University of California, Los Angeles
Department of Electrical and Computer Engineering
ECE 232E Large Scale Social and Complex Networks: Design and Algorithms

Project 2: Social Network Mining

505430686 Viacheslav Inderiakin

904627828 Mia Levy

805626088 Connor Roberts

804737257 Tameez Latib

April 29, 2021

1. Facebook Network.

1. Structural properties of the Facebook network

QUESTION 1: A first look at the network:

Answer: Facebook network is plotted in figure 1.

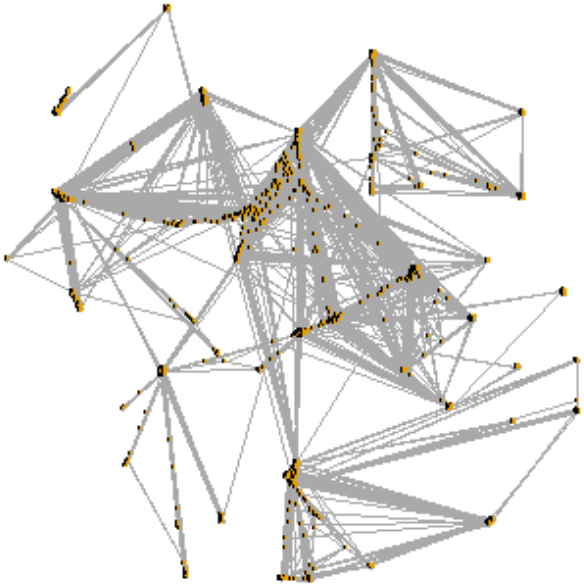


Figure 1: Facebook network.

QUESTION 1.1: Report the number of nodes and number of edges of the Facebook network:

Answer:

$$\begin{aligned} \text{Number of nodes} &= 4039 \\ \text{Number of edges} &= 88234 \end{aligned}$$

QUESTION 1.2: The Facebook network connected? If not, find the giant connected component (GCC) of the network and report the size of the GCC:

Answer: Yes, Facebook net is connected.

QUESTION 2: Find the diameter of the network. If the network is not connected, then find the diameter of the GCC:

Answer: As Facebook net is connected, we can find its diameter = 8.

QUESTION 3: Plot the degree distribution of the facebook network and report the average degree:

Answer: The degree distribution of Facebook network is provided in figure 2. Average degree of the net is 43.69.

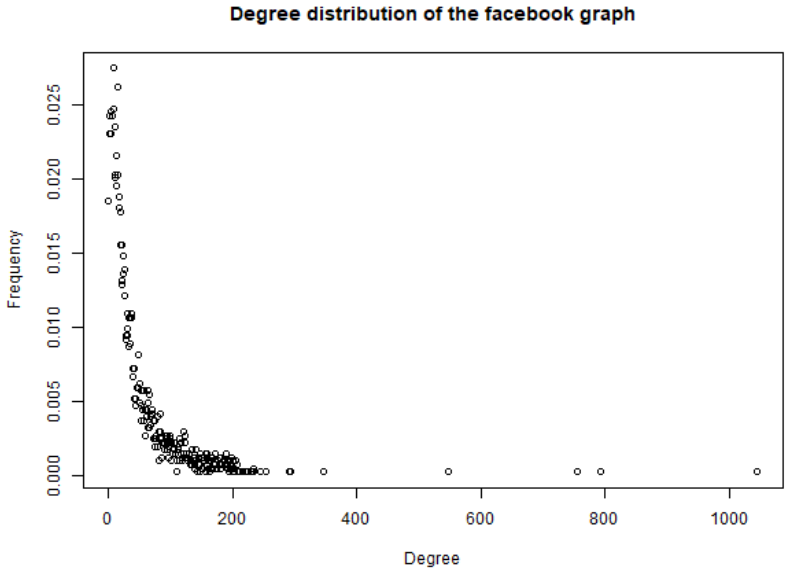


Figure 2: Facebook network degree distribution.

QUESTION 4: Plot the degree distribution of question 3 in a log-log scale. Try to fit a line to the plot and estimate the slope of the line:

Answer: The degree distribution of the Facebook network is provided in figure 3. If we try to fit a line to this distribution, its slope will be -1.18. Therefore, the Facebook network has a distribution similar to power law distribution with coefficient $k \approx 1$.

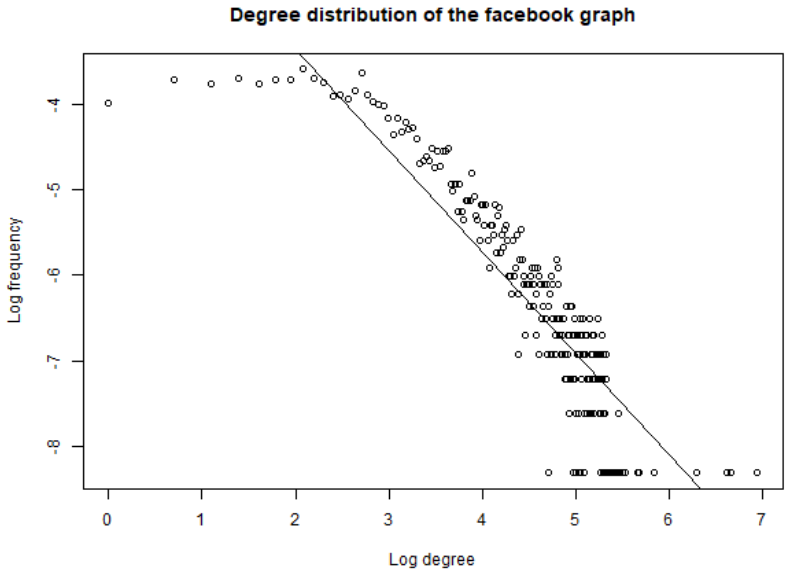


Figure 3: Facebook network degree distribution in log-log scale.

2. Personalized network.

QUESTION 5: Create a personalized network of the user whose ID is 1. How many nodes and edges does this personalized network have?

Answer: The personalized network for node 1 is provided in figure 4. The number of its nodes and vertices is provided below:

Number of nodes	=	348
Number of edges	=	2866

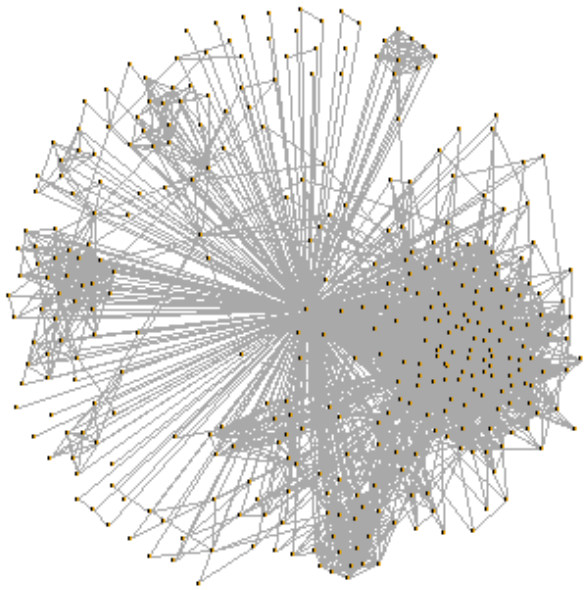


Figure 4: Personalized network for user 1.

QUESTION 6: What is the diameter of the personalized network? Please state a trivial upper and lower bound for the diameter of the personalized network:

Answer: The diameter of the personalized network of user 1 is 2. For a trivial personalized net, consisting only of a single user, the diameter is zero. For all other nets, lower bound for the diameter is 1, which is achieved when all nodes are connected with each other (i.e. when personalized network is fully connected), and we can move to any node with just one transition from any starting state. The upper bound of the diameter is 2, because both the start node and the end node are always connected to the center node, so any transition goes through 2 edges at max.

QUESTION 7: In the context of the personalized network, what is the meaning of the diameter of the personalized network to be equal to the upper bound you derived in question 6. What is the meaning of the diameter of the personalized network to be equal to the lower bound you derived in question 6 (assuming there are more than 3 nodes in the personalized network)?

Answer: Assuming there are more than 3 nodes in the personalized network, if its diameter equals to the lower bound 1, then between any 2 nodes in the network there exists an edge. If its diameter equals to the upper bound 2 (as this is the case for personalized net of node 1) this is not the case, and there are nodes which do not have an edge connecting them.

3. Core node's personalized network.

QUESTION 8: How many core nodes are there in the Facebook network. What is the average degree of the core nodes?

Answer: There are 40 core nodes. The average degree of the core nodes is 279.375. This was found by finding which nodes have a degree of over 200, and then taking the average degree of these nodes.

3.1. Community structure of core node's personalized network:

QUESTION 9: For each of the above core node's personalized network, find the community structure using Fast-Greedy, Edge-Betweenness, and Infomap community detection algorithms. Compare the modularity scores of the algorithms. For visualization purpose, display the community structure of the core node's personalized networks using colors. Nodes belonging to the same community should have the same color and nodes belonging to different communities should have different color. In this question, you should have 15 plots in total.

Answer: Comparative performance of clusterization algorithms on personalized nets is shown in table 1. From that table, we can see that there is no one-fits-all solution to clusterization of personalized nets. For example, for core id-s 1 and 1087 the best performance is achieved by Fast-Greedy algorithm. Iterative updates performed by this algorithm results in higher number of merges and bigger clusters, which better fits densely connected nets and results in higher modularity score. For sparsely connected nets like 108, 349 and 484, infomap algorithm produces smaller communities and outperforms all other algorithms. Edge-Betweenness algorithm is suboptimal in all 5 cases. Potential reason for that is the presence of the core. Since it is connected to all other vertices in the graph, many of shortest paths lie through edges connected to the core. Therefore, instead of locating communities Edge-Betweenness algorithm spends most of the time separating core from all other vertices. CLustarization performance can also be visually assessed in figure 5.

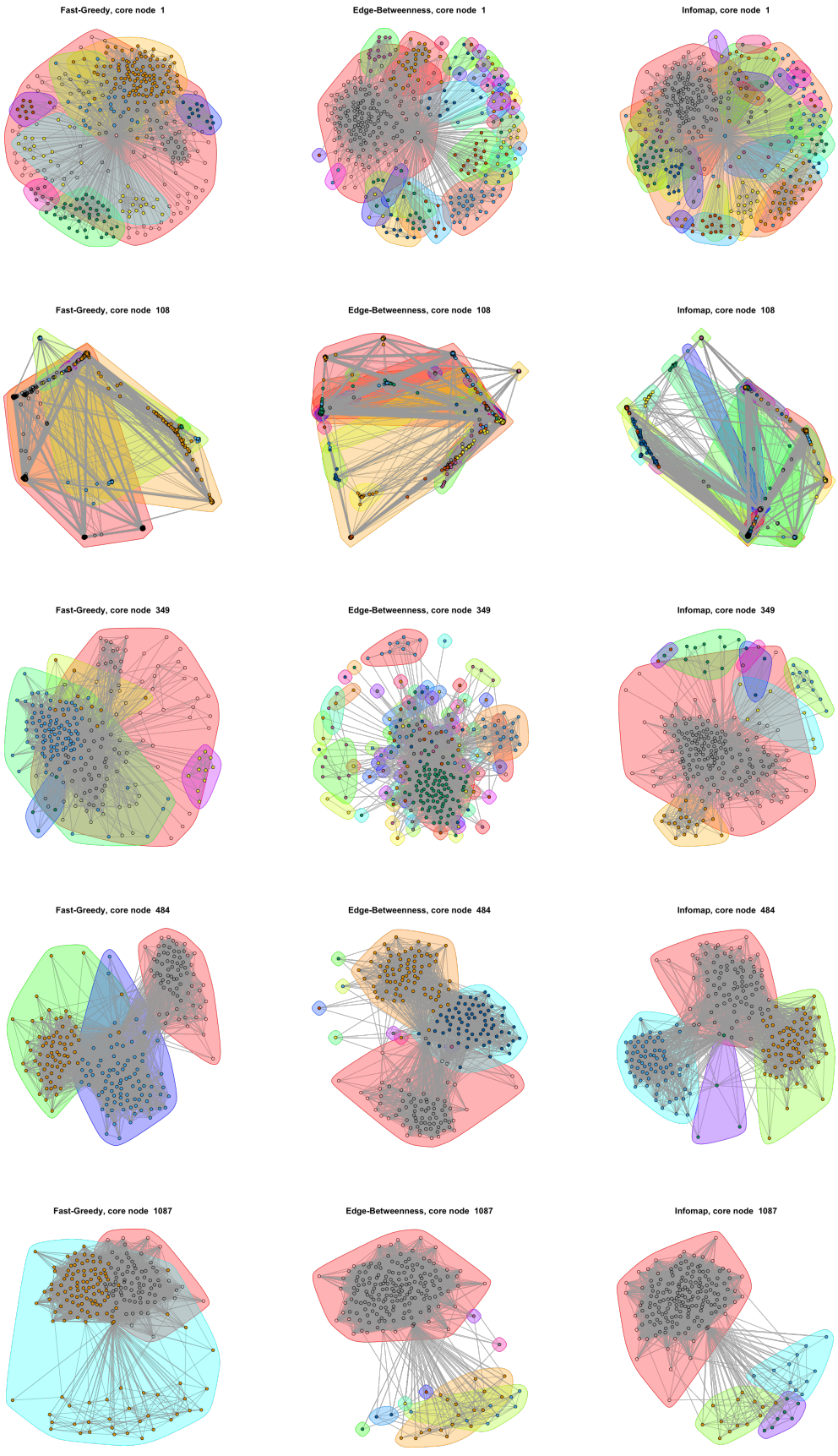


Figure 5: Community structure of personalized nets.

Core node	Fast-Greedy	Edge-Betweenness	Infomap
1	0.413	0.353	0.389
108	0.435	0.506	0.508
349	0.251	0.133	0.096
484	0.507	0.489	0.515
1087	0.145	0.027	0.026

Table 1: Modularity scores for specified core nodes and community detection algorithms.

3.2. Community structure with the core node removed.

QUESTION 10: For each of the core node's personalized network (use same core nodes as question 9), remove the core node from the personalized network and find the community structure of the modified personalized network. Use the same community detection algorithm as question 9. Compare the modularity score of the community structure of the modified personalized network with the modularity score of the community structure of the personalized network of question 9. For visualization purpose, display the community structure of the modified personalized network using colors. In this question, you should have 15 plots in total.

Answer: Since the core node is connected to all other vertices in the personalized net, its presence in any of the communities results in the increase of the number of between-clusters edges, which decreases modularity. Therefore, its removal is expected to improve clusterization results. This expectation is confirmed the empirical results shown in table 2. We can also conclude that from figure 6, which shows that the removal of the core node makes nets more sparse and results in emergence of many small clusters fully separated from others.

