3

`Modularity of Sub-Community` = 0.2727445
`Sub-Communities Structure` = 15

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

Table 8: Sub Community Structure 3

Sub-Community 4

`Modularity of Sub-Community` = 0.3340657
`Sub-Communities Structure` = 11

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

Table 9: Sub Community Structure 4

Sub-Community 5

`Modularity of Sub-Community` = 0.2836786
`Sub-Communities Structure` = 18

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

Table 10: Sub Community Structure 5

Sub-Community 6

`Modularity of Sub-Community` = 0.4707233
`Sub-Communities Structure` = 16

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

Table 11: Sub Community Structure 6

Sub-Community 7

`Modularity of Sub-Community` = 0.4374482
`Sub-Communities Structure` = 16

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

Table 12: Sub Community Structure 7

Sub-Community 8

`Modularity of Sub-Community` = 0.4255128
`Sub-Communities Structure` = 22

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

Table 13: Sub Community Structure 8

Ques 6. Random Walk - Community Structure

Inorder to exploit the idea that a node can belong to more than one community at the same time, we used personalized PageRank to study the overlapping community structures. The process was as follows,

1. Starting with random node `i`, we calculated the visit probability of all other nodes. Teleportation probability of each start node is set to 1 as its local page rank and rest of the nodes is 0. A random network was generated as a random walk on node `i` using `netrw` package.
2. Visit probability of each of the nodes in this network was calculated using `$ave.visit.prob` of `netrw`.
3. The largest 30 visit probabilities were taken into consideration and for each respective node their membership was found using `$membership` function and result from Ques. 3. and for each start node we computed,
$$M_i = \sum_j v_j m_j$$
where v_j is the visit probability and m_j is its community structure
4. Different threshold values was used to remove memberships that have very small values in M_i.
5. If in the end, the condition $length(M_i) > 2$ is satisfied for a start node it means that the node belongs to more than one community.

Using Label Propagation Community

Threshold Value = 0.5
Nodes with Multi-Community = 0

Threshold Value = 0.4
Nodes with Multi-Community = 2

Nodes
7272
7273

Table 14: Nodes with Multi-Community: Threshold = 0.4

Threshold Value = 0.3
Nodes with Multi-Community = 3

Nodes
4966
7272
7273

Table 15: Nodes with Multi-Community: Threshold = 0.3

Threshold Value = 0.2
Nodes with Multi-Community = 6

Nodes
4966
4967
7370
7372
7373
9650

Table 16: Nodes with Multi-Community: Threshold = 0.2

Threshold Value = 0.1
Nodes with Multi-Community = 20

Nodes	Nodes
4964	4969
4965	4966
4967	4968
4969	7370
7371	7372
7373	9034
9035	9036
9646	9647
9648	9649
9650	8218
8219	8220

Table 17: Nodes with Multi-Community: Threshold = 0.1

Using Fast Greedy Community

Threshold Value = 0.4
Nodes with Multi-Community = 6

Nodes	Nodes
6794	6795
8362	8363
10079	10080

Table 18: Nodes with Multi-Community: Threshold = 0.4

Threshold Value = 0.3
Nodes with Multi-Community = 12

Nodes	Nodes
6794	6795
7378	8348
8362	8363
8462	9260
9911	10079
10080	10431

Table 19: Nodes with Multi-Community: Threshold = 0.3

Analysis of Results

- It can be clearly seen from the results obtained above that the threshold limit for `fastgreed.community` is 0.3 and for `label.propagation.community` is 0.2.
- The former has a higher threshold and tends to give more nodes as compared to the latter signifying that there are more nodes belonging to multi-communities. These results are consistent with the `fastgreed.community` algorithm as it generates more communities as compared to `label.propagation.community` *(Table 2 vs. Table 3)*.
- Most of the nodes obtained are close to each other indicating a spatial continuity between the vertices.