Core node	Fast-Greedy	Edge-Betweenness	Infomap
1	0.441	0.416	0.418
108	0.458	0.521	0.520
349	0.245	0.150	0.246
484	0.534	0.515	0.543
1087	0.148	0.032	0.027

Table 2: Modularity scores for specified core nodes and community detection algorithms.

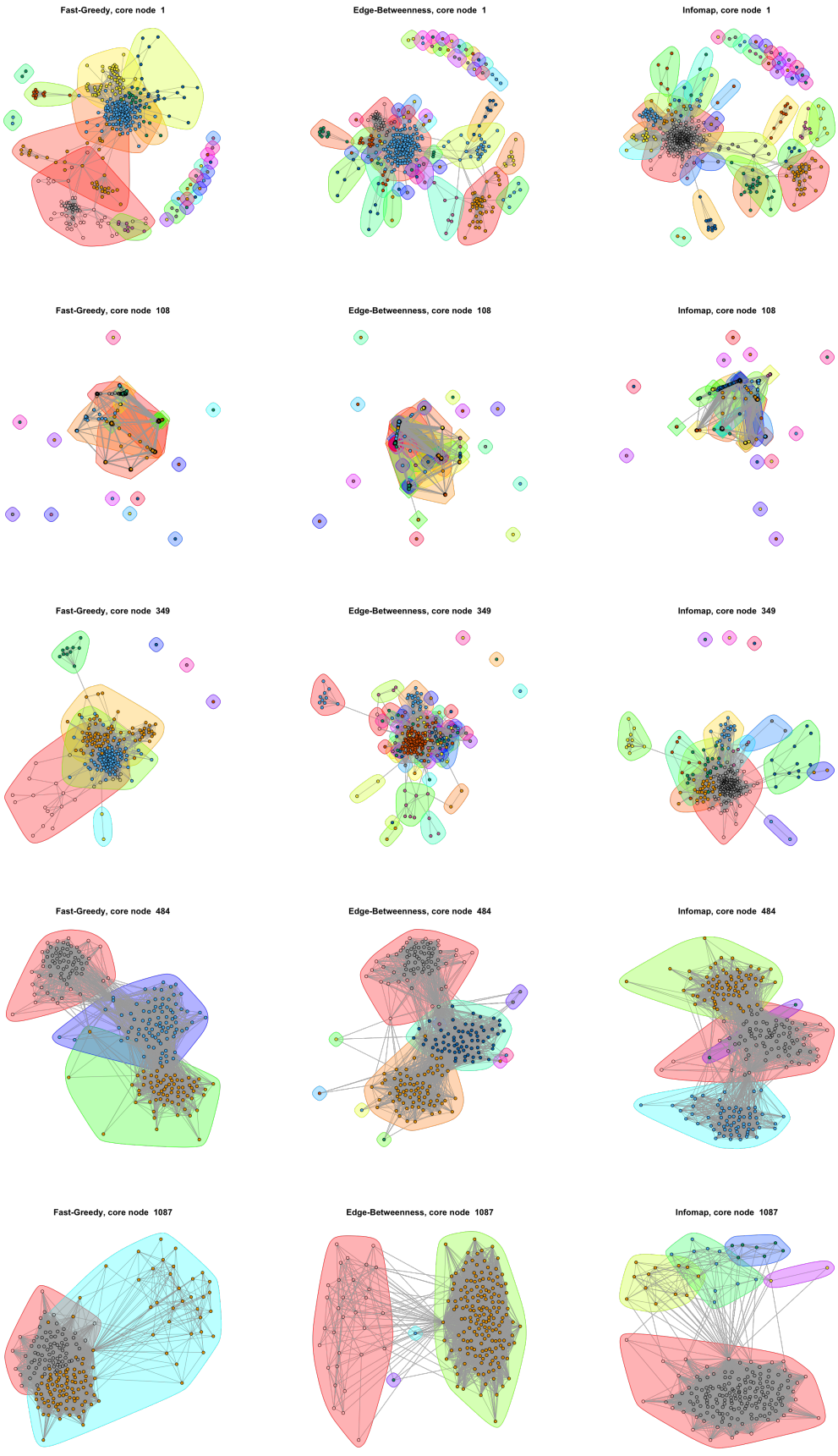


Figure 6: Community structure of personalized nets (core nodes removed).

3.3. Characteristic of nodes in the personalized network.

QUESTION 11: Write an expression relating the Embeddedness between the core node and a non-core node to the degree of the non-core node in the personalized network of the core node.

Answer:

$$embed(u, v) = deg(v) - 1$$

where u is the core node, v is a non-core node.

QUESTION 12: For each of the core node's personalized network (use the same core nodes as question 9), plot the distribution histogram of embeddedness and dispersion. In this question, you will have 10 plots.

Answer: In the context of determining the strength of social connections, distance function can be defined in many ways. In this work, we explored the following:

1. Trivial distance, where if the nodes are connected, the function returns the length of the shortest path between them; if the nodes are disconnected, the function returns constant value;
2. Thresholded from the top distance, where all distances higher than certain value are reduced to that value;
3. Thresholded from the both sides distance, where all paths which length exceeds constant value are assigned with unit distance and all others with zero distance; this case includes situation where all disconnected nodes are assigned with distance of 1 and all connected nodes with distance zero;

We found that distance 1 is optimal for finding social ties on the given core nodes, and provided distributions of embeddedness and dispersion in figure 7. For more discussion on the choice of distance function, refer to question 15. The high absolute values of the dispersion observed in the figure should not confuse us, as for some nodes the number of neighbours shared with the core exceeds 250, with many of them having distance of 2 or being disconnected.

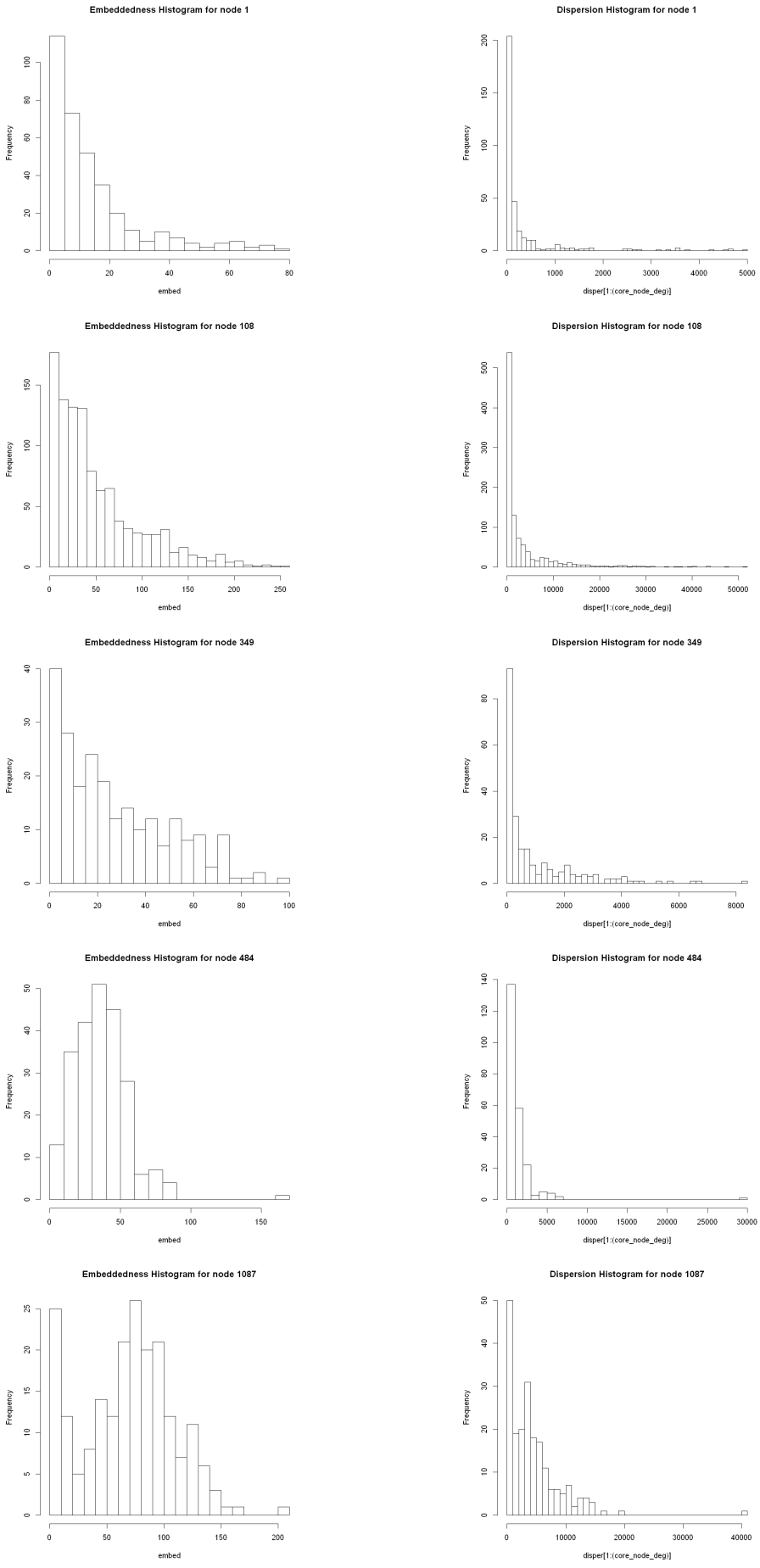


Figure 7: Embeddedness and Dispersion histogram.

QUESTION 13: For each of the core node's personalized network, plot the community structure of the personalized network using colors and highlight the node with maximum dispersion. Also, highlight the edges incident to this node. To detect the community structure, use Fast-Greedy algorithm. In this question, you will have 5 plots.

Answer: Nodes with the highest dispersion for personalized nets are shown in figure 8.

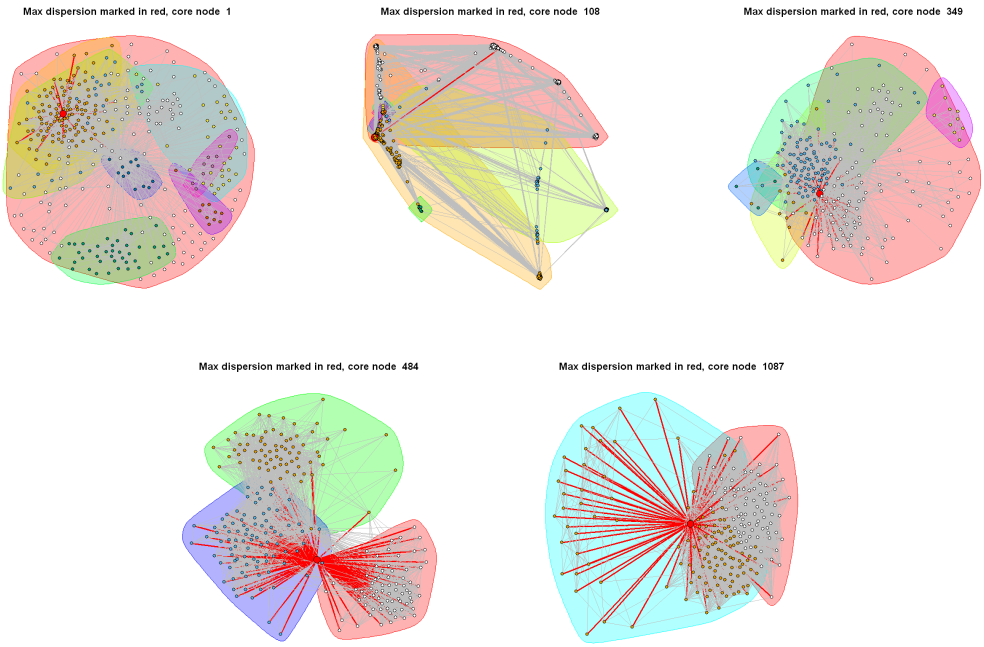


Figure 8: Highlighted node with maximum dispersion.

QUESTION 14: Repeat question 13, but now highlight the node with maximum embeddedness and the node with maximum $\frac{dispersion}{embeddedness}$ (excluding the nodes having zero embeddedness if there are any). Also, highlight the edges incident to these nodes. Report the id of those nodes.

Answer: Nodes with the highest metrics are are shown in table 3 and in figure 9.

Core node	Embeddedness	Dispersion/Embeddedness
1	57	323
108	1023	1023
349	33	33
484	1	1
1087	38	38

Table 3: Nodes with maximum embeddedness and maximum dispersion/embeddedness.

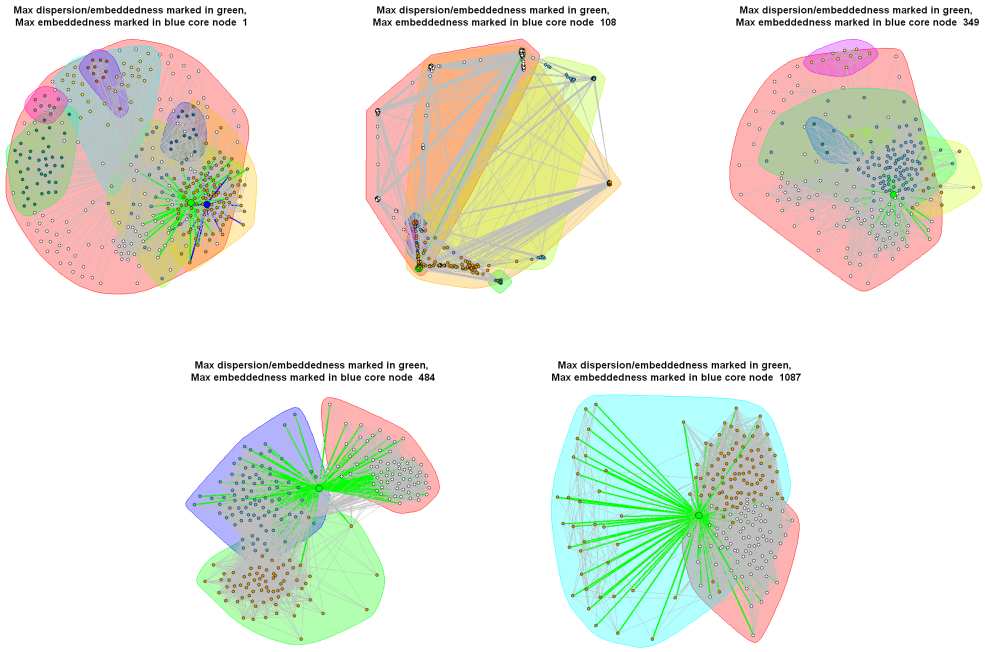


Figure 9: Highlighted node with maximum dispersion/embeddedness and embeddedness.

QUESTION 15: Use the plots from Question 13 and 14 to explain the characteristics of a node revealed by each of this measure.

Answer: Embeddedness selects the node with the most shared connections. While this sounds like a good choice to determine strong social ties, in practice the node with highest the embeddedness is just a popular member of a single community. In comparison, dispersion finds the person with friends distributed among many different communities. Generally this performs better than embeddedness for identifying close ties. However, sometimes it just idenitifies the person with the highest absolute number of connections, disregarding any structure in those connections. In order to measure quality of the connection, we divide our dispersion metric by our embeddedness to measure how disconnected these nodes are on average. This $\frac{dispersion}{embeddedness}$ metric allows us to identify people who share many friends from many different communities. In figure 8 we see that our nodes marked in red have edges that reach into many different communities. We would like this result to be enhanced by dividing it by embeddedness in figure 9. However, from the result of this figure it seems that it is not necessary. As we can see from the figure, in all of the cases except the case of node 1, nodes with the highest embeddedness are exactly those that have many friends across different communities, which is why we do not see much change. In case of core 1, it performs exactly as expected, though.

We also conducted experiments on how the choice of distance function for dispersion can affect which nodes the algorithm selects. We found that thresholding distance from the top (see function 2 in question 12) for different values of threshold does not change the nodes with the highest $\frac{dispersion}{embeddedness}$. In contrast, making distance binary (see function 3 in question 12) can result in a change for high values of threshold. However, distance function defined like this picks nodes with very small total number of shared connections, which is unlikely for a strong social connection. This is why we chose

function 1 in question 12 to do tasks 12 through 15.

4. Friend recommendation in personalized networks.

QUESTION 16: What is $|N_r|$, i.e. the length of the list N_r?

Answer: The length of the list N_r is 11. First the ego graph was created for node 415, and then all of the nodes with degree 24 were extracted.

QUESTION 17: Compute the average accuracy of the friend recommendation algorithm that uses: Common Neighbors measure, Jaccard measure, Adamic Adar measure. Based on the average accuracy values, which friend recommendation algorithm is the best?

Answer: An average accuracy was computed over 10 trials of each node in N_r, and then an average was taken over all nodes in N_r. The accuracies are provided in table 4. We can observe that using Academic Adar measure gives the best results.

	Common Neighbors	Jaccard	Adamic Adar
Average Accuracy	0.702	0.670	0.824

Table 4: Accuracy scores for friend recommendation algorithms.

2. Google+ network.

QUESTION 18: How many personal networks are there?

Answer: There are 57 networks with 2 or more circles

QUESTION 19: For the 3 personal networks (node ID given below), plot the in-degree and out-degree distribution of these personal networks. Do the personal networks have a similar in and out degree distribution? In this question, you should have 6 plots.

Answer:

As we can see from figure 10, in-degree distributions are quite different. While input degree distribution of network 101373961279443806744 resembles inverse x function, the same distribution for net 109327480479767108490 shows very low frequency for near 0 in-degree. Then we have net 115625564993990145546, which has large frequency for in-degrees up to 60 and then cuts off. The out degrees are more similar, as they all have large frequency near 0 as their defining trait.

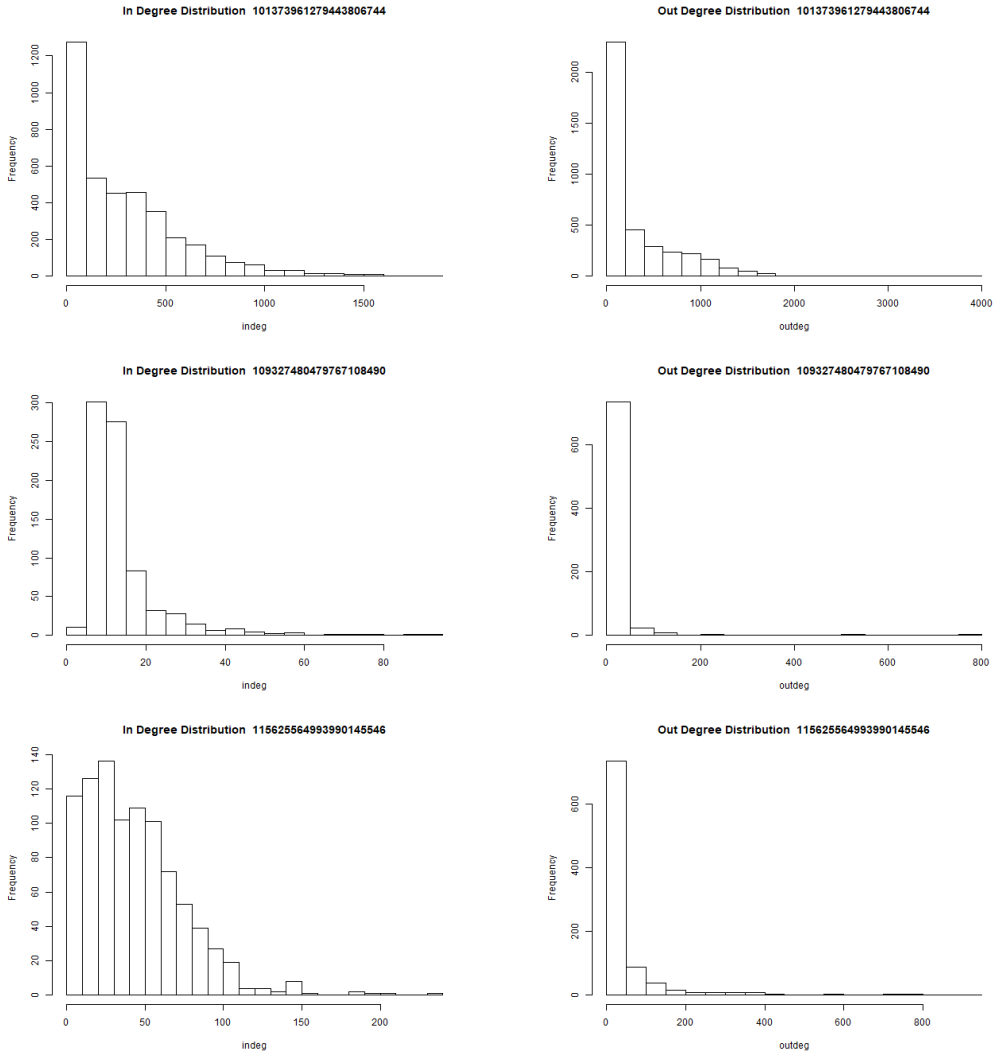


Figure 10: In- and out-degree distribution for 3 personal networks.

QUESTION 20: For the 3 personal networks picked in question 19, extract the community structure of each personal network using Walktrap community detection algorithm. Report the modularity scores and plot the communities using colors. Are the modularity scores similar? In this question, you should have 3 plots.

Answer: Modularity scores for 3 networks are provided in table 5. From that table, we can see that modularity is low and moderately varies among those nets. Figure **??** also suggests that walktrap algorithm is not very successfull in clustering those nets. The reason is the structure of these graphs: clusters are strongly connected with the core and with each other, so it is hard to separate them.

Core id	Modularity
101373961279443806744	0.19
109327480479767108490	0.25
115625564993990145546	0.32

Table 5: Modularity scores for Google+ nets.

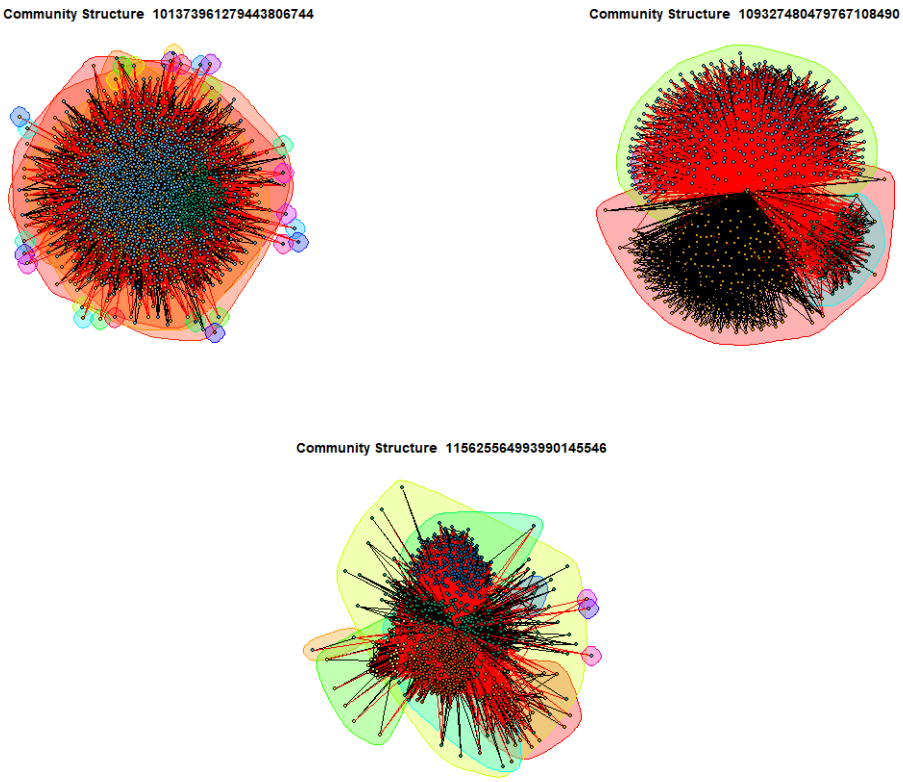


Figure 11: Community structures for Google+ nets.

QUESTION 21: Based on the expression for h and c, explain the meaning of homogeneity and completeness in words.

Answer: The **homogeneity** is a measure of how well clusters fit inside a circle. If we assign each circle with a label, this metric would show how different are labels inside the clusters. In the ideal case where each cluster has members only of 1 circle, homogenity equals to 1. **Completeness** is a measure of how fully clusters absorb circles. In the ideal case where each cluster accomodates all members of one or several circles, the completeness is 1.

QUESTION 22: Compute the h and c values for the community structures of the 3 personal network (same nodes as question 19). Interpret the values and provide a detailed explanation. Are there negative values? Why?

Answer: The homogenity and completeness values for the 3 nets are provided in table 6. We can see that depending on the net, walktrap algorithm was able to better or worse fit clusters inside the circles. The values of completeness are either low or even negative for all 3 nets. The reason behind this is how circles are defined. In general, when each node belongs to a single class and a single cluster, we expect both completeness and homogenity to be between 0 and 1. However, that is not the case for Google+ netwwork, where a single node can belong to 1, 2 or more circles. This increases $H(K|C)$ and shifts lower limit of c, which results in negative completeness.

Core id	h	c
101373961279443806744	0.004	−1.504
109327480479767108490	.852	0.329
115625564993990145546	0.452	−3.424

Table 6: Modularity scores for Google+ nets.

1-17

April 29, 2021

1 Import libs

[1]:
```
library('igraph')
library('Matrix')
library('pracma')
```

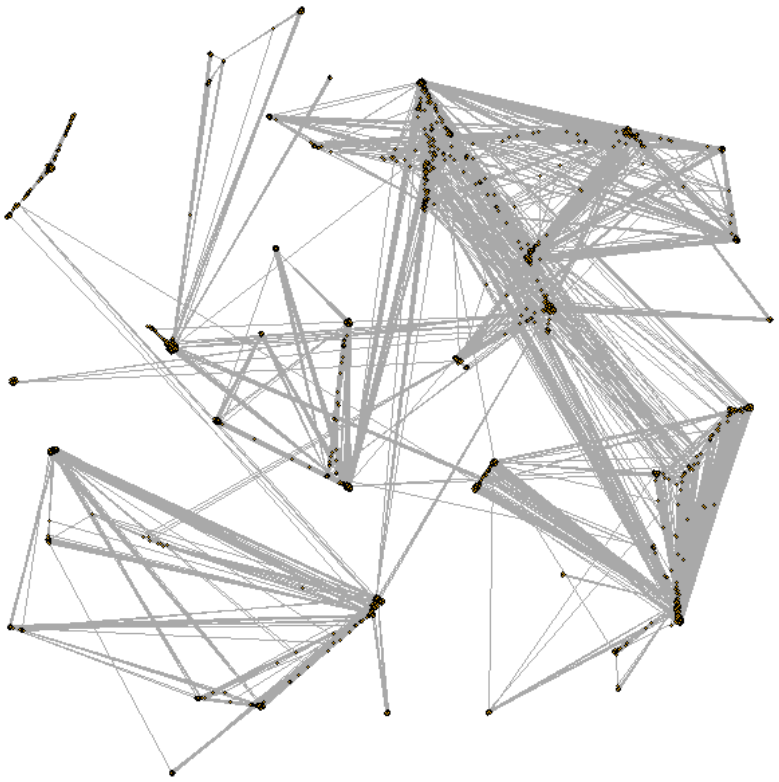
```
Attaching package: 'igraph'

The following objects are masked from 'package:stats':

    decompose, spectrum

The following object is masked from 'package:base':

    union


Attaching package: 'pracma'

The following objects are masked from 'package:Matrix':

    expm, lu, tril, triu
```

2 1. Structural properties of the Facebook network

QUESTION 1: A first look at the network:

[2]:
```
facebook <- read_graph("data/facebook_combined.txt", directed=FALSE)

# make plot
plot(facebook, vertex.size=1, vertex.label=NA)

# # save plot
# png(file="plots/q1.png", width=600, height=450)
# plot(facebook, vertex.size=1, vertex.label=NA)
```

```
# dev.off()
```



QUESTION 1.1: Report the number of nodes and number of edges of the Facebook network.

[4]:
```
print("number of vertices")
gorder(facebook)
print("number of edges")
gsize(facebook)
```

```
[1] "number of vertices"
```

4039

```
[1] "number of edges"
```

88234

QUESTION 1.2: Is the Facebook network connected? If not, find the giant connected component (GCC) of the network and report the size of the GCC.

[3]:
```
is.connected(facebook)
```

TRUE

As facebook graph is connected, GCC size is 4039

QUESTION 2: Find the diameter of the network. If the network is not connected, then find the diameter of the GCC.

[6]:
```
diameter(facebook)
```

8

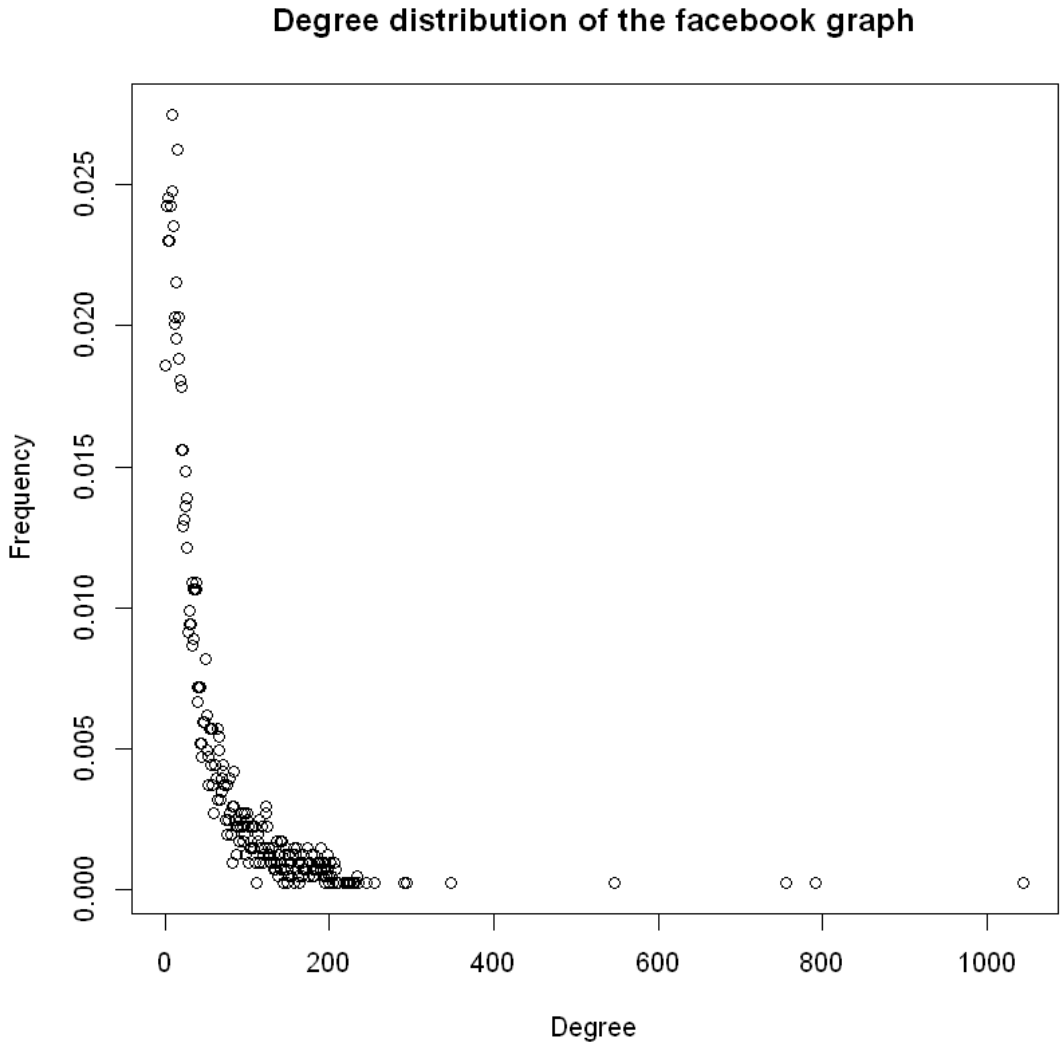
QUESTION 3: Plot the degree distribution of the facebook network and report the average degree.

[7]:
```
generate_mask <- function(vector_w_zeros) {
    # function that generates mask, which removes zero values from the vector

    # find which indices we need to remove
    to_remove <- which(vector_w_zeros == 0)
    # initialize mask
    mask <- c(ones(1, length(vector_w_zeros)))
    # mark elements we need to remove in the mask
    for (i in 1:length(vector_w_zeros)) {
        if (is.element(i, to_remove)) {
            mask[i] <- 0
        }
    }
    # convert mask to boolean
    mask <- mask > 0.5

    return(mask)
}
```

[8]:
```
# get x axis
degrees <- seq_along(degree.distribution(facebook)) - 1
# get y axis
distribution <- degree.distribution(facebook)
# get a mask to filter out zeros
mask <- generate_mask(distribution)

# plot for reference
plot(degrees[mask], distribution[mask],
     main="Degree distribution of the facebook graph",
```

```
     xlab="Degree",
     ylab="Frequency",)

# save plot
png(file="plots/q3.png", width=600, height=450)

plot(degrees[mask], distribution[mask],
     main="Degree distribution of the facebook graph",
     xlab="Degree",
     ylab="Frequency",)

dev.off()
```

png: 2



[9]:
```
# init average
average = 0
# for all nodes
for (i in 1:gorder(facebook)) {
    # add node's degree to average
    average <- average + degree(facebook, v=i)
}
# divide total by the number of nodes
average <- average / gorder(facebook)
print("Average degree")
print(average)
```

```
[1] "Average degree"
[1] 43.69101
```

QUESTION 4: Plot the degree distribution of Question 3 in a log-log scale. Try to fit a line to the plot and estimate the slope of the line.

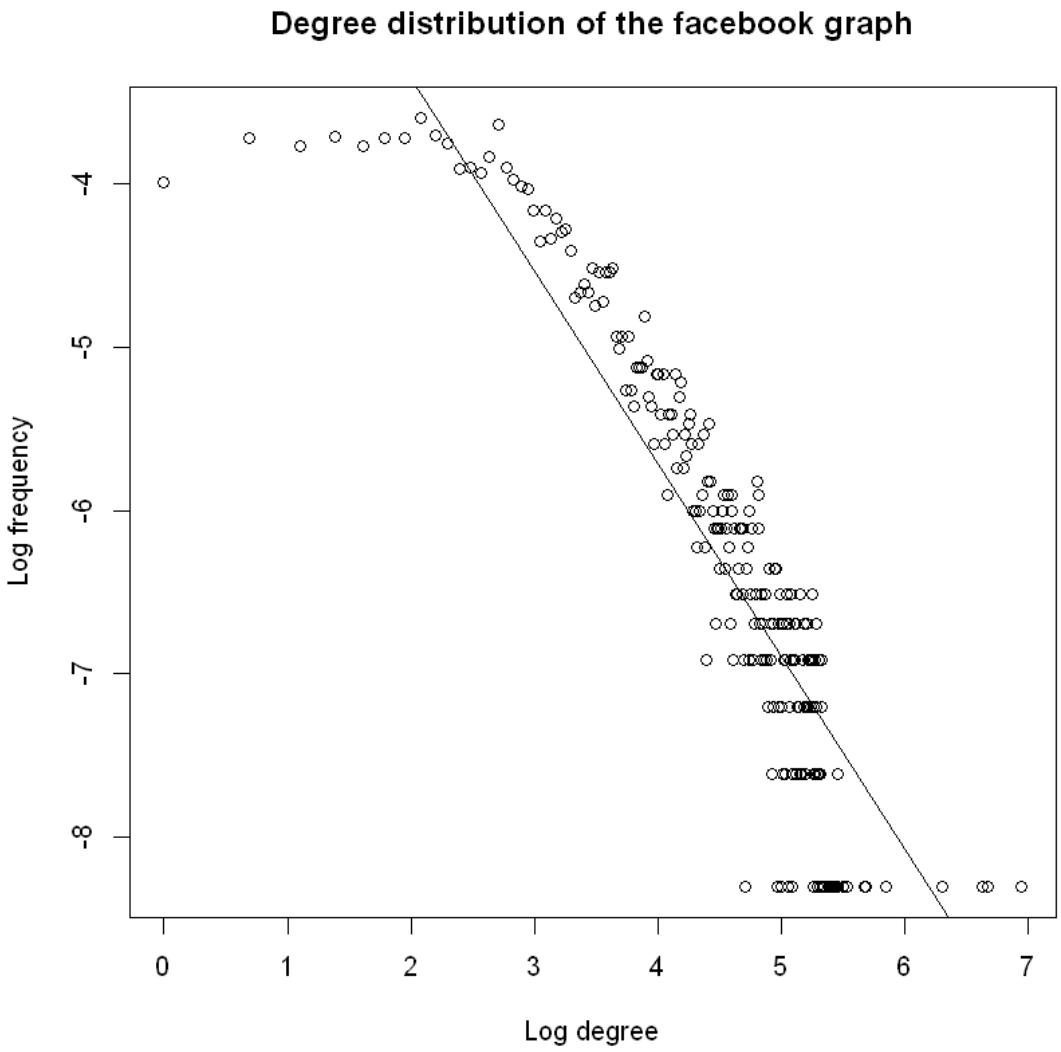
[10]:
```
# get degrees present in a net along with frequencies with which they appear
degrees <- seq_along(degree.distribution(facebook)) - 1
distribution <- degree.distribution(facebook)
# convert them to collections and then to matrices
X <- matrix(c(degrees), byrow=TRUE, nrow=1)
Y <- matrix(c(distribution), byrow=TRUE, nrow=1)
# delete entries with zero frequencies from both matrices
# this allows to avoid - infinity values after log scaling and makes sense
# as these data is not actually present in the net
indices = which(Y!=0,arr.ind = T)
X <- X[indices]
Y <- Y[indices]
# log scale data
X <- log(X)
Y <- log(Y)
# select how many elements you want to delete from the end to avoid outliners
delete <- 0
# calculate len of the desired array
len <- size(X)[2] - delete
# get the slices of both matrices
X <- X[0:len]
Y <- Y[0:len]
```

[11]:
```
# train linear regression model on the data
model = lm(Y ~ X)
```

[12]:
```
# plot for reference
plot(X, Y,
```

```
     main="Degree distribution of the facebook graph",
     xlab="Log degree",
     ylab="Log frequency",)
abline(model)

# save plot
png(file="plots/q4.png", width=600, height=450)

plot(X, Y,
     main="Degree distribution of the facebook graph",
     xlab="Log degree",
     ylab="Log frequency",)
abline(model)

dev.off()
```

png: 2



[13]:
```
# print model summary
# coefficient we are looking for is X estimate.
summary(model)
```

```
Call:
lm(formula = Y ~ X)

Residuals:
    Min      1Q  Median      3Q     Max
-2.9940 -0.3032  0.1697  0.4064  0.8927

Coefficients:
```

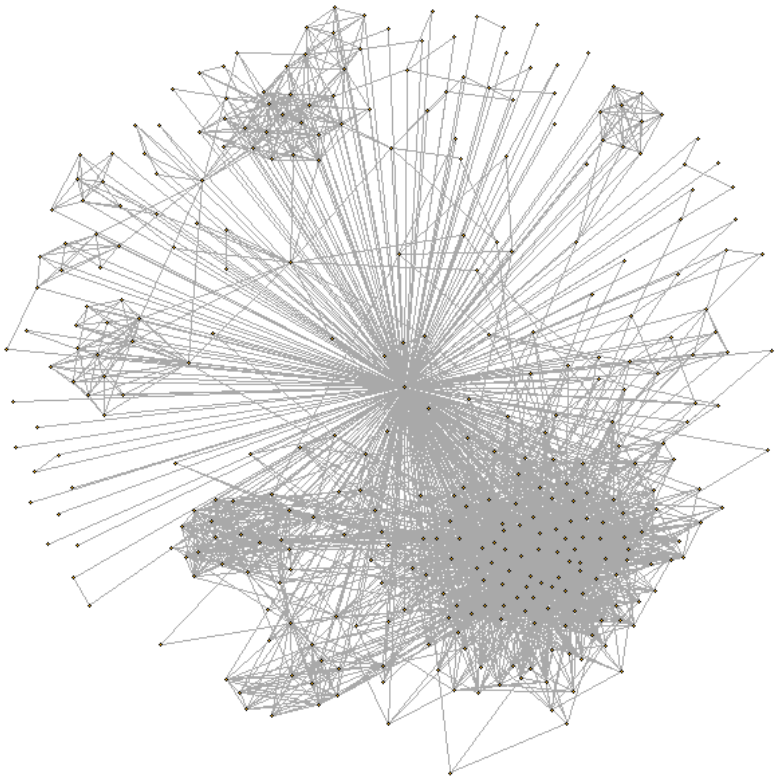
```
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.99222    0.17522  -5.663 4.52e-08 ***
X           -1.18016    0.03821 -30.889  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5772 on 225 degrees of freedom
Multiple R-squared:  0.8092,        Adjusted R-squared:  0.8083
F-statistic: 954.1 on 1 and 225 DF,  p-value: < 2.2e-16
```

We can see that facebook network distribution resembles power law distribution with $k \approx 1$

QUESTION 5: Create a personalized network of the user whose ID is 1. How many nodes and edges does this personalized network have?

[14]:
```r
id1 <- make_ego_graph(facebook, order = 1, nodes=1)[[1]]

# plot the personalized net
plot(id1, vertex.size=1, vertex.label=NA)

# save plot
png(file="plots/q5.png", width=600, height=450)
plot(id1, vertex.size=1, vertex.label=NA)
dev.off()
```

png: 2



[15]:
```
print("number of vertices")
gorder(id1)
print("number of edges")
gsize(id1)
```

```
[1] "number of vertices"
```

348

```
[1] "number of edges"
```

2866

QUESTION 6: What is the diameter of the personalized network? Please state a trivial upper and lower bound for the diameter of the personalized network.

[55]:
```
diameter(id1)
```

2

trivial - 0, if there is no vertices connected to the central vertex. Maximal - 2, as all vertices are connected to the center one.

QUESTION 7: In the context of the personalized network, what is the meaning of the diameter of the personalized network to be equal to the upper bound you derived in Question 6. What is the meaning of the diameter of the personalized network to be equal to the lower bound you derived in Question 6 (assuming there are more than 3 nodes in the personalized network)?

Assuming how question is asked, lower bound for net with > 3 nodes is 1, higher is 2. If diameter $==$ lower bound, then the net is fully connected meaning each node is connected with each. If diameter $==$ higher, then it is not.

QUESTION 8: How many core nodes are there in the Facebook network. What is the average degree of the core nodes?

[ ]:
```
# Save degree array of Facegbook graph in "sizes"
sizes = degree(facebook)
# For which of these indices is the degree > 200?
coreInd = which(sizes > 200)
print(length(coreInd))
# Take the average degree for these indices
avCoreDeg = mean(degree(facebook, v=V(facebook)[coreInd]))
print(avCoreDeg)
```

QUESTION 9: Node ID 1, Node ID 108, Node ID 349, Node ID 484, Node ID 1087. For each of the above core node's personalized network, find the community structure using Fast-Greedy, Edge-Betweenness, and Infomap community detection algorithms. Compare the modularity scores of the algorithms. For visualization purpose, display the community structure of the core node's personalized networks using colors. Nodes belonging to the same community should have the same color and nodes belonging to different communities should have different color. In this question, you should have 15 plots in total.

[ ]:
```
node_ids = c(1, 108, 349, 484, 1087)
# List of personalized networks matching the specified node ids
personalized_networks = make_ego_graph(facebook, 1, nodes=V(facebook)[node_ids])
# For each core node's personalized network, find the community structure using
# Fast-Greedy, Edge-Betweenness, Infomap
for (n in 1:5){
    # Find community structure using Fast-Greedy:
    cfg = cluster_fast_greedy(personalized_networks[[n]])
    cat("Fast-Greedy Modularity Score: ", modularity(cfg), "\n")
    # Plot community structure
    cfg_node_color <- cfg$membership -1
    plot(personalized_networks[[n]], mark.groups=groups(cfg), edge.arrow.size=.
↪4,
         vertex.color=cfg_node_color, vertex.size=3,vertex.label="",
```

```
    main=paste("Fast-Greedy, core node ",node_ids[n],collapse=""))

    # Find community structure using Edge-Betweenness:
    ceb = cluster_edge_betweenness(personalized_networks[[n]])
    cat("Edge-Betweenness Modularity Score: ", modularity(ceb), "\n")
    # Plot community structure
    ceb_node_color <- ceb$membership -1
    plot(personalized_networks[[n]], mark.groups=groups(ceb), edge.arrow.size=.
↪4,
         vertex.color=ceb_node_color, vertex.size=3,vertex.label="",
    main=paste("Edge-Betweenness, core node ",node_ids[n],collapse=""))

    # Find community structure using Infomap:
    cim = cluster_infomap(personalized_networks[[n]])
    cat("Infomap Modularity Score: ", modularity(cim), "\n")
    # Plot community structure
    cim_node_color <- cim$membership -1
    plot(personalized_networks[[n]], mark.groups=groups(cim), edge.arrow.size=.
↪4,
         vertex.color=cim_node_color, vertex.size=3,vertex.label="",
    main=paste("Infomap, core node ",node_ids[n],collapse=""))


}
```

QUESTION 10: For each of the core node's personalized network (use same core nodes as Question 9), remove the core node from the personalized network and find the community structure of the modified personalized network. Use the same community detection algorithm as Question 9. Compare the modularity score of the community structure of the modified personalized network with the modularity score of the community structure of the personalized network of Question 9. For visualization purpose, display the community structure of the modified personalized network using colors. In this question, you should have 15 plots in total.

[ ]:
```
for (n in 1:5){
    # get the degree distribution of the net
    deg <- degree(some_graph)
    # find core node
    core <- which.max(deg)
    # Delete the core node
    curr_network <- delete_vertices(personalized_networks[[n]], core)
    # Find community structure using Fast-Greedy:
    cfg = cluster_fast_greedy(curr_network)
    cat("Fast-Greedy Modularity Score: ", modularity(cfg), "\n")
    # Plot community structure
    cfg_node_color <- cfg$membership -1
    plot(curr_network, mark.groups=groups(cfg), edge.arrow.size=.4,
         vertex.color=cfg_node_color, vertex.size=3,vertex.label="",
```

```
    main=paste("Fast-Greedy, core node ",node_ids[n],collapse=""))

    # Find community structure using Edge-Betweenness:
    ceb = cluster_edge_betweenness(curr_network)
    cat("Edge-Betweenness Modularity Score: ", modularity(ceb), "\n")
    # Plot community structure
    ceb_node_color <- ceb$membership -1
    plot(curr_network, mark.groups=groups(ceb), edge.arrow.size=.4,
         vertex.color=ceb_node_color, vertex.size=3,vertex.label="",
    main=paste("Edge-Betweenness, core node ",node_ids[n],collapse=""))

    # Find community structure using Infomap:
    cim = cluster_infomap(curr_network)
    cat("Infomap Modularity Score: ", modularity(cim), "\n")
    # Plot community structure
    cim_node_color <- cim$membership -1
    plot(curr_network, mark.groups=groups(cim), edge.arrow.size=.4,
         vertex.color=cim_node_color, vertex.size=3,vertex.label="",
    main=paste("Infomap, core node ",node_ids[n],collapse=""))


}
```

QUESTION 11: Write an expression relating the Embeddedness between the core node and a non-core node to the degree of the non-core node in the personalized network of the core node.

See overleaf

QUESTION 12: For each of the core node's personalized network (use same core nodes as Question 9), remove the core node from the personalized network and find the community structure of the modified personalized network. Use the same community detection algorithm as Question 9. Compare the modularity score of the community structure of the modified personalized network with the modularity score of the community structure of the personalized network of Question 9. For visualization purpose, display the community structure of the modified personalized network using colors. In this question, you should have 15 plots in total.

[4]:
```
get_embeddedness <- function(some_graph) {
    # function that finds embdeddedness for each node except the core node (as␣
 ↪it is not defined for the core)

    # get the degree distribution of the net
    deg <- degree(some_graph)
    # account for the core node
    embs <- deg - 1
    # find core node
    core <- which.max(deg)
    # embeddedness for the core node is not defined
    embs <- embs[-core]
```

```
    return(embs)
}
```

[77]:
```
get_dispersion <- function(some_graph) {
    # function that finds dispersion for each node except the core node (as it␣
↪is not defined for the core)

    # declare dispersion
    dispersion <- c(rep(0, length(V(some_graph))))
    # as we will be working with subgraphs, we will need a stable␣
↪attributesthat will not change after
    # inducing subgraphs with removed vertices; thus, we make a "custom_id";␣
↪remember that this operation
    # does not yet change real id of nodes
    # signature: set_vertex_attr(graph, name, index = V(graph), value)
    marked_graph <- set_vertex_attr(some_graph, "custom_id", value=1:
↪length(V(some_graph)))
    # get degree distribution
    deg <- degree(marked_graph)
    # find true id of core node in the graph
    core <- which.max(deg)
    # for each node
    for (i in 1:length(V(marked_graph))) {
        # except the core
        if (i != core) {
            # find the neighbours of the node
            nbs <- neighbors(marked_graph, v=i)
            # delete the core from the neighbours
            nbs <- nbs[-which(nbs==core)]
            # get csutom id-s of neighbors
            nbs_custom_id <- vertex_attr(marked_graph, "custom_id", index = nbs)
            # get modified graph of personalized graph (article, page 3 on the␣
↪right)
            subnet <- delete_vertices(marked_graph, c(i, core))
            # get list of custom id-s of new subgraph
            subnet_custom_id <- vertex_attr(subnet, "custom_id")
            # get true id-s of neighbors in the subgraph
            nbs_subgraph <- which(subnet_custom_id %in% nbs_custom_id)
            # get distances for the neighbours
            dist <- distances(subnet, v = nbs_subgraph, to = nbs_subgraph)
            # calculate dispersion based on custom distance function
            dispersion[i] <- sum((dist >= 3) * 1) / 2
        }
    }
    # dispersion for the core node is not defined
    dispersion <- dispersion[-core]
    return(dispersion)
```

```
}
```

[ ]:
```
node_ids = c(1, 108, 349, 484, 1087)
# List of personalized networks matching the specified node ids
personalized_networks = make_ego_graph(facebook, 1,␣
↪nodes=V(facebook)[node_ids], mindist = 0)
# For each core node's personalized network, find the community structure using
# Fast-Greedy, Edge-Betweenness, Infomap
pair = matrix(data = 0, nrow = 2, ncol = 1)
for (n in 1:5){
    #plot(personalized_networks[[n]], vertex.size=1, vertex.label=NA)
    # Find community structure using Fast-Greedy:
    core_node = which.max(ego_size(personalized_networks[[n]], order = 1, nodes␣
↪= V(personalized_networks[[n]])))
    core_node_deg = ego_size(personalized_networks[[n]], order = 1, nodes =␣
↪core_node)
    embed = matrix(data = 0, nrow = core_node_deg, ncol = 1)
    disper = matrix(data = 0, nrow = core_node_deg, ncol = 1)
    core_neighbors = neighbors(personalized_networks[[n]],core_node)
    pair[1] = core_node
    #print(n)
    for (i in 1:(core_node_deg-2)){

        if(i != core_node){

            inter = intersection(core_neighbors,␣
↪neighbors(personalized_networks[[n]],i))
            #print(i)
            embed[i] = length(intersection(core_neighbors,␣
↪neighbors(personalized_networks[[n]],i)))









        }
        #print(inter)
    }
    embed[core_node] = 0
    hist(embed, breaks = 20, main = " ")
    title(main = paste("Embeddedness Histogram for node ", node_ids[n], sep =␣
↪""), font.main = 2)

}
```

```
[ ]: node_ids = c(1, 108, 349, 484, 1087)
     # List of personalized networks matching the specified node ids
     personalized_networks = make_ego_graph(facebook, 1,␣
      ↪nodes=V(facebook)[node_ids], mindist = 0)
     # For each core node's personalized network, find the community structure using
     # Fast-Greedy, Edge-Betweenness, Infomap
     pair = matrix(data = 0, nrow = 2, ncol = 1)
     max_disp = matrix(data = 0, nrow = 5, ncol = 1)
     max_emb = matrix(data = 0, nrow = 5, ncol = 1)
     max_both = matrix(data = 0, nrow = 5, ncol = 1)
     for (n in 1:5){

         #plot(personalized_networks[[n]], vertex.size=1, vertex.label=NA)
         # Find community structure using Fast-Greedy:
         core_node = which.max(ego_size(personalized_networks[[n]], order = 1, nodes␣
      ↪= V(personalized_networks[[n]])))
         core_node_deg = ego_size(personalized_networks[[n]], order = 1, nodes =␣
      ↪core_node)
         embed = matrix(data = 0, nrow = core_node_deg, ncol = 1)
         disper = matrix(data = 0, nrow = core_node_deg, ncol = 1)
         core_neighbors = neighbors(personalized_networks[[n]],core_node)
         pair[1] = core_node
         #print(n)
         for (i in 1:(core_node_deg)){

             if(i != core_node){

                 inter = intersection(core_neighbors,␣
      ↪neighbors(personalized_networks[[n]],i))
                 #print(i)
                 embed[i] = length(intersection(core_neighbors,␣
      ↪neighbors(personalized_networks[[n]],i)))
                 #print(inter)
                 pair[2] = i
                 curr_vert = V(personalized_networks[[n]])
                 remove_vert = setdiff(curr_vert,inter)
                 curr_network <- delete_vertices(personalized_networks[[n]],␣
      ↪remove_vert)

                 new_vert = V(curr_network)
                 #print(distances(curr_network,i,V(curr_network)))
                 #print(core_node_deg)
                 dist = distances(curr_network, new_vert, new_vert)

                 for (k in 1:length(dist)){
                     if (dist[k] == Inf && length(dist) > 0){
                         dist[k] = 2
```

```
                }
            }

            disper[i] = sum(dist)
            #print(i)

        }
        #print(inter)
    }
    #print((disper))
    for (j in 1:length(disper)){
                if (is.na(disper[j])){
                    disper[j] = 0
                }
            }

    print(length(disper))
    embed[core_node] = 0
    hist(embed, breaks = 20, main = " ")
    title(main = paste("Embeddedness Histogram for node ", node_ids[n], sep =␣
↪""), font.main = 2)

    max_disp[n] = which.max(disper)
    hist(disper[1:(core_node_deg)], breaks = 40, main = " ", )
    title(main = paste("Dispersion Histogram for node ", node_ids[n], sep =␣
↪""), font.main = 2)

    max_emb[n] = which.max(embed)
    both = disper/embed
    #print(both)
    max_both[n] = which.max(both)
}
```

Question 13 For each of the core node's personalized network, plot the community structure of the personalized network using colors and highlight the node with maximum dispersion. Also, highlight the edges incident to this node. To detect the community structure, use Fast-Greedy algorithm. In this question, you will have 5 plots.

[ ]:
```
for (n in 1:5){
    # Find community structure using Fast-Greedy:
    g = personalized_networks[[n]]
    ind = matrix(data = 1, nrow = length(V(g)), ncol = 1)
    ind[max_disp[n]] = 2
    cfg = cluster_fast_greedy(g)
    # Plot community structure
    cfg_node_color <- cfg$membership -1
    V(g)$color <- ifelse(V(g) == max_disp[n], "red", cfg_node_color)
```

```
    #E(g)[incident_edges(g,max_disp[n])]$color <-  "red"
    #E(g)$color <- incident_edges(g,max_disp[n]),
    neigh = neighbors(g,max_disp[n])
    edges = matrix(data = max_disp[n], nrow =␣
↪2*length(neighbors(g,max_disp[n])))
    for (i in 1:length(neigh)){
        edges[2*i] = neigh[i]
    }
    #print(edges)
    ei <- get.edge.ids(g, edges)
    #print(ei)
    E(g)$color <- "grey"
    E(g)[ei]$color <- "red"
    E(g)$width <- 1
    E(g)[ei]$width <- 3

    #print(incident_edges(g,max_disp[n]) )
    V(g)[max_disp[n]]$color<-"red"
    plot(personalized_networks[[n]], mark.groups=groups(cfg), edge.color =␣
↪E(g)$color,
         vertex.color=V(g)$color, vertex.size=ind*3, vertex.label="", edge.
↪width = E(g)$width,
    main=paste("Max dispersion marked in red, core node␣
↪",node_ids[n],collapse=""))
```

Question 14 Repeat Question 13, but now highlight the node with maximum embeddedness and the node with maximum dispersion embeddedness (excluding the nodes having zero embeddedness if there are any). Also, highlight the edges incident to these nodes. Report the id of those nodes.

[ ]:
```
for (n in 1:5){
    # Find community structure using Fast-Greedy:
    g = personalized_networks[[n]]
    ind = matrix(data = 1, nrow = length(V(g)), ncol = 1)
    ind[max_emb[n]] = 2
    ind[max_both[n]] = 2
    cfg = cluster_fast_greedy(g)
    # Plot community structure
    cfg_node_color <- cfg$membership -1
    V(g)$color <- ifelse(V(g) == max_emb[n], "blue", cfg_node_color)
    V(g)[max_both[n]]$color<-"green"
    neigh = neighbors(g,max_both[n])
    edges = matrix(data = max_both[n], nrow =␣
↪2*length(neighbors(g,max_both[n])))
    for (i in 1:length(neigh)){
        edges[2*i] = neigh[i]
    }
    #print(edges)
```

```
    ei <- get.edge.ids(g, edges)

    neigh1 = neighbors(g,max_emb[n])
    edges1 = matrix(data = max_emb[n], nrow = 2*length(neighbors(g,max_emb[n])))
    for (i in 1:length(neigh1)){
        edges1[2*i] = neigh1[i]
    }
    #print(edges)
    ei1 <- get.edge.ids(g, edges1)

    #print(ei)

    E(g)$color <- "grey"

    E(g)[ei1]$color <- "blue"
    E(g)[ei]$color <- "green"
    E(g)$width <- 1
    E(g)[ei]$width <- 3
    E(g)[ei1]$width <- 3

    #print(incident_edges(g,max_disp[n]) )
    plot(g, mark.groups=groups(cfg), edge.color = E(g)$color,
         vertex.color=V(g)$color, vertex.size=ind*3, vertex.label="", edge.
 ↪width = E(g)$width,
         main=paste("Max dispersion/embeddedness marked in green,\n Max␣
 ↪embeddedness marked in blue core node ",node_ids[n],collapse=""))
}
```

Question 15 answer on overleaf

Having defined the friend recommendation procedure, we can now apply it to the personalized network of node ID 415. Before we apply the algorithm, we need to create the list of users who we want to recommend new friends to. We create this list by picking all nodes with degree 24. We will denote this list as Nr.

QUESTION 16: What is |Nr|, i.e. the length of the list Nr?

[ ]:
```
# Make personalized network of node ID 415
p_net_415 <- make_ego_graph(facebook, order = 1, nodes=V(facebook)[415])[[1]]
# Make a list of all nodes with degree 24
sizes = degree(p_net_415)
ids_24 = which(sizes == 24)
print(length(ids_24))
print(ids_24)
```

QUESTION 17: Compute the average accuracy of the friend recommendation algorithm that uses:

- Common Neighbors measure

- Jaccard measure
- Adamic Adar measure

Based on the average accuracy values, which friend recommendation algorithm is the best?

Hint Useful function(s): similarity

[ ]:
```
recommend_common_neighbors = function (s_i, g, num_rec){
#     1. For each node in the network that is not a neighbor of i, compute the␣
 ↪jaccard measure
# between the node i and the node not in the neighborhood of i
# Compute Jaccard(i, j)  j   S
# C
# i
# 2. Then pick t nodes that have the highest Jaccard measure with node i and␣
 ↪recommend
# these nodes as friends to node i

    # For each node in g that is not a neighbor of i (Remove vertices in s_i␣
 ↪from g)
    # Compute the common neighbors similarity between S_i
    all_vertices = V(g)
    # Subtracts s_i from the set of all vertices
    remove_neighbors = setdiff(all_vertices, s_i)
    #cat("Vertices without neighbors: ", remove_neighbors, "\n")
    num_neighs = c()
    for (v in remove_neighbors){
        # v's neighbors are the set s_j
        s_j = neighbors(g,v)
        num_common_neighbors = length(intersect(s_i, s_j))
        num_neighs = append(num_neighs, num_common_neighbors)
    }
    #cat("num_neighs", num_neighs)
    top_indices = order(num_neighs, decreasing=TRUE)[1:num_rec]
    #cat("Top indices: ", top_indices, "\n")
    recs = remove_neighbors[top_indices]

    return(recs)
}
```

[ ]:
```
recommend_jaccard = function(node_i, s_i, g, num_rec){
    all_vertices = V(g)
    remove_neighbors = setdiff(all_vertices, s_i)
    #cat("Vertices without neighbors: ", remove_neighbors, "\n")
    #cat("Node i id: ", node_i, "\n")
    jaccards = c()
    # This g is the whole graph
    jaccard_matrix = similarity(g, method='jaccard')
```

```
    for (v in remove_neighbors){
        jaccard = jaccard_matrix[node_i,v]
        jaccards = append(jaccards, jaccard)
    }
    #cat("Jaccard score list: ", jaccards, "\n")
    top_indices = order(jaccards, decreasing=TRUE)[1:num_rec]
    #cat("Top indices: ", top_indices, "\n")
    recs = remove_neighbors[top_indices]
    return(recs)
}
```

[ ]:
```
recommend_adamic_adar = function(node_i, s_i, g, num_rec){
    all_vertices = V(g)
    remove_neighbors = setdiff(all_vertices, s_i)
    #cat("Vertices without neighbors: ", remove_neighbors, "\n")
    #cat("Node i id: ", node_i, "\n")
    adamic_adars = c()
    # This g is the whole graph
    # According to documentation invlogweighted is Adamic-Adar (I think)
    adamic_adar_matrix = similarity(g, method='invlogweighted')
    for (v in remove_neighbors){
        adamic_adar = adamic_adar_matrix[node_i,v]
        adamic_adars = append(adamic_adars, adamic_adar)
    }
    top_indices = order(adamic_adars, decreasing=TRUE)[1:num_rec]
    #cat("Top indices: ", top_indices, "\n")
    recs = remove_neighbors[top_indices]
    return(recs)
}
```

[ ]:
```
scores_c = c()
scores_j = c()
scores_a = c()
for (node in ids_24){
# Remove each edge of node i at random with probability 0.25

    #cat("Printing node ", node, "\n")
    for (iter in c(1:10)){
        #cat("Iteration ", iter, "\n")
        p_net_dropped_nodes = p_net_415
        # List of dropped nodes
        R_i = c()
        # Iterate through node's neighbors
        node_neighbors = neighbors(p_net_415,node)
        #cat("Node ", node, " neighbor list: ", node_neighbors, "\n")
        # Drop each neighbor with p=0.25
        for (n in node_neighbors){
```

```
            # If the random sample is less than 0.25
            if (runif(1, 0, 1) <= 0.25){
                # Drop the node
                p_net_dropped_nodes = delete_edges(p_net_dropped_nodes, edge(n,␣
↪node))
                # And add it to the list of dropped nodes
                R_i = append(R_i, n)
                #print(n)
                #cat("Dropped node ", n, "\n")
            }
        }
        node_neighbors = setdiff(node_neighbors, R_i)
        #node_neighbors = append(node_neighbors, node)
        #Use one of the three neighborhood based measures to recommend |Ri| new␣
↪friends to the user i.
        num_new_friends = length(R_i)
        #cat("Number of dropped nodes/new friends: ", num_new_friends, "\n")
        #node_neighbors = append(node_neighbors, node)
        # Dropped edges or original?
        # Common neighbors recommendation
        recs_c =␣
↪recommend_common_neighbors(node_neighbors,p_net_dropped_nodes,num_new_friends)
        # Jaccard recommendation
        recs_j =␣
↪recommend_jaccard(node,node_neighbors,p_net_dropped_nodes,num_new_friends)
        # Adamic-Adar recommendation
        recs_a =␣
↪recommend_adamic_adar(node,node_neighbors,p_net_dropped_nodes,num_new_friends)
        #cat("Recs: ", recs, "\n")
        #cat("dropped nodes: ", R_i, "\n")
        #print(recs)
        #print(R_i)
        pi_intersect_ri_c = length(intersect(recs_c, R_i))
        pi_intersect_ri_j = length(intersect(recs_j, R_i))
        pi_intersect_ri_a = length(intersect(recs_a, R_i))

        score_c = pi_intersect_ri_c / num_new_friends
        score_j = pi_intersect_ri_j / num_new_friends
        score_a = pi_intersect_ri_a / num_new_friends

        scores_c = append(scores_c, score_c)
        scores_j = append(scores_j, score_j)
        scores_a = append(scores_a, score_a)

    }

}
```

```
print(mean(scores_c))
print(mean(scores_j))
print(mean(scores_a))
```

18-22

April 29, 2021

1 Import libs

[1]:
```
library('igraph')
library('Matrix')
library('pracma')
```

```
Warning message:
"package 'igraph' was built under R version 3.6.3"
Attaching package: 'igraph'

The following objects are masked from 'package:stats':

    decompose, spectrum

The following object is masked from 'package:base':

    union

Warning message:
"package 'Matrix' was built under R version 3.6.3"Warning message:
"package 'pracma' was built under R version 3.6.3"
Attaching package: 'pracma'

The following objects are masked from 'package:Matrix':

    expm, lu, tril, triu
```

Create directed personal networks for users who have more than 2 circles. The data required to create such personal networks can be found in the file named gplus.tar.gz. **QUESTION 18:** How many personal networks are there?

[2]:
```
path_to_files = "data/gplus/gplus"
#list of all files ending w circles
gpluscircles = list.files(path=path_to_files, pattern="*.circles")
count = 0
#for each circle file, read + check if > 2 lines.
for (circle in gpluscircles){
```

```
    circle_path = paste(path_to_files, circle, sep="/")
    count = count + (length(readLines(circle_path)) > 2)*1

}
print(count)
```

```
[1] 57
```

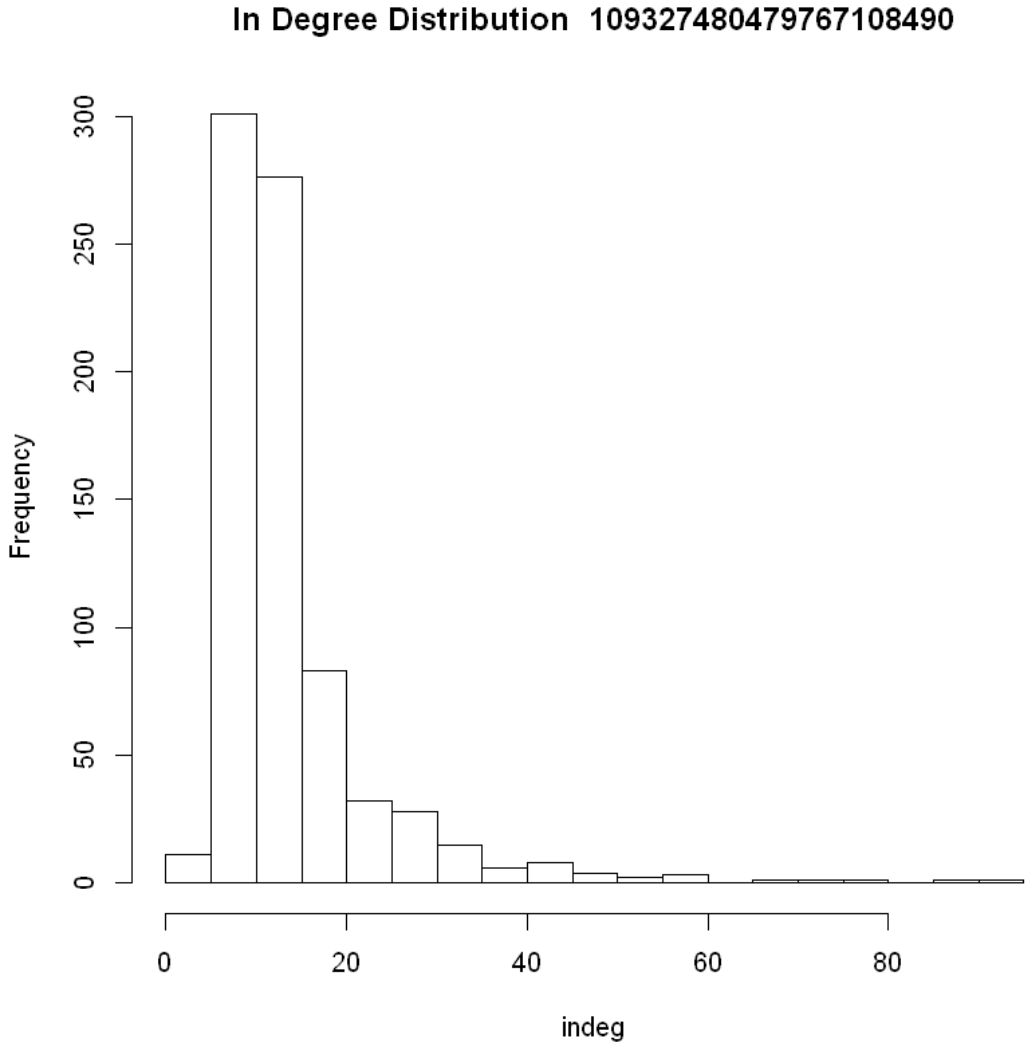
QUESTION 19: For the 3 personal networks (node ID given below), plot the in-degree and outdegree distribution of these personal networks. Do the personal networks have a similar in and out degree distribution? In this question, you should have 6 plots. • 109327480479767108490 • 115625564993990145546 • 101373961279443806744

QUESTION 20: For the 3 personal networks picked in Question 19, extract the community structure of each personal network using Walktrap community detection algorithm. Report the modularity scores and plot the communities using colors. Are the modularity scores similar? In this question, you should have 3 plots.

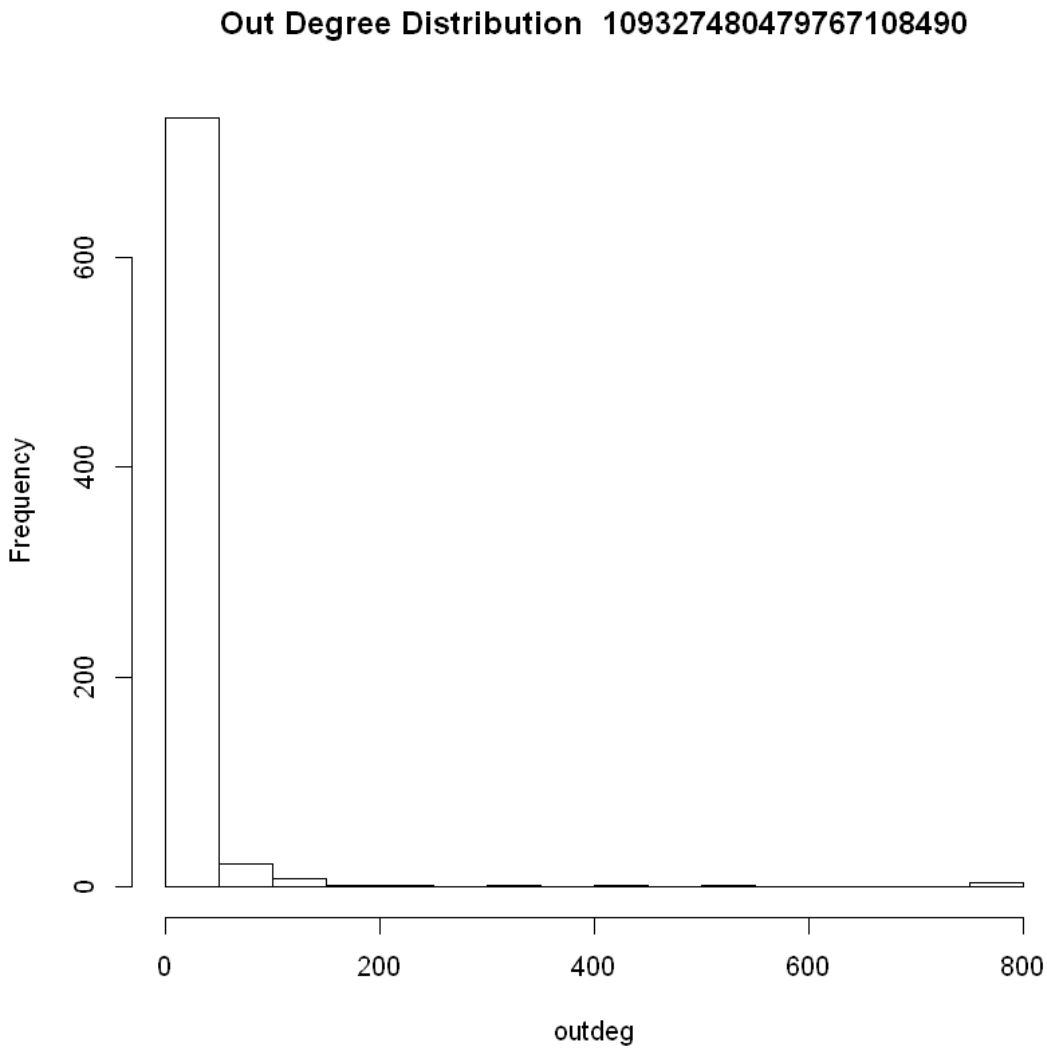
[26]:
```
nets = c("109327480479767108490", "115625564993990145546",␣
 ↪"101373961279443806744")
comms = c()
for (net1 in nets){
    #net1 = "109327480479767108490"
    net1_path = paste(path_to_files, "/", net1, ".edges", sep = "")
    print(net1_path)
    #Read in graph from file
    graph1 = read_graph(net1_path, format="ncol", directed=TRUE)
    #Add vertices for each entry
    graph1 = add.vertices(graph1, nv = 1, name = net1)

    edges = c()
    for (i in seq(1, vcount(graph1)-1, 1)){
        #vcount(graph1) corresponds to vertex "net1"
        #So create an edge from you to any other node
        edges = c(edges, c(vcount(graph1),i))
    }

    #add edges, then plot in/out degrees
    graph1 = add_edges(graph1, edges)
    indeg = degree(graph1, mode = "in")
    hist(indeg, main = paste("In Degree Distribution ", net1), breaks=25)
    outdeg = degree(graph1, mode = "out")
    hist(outdeg, main = paste("Out Degree Distribution ", net1), breaks=25)

    png(file=paste("plots/q20_", net1, "indeg.png", sep=""), width=600,␣
 ↪height=450)
    hist(indeg, main = paste("In Degree Distribution ", net1), breaks=25)
    dev.off()
```

```
    png(file=paste("plots/q20_", net1, "outdeg.png", sep=""), width=600,␣
↪height=450)
    hist(outdeg, main = paste("Out Degree Distribution ", net1), breaks=25)
    dev.off()

    community  = walktrap.community(graph1)
    modularity = modularity(community)

    print(paste(net1, " modularity is ", modularity))
    #edge.arrow.mode=0, edge.lty=0,
    plot(community, graph1, main=paste("Community Structure ", net1), edge.
↪arrow.size=.4,  vertex.size=3, vertex.label="", edge.arrow.mode=0)
    png(file=paste("plots/q21_", net1, "comm.png", sep=""), width=600,␣
↪height=450)
    plot(community, graph1, main=paste("Community Structure ", net1), edge.
↪arrow.size=.4,  vertex.size=3, vertex.label="", edge.arrow.mode=0)
    dev.off()
    comms = c(comms, community)
}
```

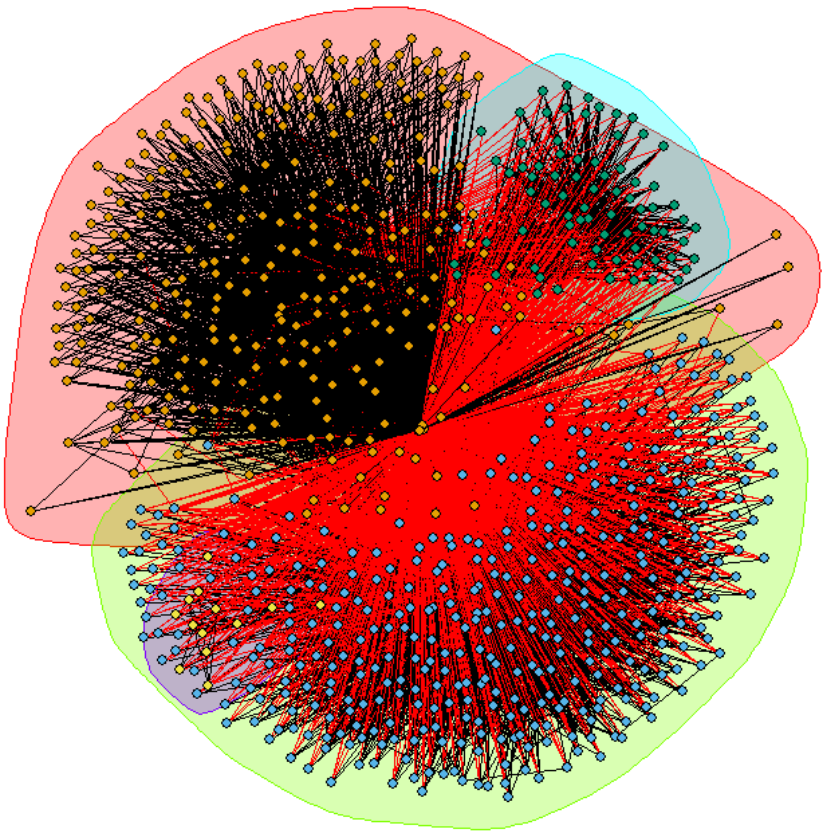
```
[1] "data/gplus/gplus/109327480479767108490.edges"
```

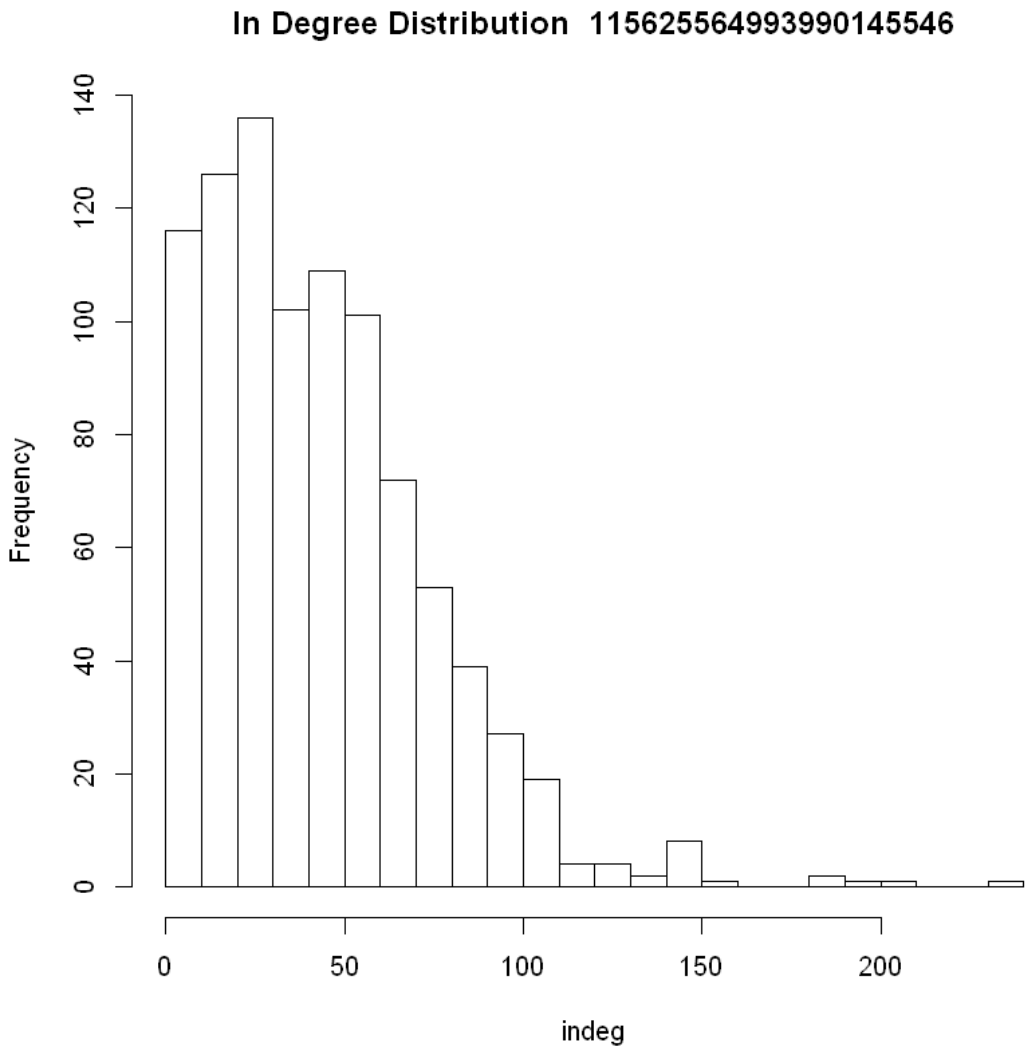



```
[1] "109327480479767108490  modularity is  0.252765387296677"
```

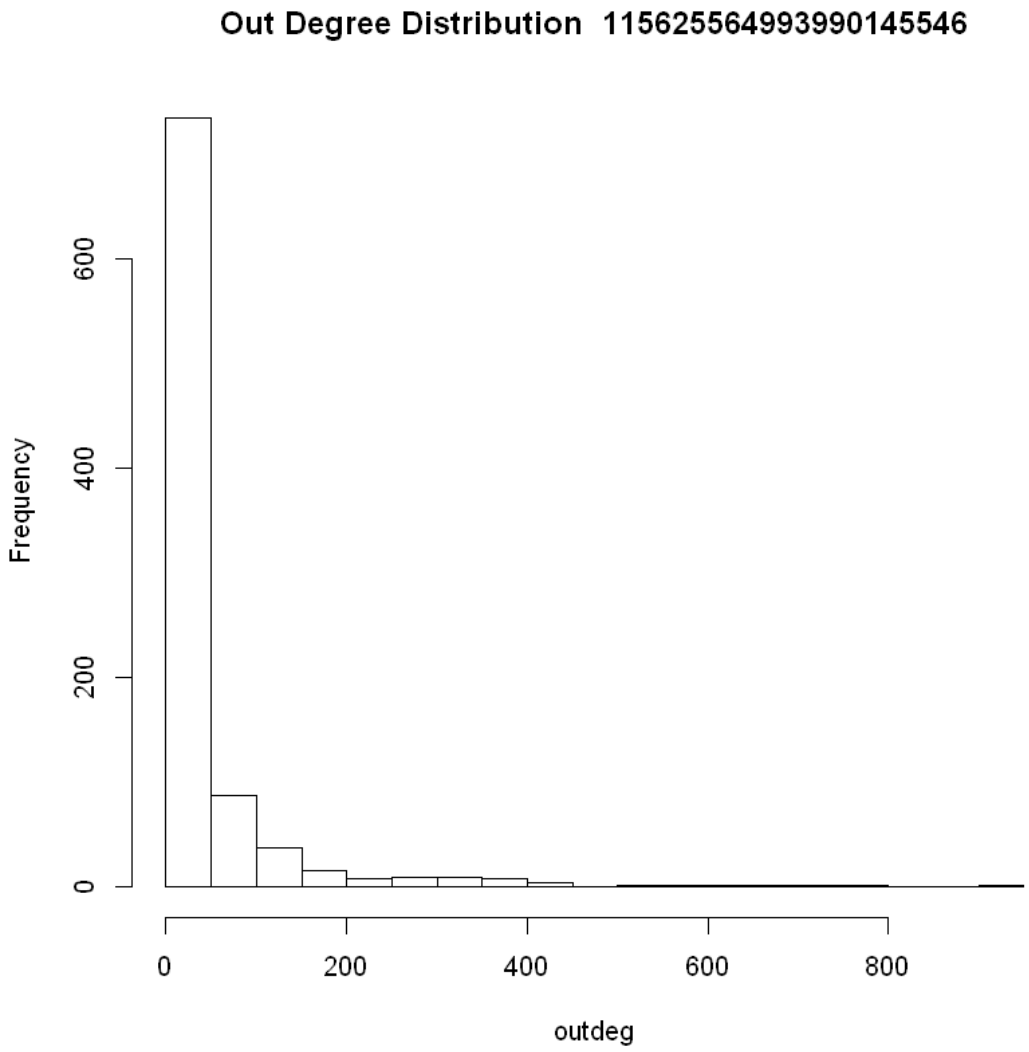



```
[1] "data/gplus/gplus/115625564993990145546.edges"
```

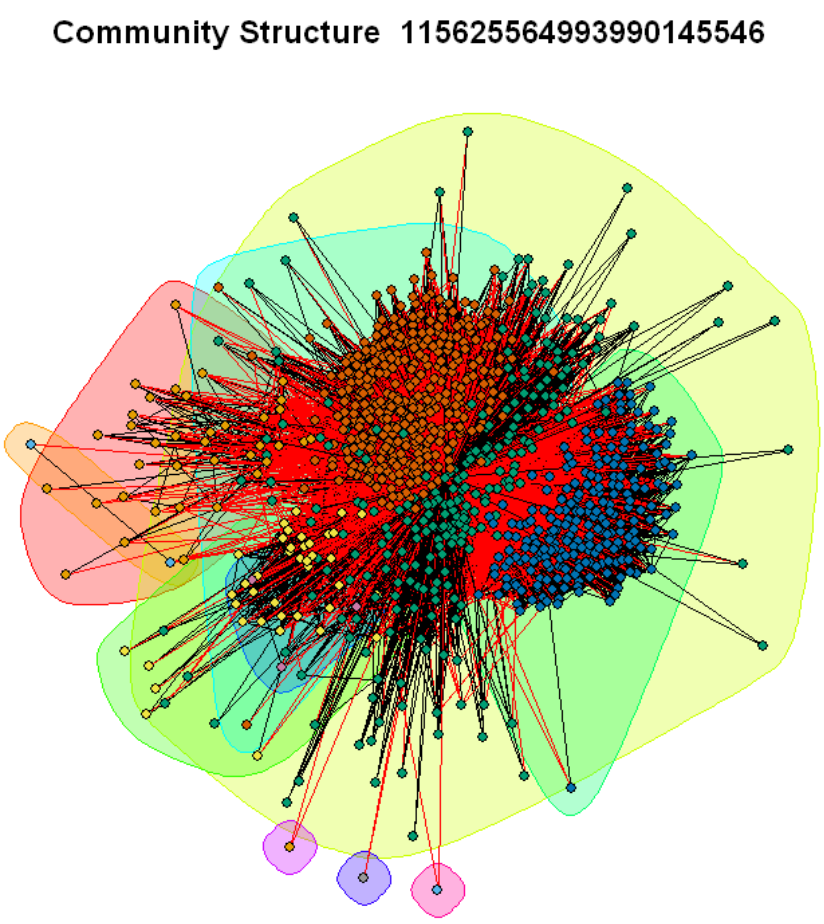

Community Structure 109327480479767108490

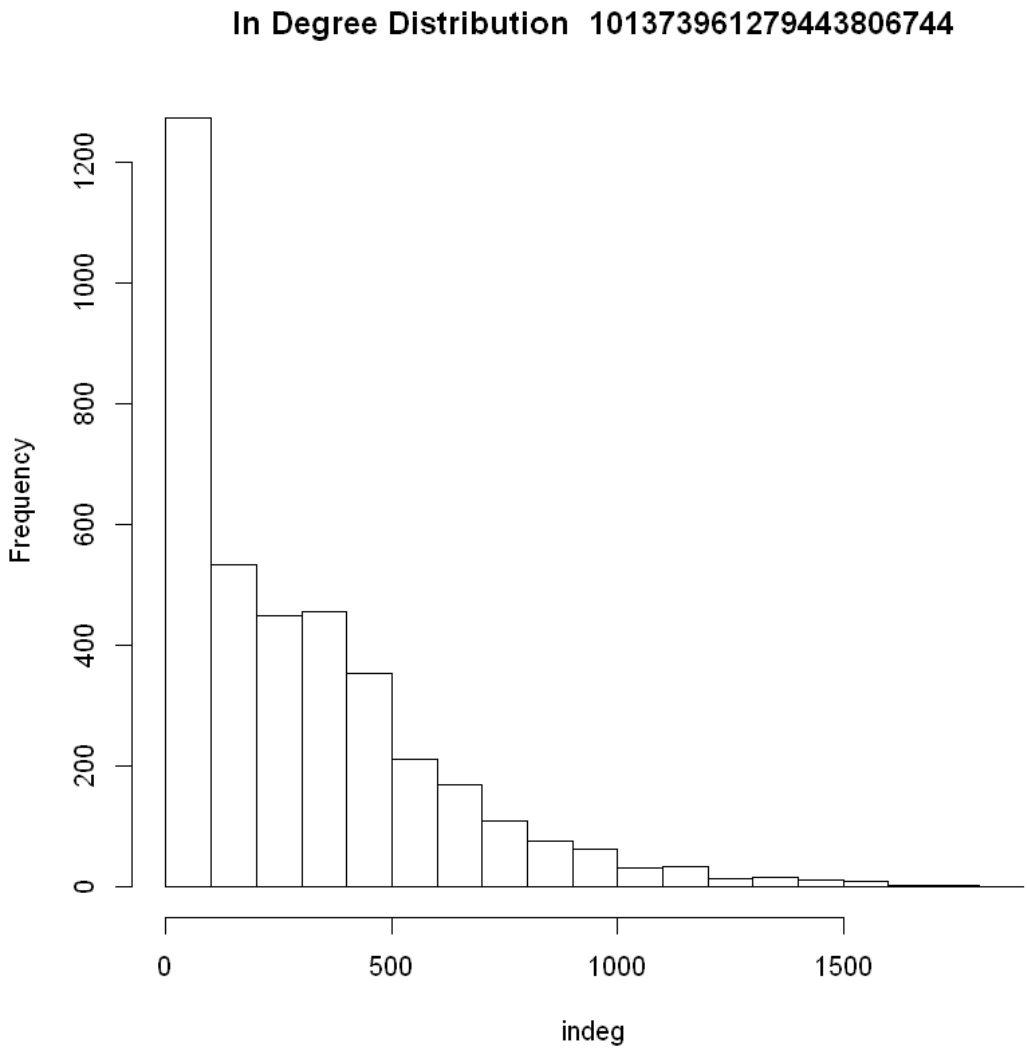



```
[1] "115625564993990145546  modularity is  0.319472551345825"
```

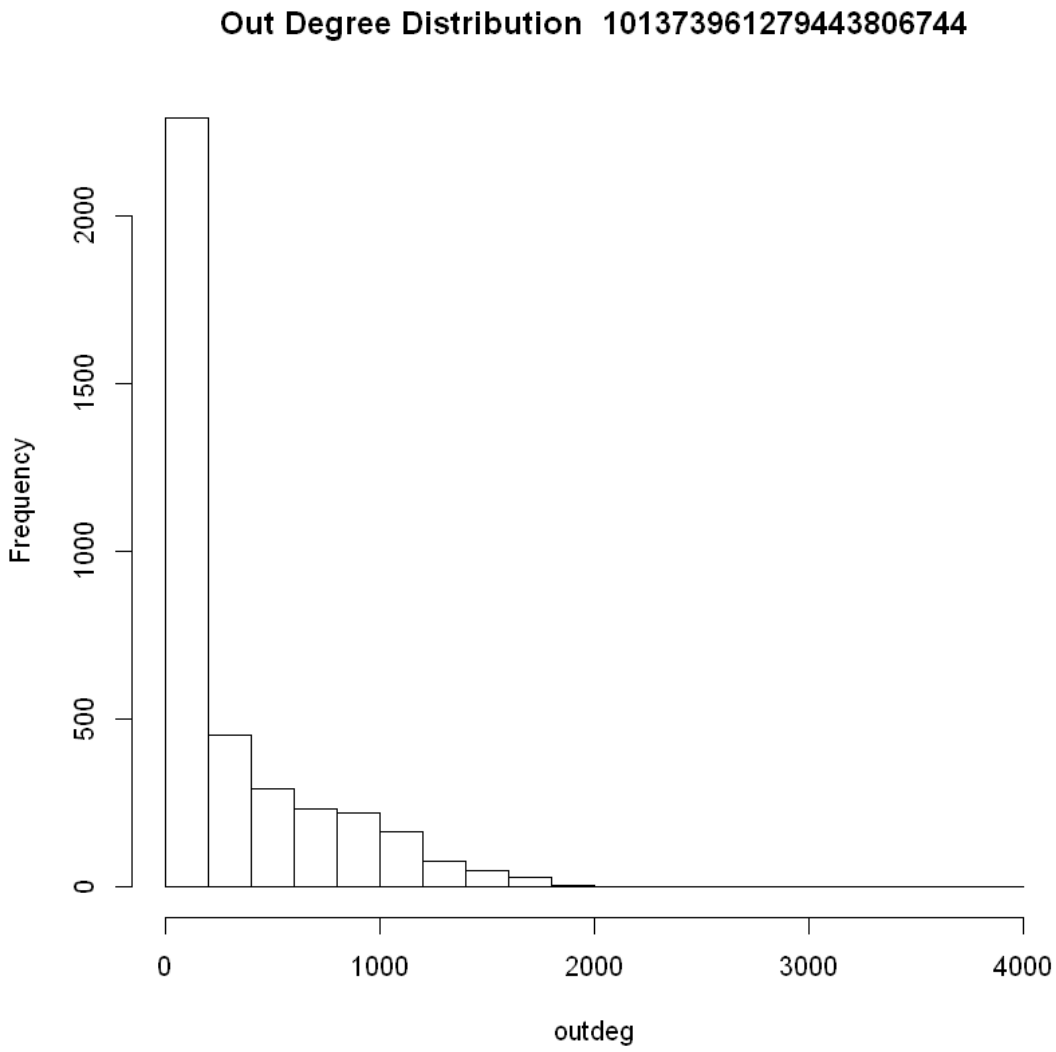


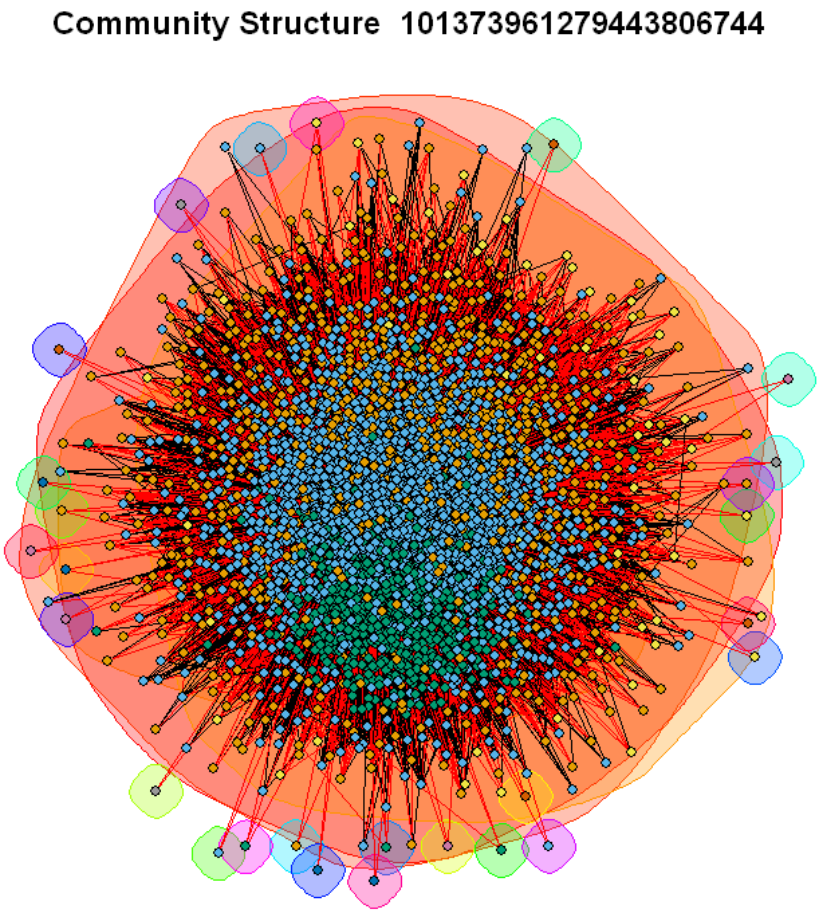

```
[1] "data/gplus/gplus/101373961279443806744.edges"
```


Community Structure 115625564993990145546




```
[1] "10137396127944380674  modularity is  0.191090270876884"
```


Out Degree Distribution 101373961279443806744
Frequency
0
500
1000
1500
2000
0
1000
2000
3000
4000
outdeg



QUESTION 22: Compute the h and c values for the community structures of the 3 personal network (same nodes as Question 19). Interpret the values and provide a detailed explanation. Are there negative values? Why?

[6]:
```
nets = c("109327480479767108490", "115625564993990145546",␣
 ↪"101373961279443806744")
for (net in nets){
    circle_path = paste(path_to_files, "/", net, ".circles", sep = "")
    print(circle_path)
    circles_lines = readLines(circle_path)
    print("circles lines")
    #print(circles_lines)
```

```
    #circles will be an array where each entry is a circle (and each circle has␣
↪many nodes)
    circles = c()
    #all_members will be flat version of circles
    all_members = c()
    for (line in circles_lines){

        #split the read line into array
        arr =  strsplit(line,"\t")[[1]]
        #First element is not a node, so remove it
        arr = arr[2:length(arr)]
        #print(length(arr))
        circles = c(circles,  list(arr) )
        all_members = c(all_members, arr)
    }
    all_members = unique(all_members)
    #N = number of unique els among all circles
    N = length(all_members)


    #ai = length(Ci)
    ais = c()
    for (circ in circles){
        ais = c(ais, length(circ))
    }
    #print("ais")
    #print(ais)
    #compute H(C)
    hc = 0
    for (ai in ais){
        #For all "if != 0" checks: if ai = 0, then log(0) gives error.
        #But we have factor of ai at front, so if 0 then we don't need to add
        if (ai != 0){
            hc = hc + ai/N * log(ai/N)
        }
    }
    hc = -1*hc

    #Get community
    #print("hc")
    #print(hc)
    net_path = paste(path_to_files, "/", net, ".edges", sep = "")
    graph = read_graph(net_path, format="ncol", directed=TRUE)
    graph = add.vertices(graph, nv = 1, name = net)
    edges = c()
    for (i in seq(1, vcount(graph)-1, 1)){
        edges = c(edges, c(vcount(graph),i))
```

```
}

graph = add_edges(graph, edges)
community  = walktrap.community(graph)
#print("got community structure")
#|K| and |C|
num_communities = max(community$membership)
num_circles = length(circles)
nodes = V(graph)$name
#bj = numver of nodes in community j (Kj) and has circle data
bjs = c()
hk = 0

#Compute H(K)
for (j in 1:num_communities){
    #print(community$membership)
    #print(community$membership == j)

    #for each node, check if 1. in commmunity and 2. has circle data
    comm_nodes = nodes[which(community$membership == j)]
    bj = 0
    for (node in comm_nodes){
        if (node %in% all_members){
            bj = bj + 1
        }
    }

    if (bj != 0){
        hk = hk + bj/N * log(bj/N)
    }
    bjs = c(bjs, bj)

}
hk = hk * -1


hc_given_k = 0
hk_given_c = 0
#Lastly loop over i and j, compute H(K|C) H(C|K)
for (j in 1:num_communities){
    #Kj = community j
    Kj = nodes[which(community$membership == j)]
    bj = bjs[j]

    #loop over circles
    for (i in 1:num_circles){
        #check how many in Ci and Kj
```

```
            Ci = circles[[i]]
            #print("circle i")
            #print(Ci)
            Aji = 0
            for (node in Kj){
                if (node %in% Ci){
                    Aji = Aji + 1
                }
            }
            if (Aji != 0){
                hc_given_k = hc_given_k + Aji/N*log(Aji/bj)
                hk_given_c = hk_given_c + Aji/N*log(Aji/ais[i])
            }

        }

    }
    hc_given_k = hc_given_k * -1
    hk_given_c = hk_given_c * -1

    print(paste("H(C)", hc, "H(K)", hk, "H(C|K)", hc_given_k, "H(K|C)",␣
↪hk_given_c))
    homogeneity = 1 - hc_given_k/hc
    completeness = 1 - hk_given_c/hk
    print(paste("Net: ", net, "h = ", homogeneity, "c = ", completeness))
    #####

}
```

```
[1] "data/gplus/gplus/109327480479767108490.circles"
[1] "circles lines"
[1] "H(C) 1.05077934757594 H(K) 1.00520818089008 H(C|K) 0.155636061763332 H(K|C)
0.673616224340775"
[1] "Net:  109327480479767108490 h =  0.851885115440867 c =  0.329873913536689"
[1] "data/gplus/gplus/115625564993990145546.circles"
[1] "circles lines"
[1] "H(C) 8.46514668159249 H(K) 1.08119096358335 H(C|K) 4.63982898243534 H(K|C)
4.78314811509253"
[1] "Net:  115625564993990145546 h =  0.451890303032235 c =  -3.4239623491117"
[1] "data/gplus/gplus/101373961279443806744.circles"
[1] "circles lines"
[1] "H(C) 0.384319958960942 H(K) 0.493330612772394 H(C|K) 0.382833906292573
H(K|C) 1.23541745845449"
[1] "Net:  101373961279443806744 h =  0.0038667069813052 c =  -1.5042383879479"
```

[ ]:

[ ]: