UCLA

ECE 232 - Large Scale Social and Complex Networks: Design and Algorithms
Spring 2021

Project 2: Social Network Mining

Authors:
Swapnil Sayan Saha (UID: 605353215)
Grant Young (UID: 505627579)

FACEBOOK NETWORK

Question 1.1:

In this question, we are asked to create the Facebook network from the edgelist file and report the number of nodes and number of edges of the Facebook network. To create the network, we first read the contents of `facebook combined.txt` file using `read.table()` function, and then create an undirected graph from the dataframe using `graph.data.frame()` function from igraph library in R. The resulting network is shown in Figure 1:

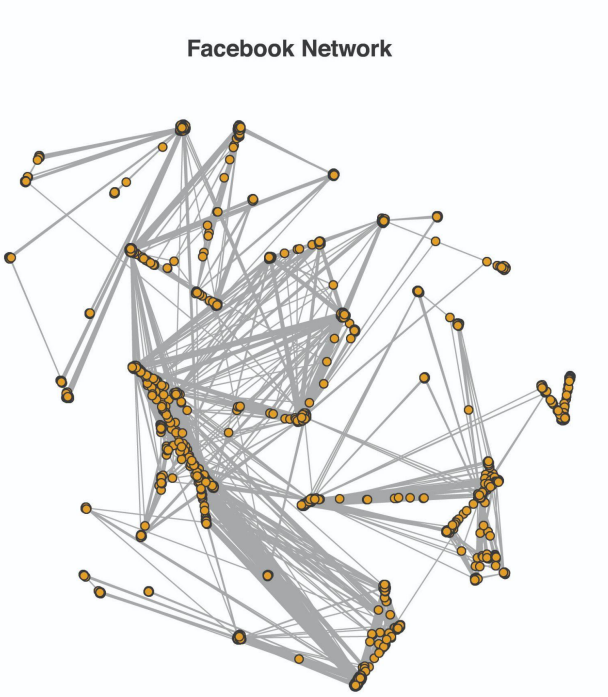


Figure 1: Generated Facebook network created from edgelist file.

We use `gorder()` and `gsize()` functions to calculate the number of nodes and edges of the network respectively. **The resulting network has 4039 nodes** (users) **and 88234 edges** (links), which is the exact number of nodes and edges reported in the Stanford SNAP Facebook Dataset website: (http://snap.stanford.edu/data/egonets-Facebook.html).

Question 1.2:

In this question, we are asked to find out if the Facebook network is connected or not. We use `is.connected()` function to find out the connectivity of the network. **We found out that the entire network is connected**. This means that there are no isolated users (nodes) without any friends in the network and all users have some connection (edges) with other users. Theoretically, social networks can be thought of as Barabasi-Albert networks with weak preferential attachment, referring to the fact that users with higher degrees or connectedness are more likely to receive edges with newer users. Since each new node is always attached to a connected node in the existing network, the final network is theoretically always connected by construction.

Question 2:

In this question, we are asked to find the diameter of the network. The diameter of a graph is defined as the largest distance between any pair of nodes (greatest shortest path between any two nodes). We use the `diameter()` function and find the **diameter of the network to be 8**.

Question 3:

In this question, we are asked to plot the degree distribution of the network and report the average degree of the network. For undirected graphs, the degree of a vertex v_i is defined as:

$$deg(v_i) = ||\{(v_i, v_j) \in E\}||$$

where, E is the set of edges of the graph. The degree distribution of the graph, which is the probability distribution function of the random variable X, denoting the degree of a randomly selected vertex of the graph, is given by:

$$\mathcal{P}(X = k) = \frac{\text{number of vertices with degree } k}{\text{total number of vertices}}$$

The average or expected degree of the network $\mathbb{E}(deg)$ is given by:

$$\left(\mathbb{E}_m(deg) = \frac{1}{m}\sum_{k=1}^{k_{\max}} k \times \mathcal{N}_k\right) = \left(\mathbb{E}(deg)\sum_{k=1}^{k_{\max}} k \times P_k\right) \forall (m \to \infty)$$

where m = number of nodes picked, $\mathcal{N}_k$ = number of vertices with degree k, P_k = probability that a randomly picked node has degree k.

Figure 2 shows the plot of the degree distribution of the network. **The average degree is 43.691013**.

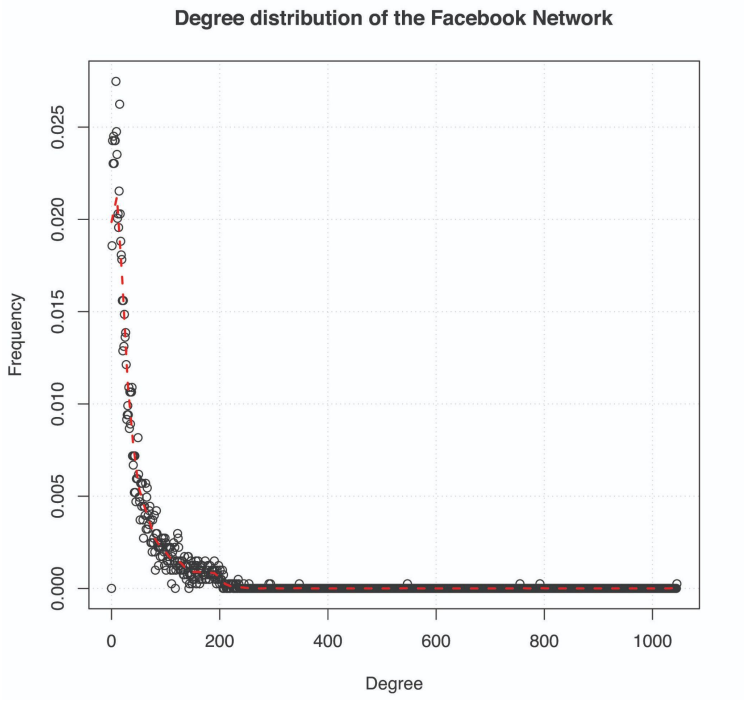


Figure 2: Degree distribution of the Facebook network.

From Figure 2, we observe that the degree distribution follows the power-law model, exhibiting the following fat-tailed degree distribution:

$$P_k \propto \frac{1}{k^\gamma} = \frac{1}{k^\gamma \sum_{k=1}^{k_{\max}} k^{-\gamma}}, \quad k \in 1, 2, 3, ..., k_{\max}, \quad \gamma > 0$$

$-\gamma$ is the gradient of the $\log P_k$ versus $\log k$ curve, which is linear. This is expected for scale-free power-law networks with weak linear preferential attachment.

Question 4:

In this question, we are asked to plot the degree distribution obtained in Question 3 in log-log scale and estimate the slope of the best fit linear regression line. Figure 3 shows the plot. **The gradient of the line is -1.2475**.

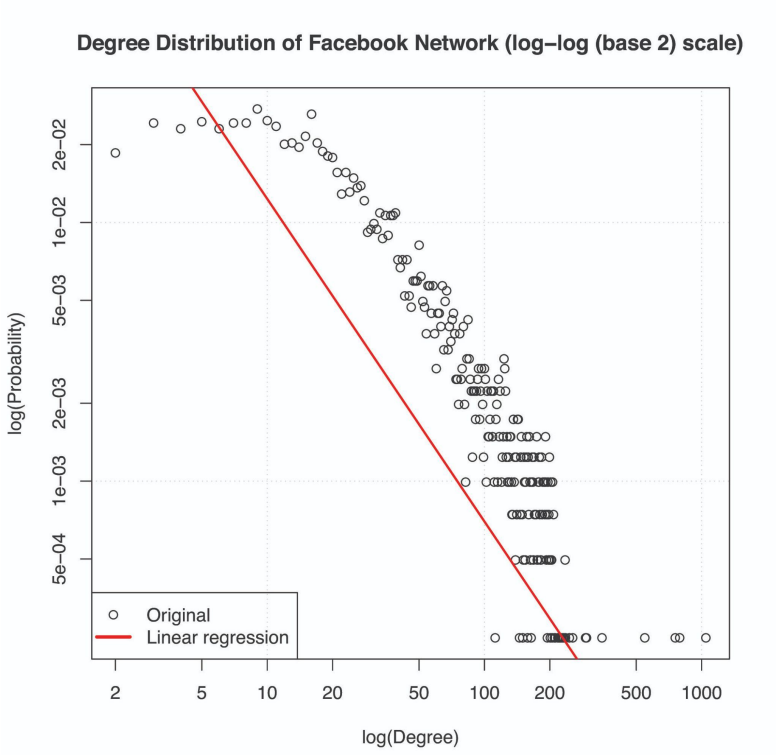


Figure 3: Degree distribution of the Facebook network in log-log scale.

From Figure 3, we see that the degree-distribution in the log-log scale is approximately linear. As discussed in Question 3, the degree distribution of scale-free networks is linear in the log-log scale:

$$P_k \propto \frac{1}{k^\gamma} = \frac{1}{k^\gamma \sum_{k=1}^{k_{\max}} k^{-\gamma}}, \quad k \in 1, 2, 3, ..., k_{\max}, \quad \gamma > 0$$

Since $1 < \gamma \leq 2$, both the expected degree and variance are unbounded, leading to a dense network with a significant number of high degree nodes. However, since $\gamma << 3$, the nodes in the Facebook network are not strongly preferentially attached, but follow weak preferential attachment. In preferential attachment models ($\gamma \approx 3$), nodes with higher degrees or connectedness are more likely to receive edges with newer nodes without any heuristics on whether an incoming node is related to a high-degree node or not. This results in a sparse network with a large number of high-degree nodes. However, for social networks, users tend to form communities with known (mutual) friends, leading to densely packed communities of known users with heterogeneous connectedness.

Question 5:

In this question, we are asked to create a personalized network of the user whose ID is 1, and count the number of nodes and edges of the network. A personalized network is defined as a subgraph induced by user v_i and its neighbours. In other words, the network concerns the neighbours of a target (core) node. The generated network is shown in Figure 4. **This network has 348 nodes and 2866 edges**.

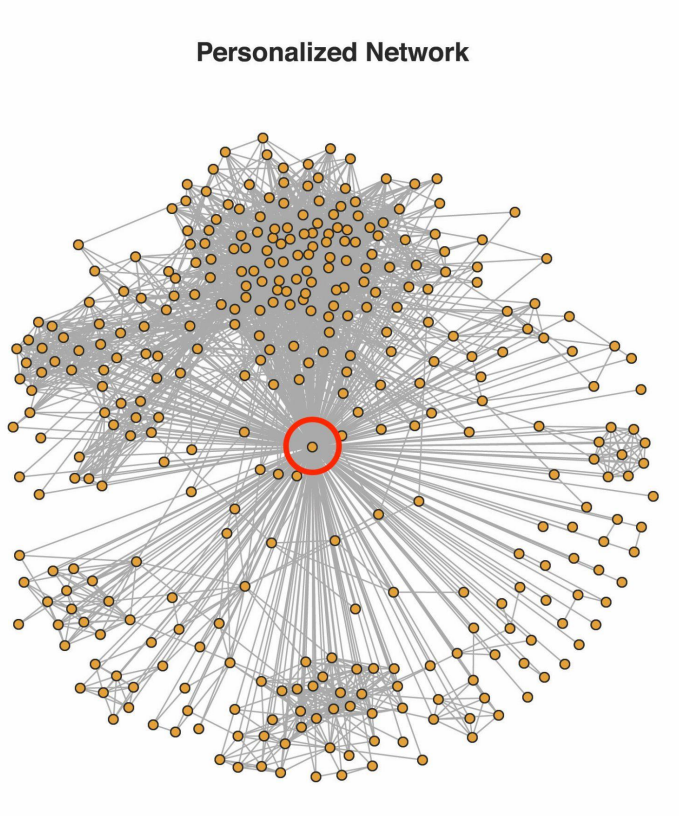


Figure 4: Personalized network for user ID 1, with the target user (called core node) circled in red.

Question 6:

The diameter of the personalized network for user ID 1 is 2. The trivial lower bound of the personalized network diameter is 1 and the trivial upper bound for the personalized network is 2.

Question 7:

The diameter of a graph is defined as the largest distance between any pair of nodes (greatest shortest path between any two nodes). If the diameter of a personalized network is 2, it means that there exists users who are not connected directly in the subgraph, but are connected through a mutual user. In other words, the network is not fully connected and there exists vertices v_i and v_j in the network such that that v_i and v_j are connected via a mutual tertiary node v_k. Not all users in the network know each other directly.

On the other hand, if the diameter of a personalized network is 1, then all users in the network know each other directly and there exists direct edges between all vertices in the network. The network in this case is fully connected.

Question 8:

In this question, we are asked to find the total number of core nodes and their average degree in the Facebook network. A core node is defined as the nodes that have more than 200 neighbors. **The Facebook network has 40 core nodes, with an average degree of 279.375**.

Question 9:

In this question, we are asked to find the community structures of the personalized networks of 5 core nodes (node IDs 1, 108, 349, 484 and 1087) using Fast-Greedy, Edge-Betweenness, and Infomap community detection algorithms and also report the resulting modularity scores.

Community structure of a graph is defined as a clustering of the vertices in the network such that the number of inter cluster edges is much smaller than the number of intra cluster edges. In other words, community structures segregate regions of high connectedness from the overall sparse network structure, with a simple example shown in Figure 5.

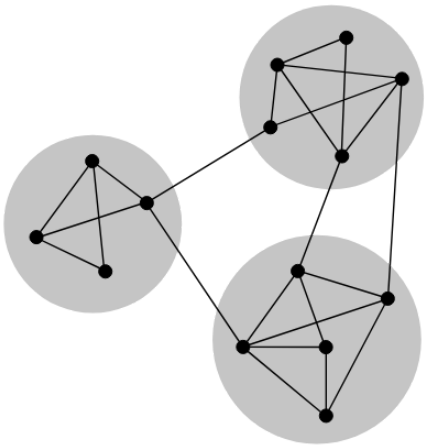


Figure 5: Community structure showing 3 clusters of high connectedness in a random graph.

To find community structures, we use Mark Newman modularity index. For a particular cluster C_i, the modularity index m_{C_i} is given as:

$$m_{C_i} = f_{C_i} - r_{C_i}$$

where, f_{C_i} is the number of edges in C_i and r_{C_i} is the expected number of edges in C_i if the network was created randomly with the same degree distribution. Expanding the equation:

$$m_{C_i} = \sum_{i,j \in C_i} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$$

where, A refers to the node-node incidence matrix, k refers to the degree of nodes selected for random stub (dangling edges) matching and m is the number of edges in r_{C_i}. For a network P partitioned into k disjoint clusters, the modularity index of the network is given as:

$$Q(P) = \sum \frac{1}{2m} \sum_{i,j \in C_i} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$$

The node-node incidence matrix is a mathematical representation of a graph with the rows and columns corresponding to the vertices and edges of a graph respectively. To construct A:

- If an edge j is leaving from node i, A_{ij} is +1.
- If an edge j is entering towards node i, A_{ij} is -1 (for undirected graphs, -1 and +1 will just be 1).
- If an edge j is neither entering towards node i nor leaving, A_{ij} is 0.

The maximum value of $Q(P)$ is 1. The goal of the clustering algorithm is to find the network partitioning with the highest modularity index. Intuitively, modularity measures the strength of the network division into modules with strong interconnections (community structures). The higher the modularity, the larger is the number of edges within communities, while the communities are sparsely connected to each other.

The greedy algorithm for finding out the modularity index is as follows:

- Start with n clusters.
- Compute the change in $Q(P)$ if a pair of clusters among all the existing clusters is merged and perform the merge if $Q(P)$ increases after merging.
- Repeat step 2 until no significant changes in $Q(P)$ are observed.

The edge-betweenness of an edge e, denoted as $w(e)$, in terms of the shortest path between each pair of nodes in the network is denoted as:

$$w(e) = \text{Number of shortest paths } e \text{ is a part of}$$

Edges which act as bridges between areas of high connectedness have high values of $w(e)$ because the shortest paths between nodes in different clusters are connected to these edges. The edge-betweenness community detection algorithm (also known as Girvan–Newman algorithm) is given as follows:

- Compute $w(e)$ for all edges and delete the edge with the highest $w(e)$.
- Repeat step 1 for the modified network and compute the $Q(P)$. If $Q(P)$ increases, repeat step 1, else terminate the algorithm.

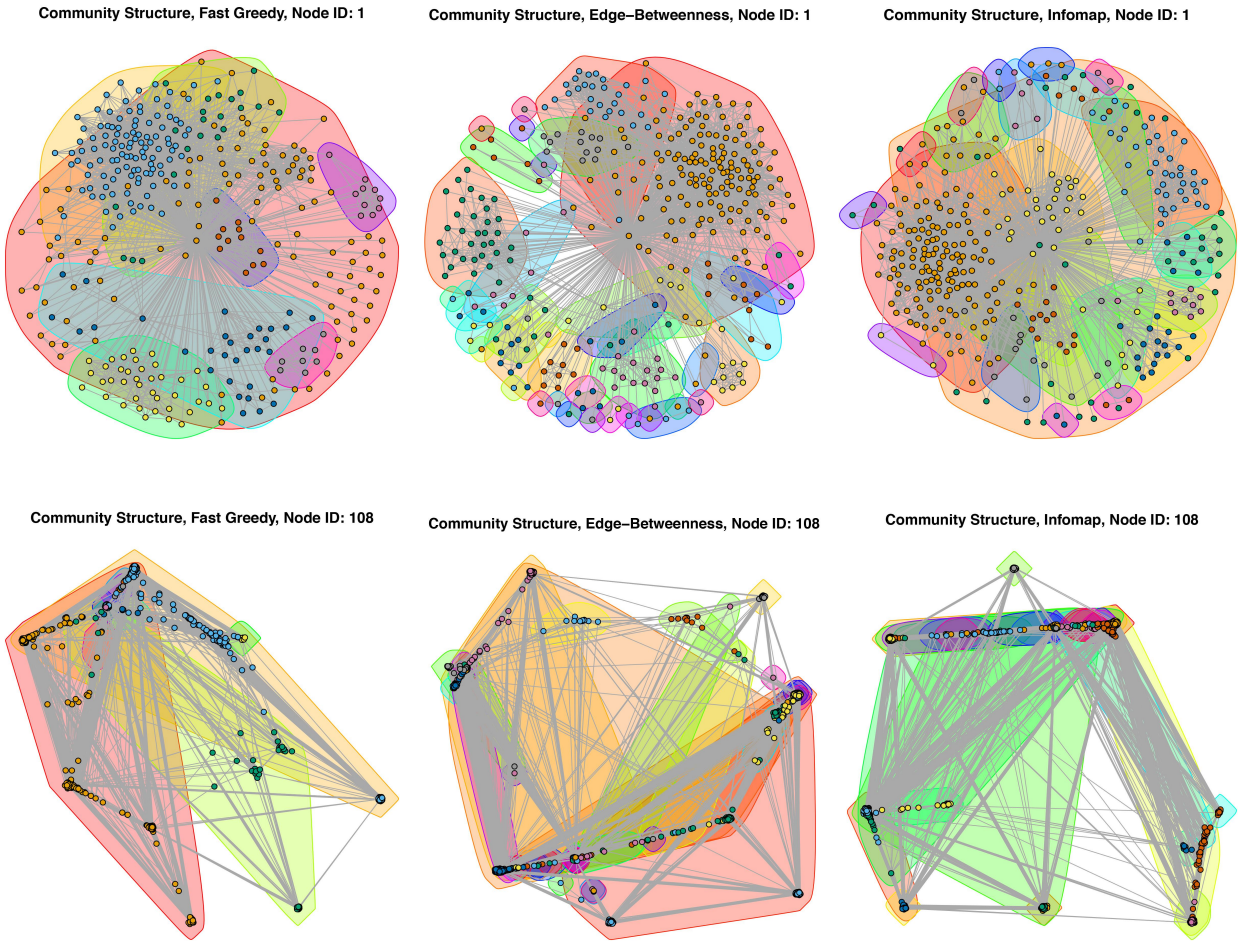
Infomap finds the community structure that minimizes the expected description length of a random walker trajectory. Informap uses probability flow on random walks (information theoretic approach) as a surrogate for information flow in the network, segregating the network into modules based on a compressed description of probability flow.

Table 1 shows the modularity scores of the 5 personalized networks obtained via the three community detection algorithms.

Table 1: Modularity scores of personalized networks for 5 core nodes using 3 community detection algorithms.

Core Node	Node Count	Edge Count	Fast-Greedy	Edge-Betweenness	Infomap
1	348	2866	0.413101	0.353302	0.394125
108	1046	27795	0.435929	0.506755	0.508223
349	230	3441	0.251715	0.133528	0.095464
484	232	4525	0.507002	0.489095	0.515279
1087	206	7409	0.145531	0.027624	0.026907

Figure 6 shows the community structures for each of the core nodes for the three algorithms.



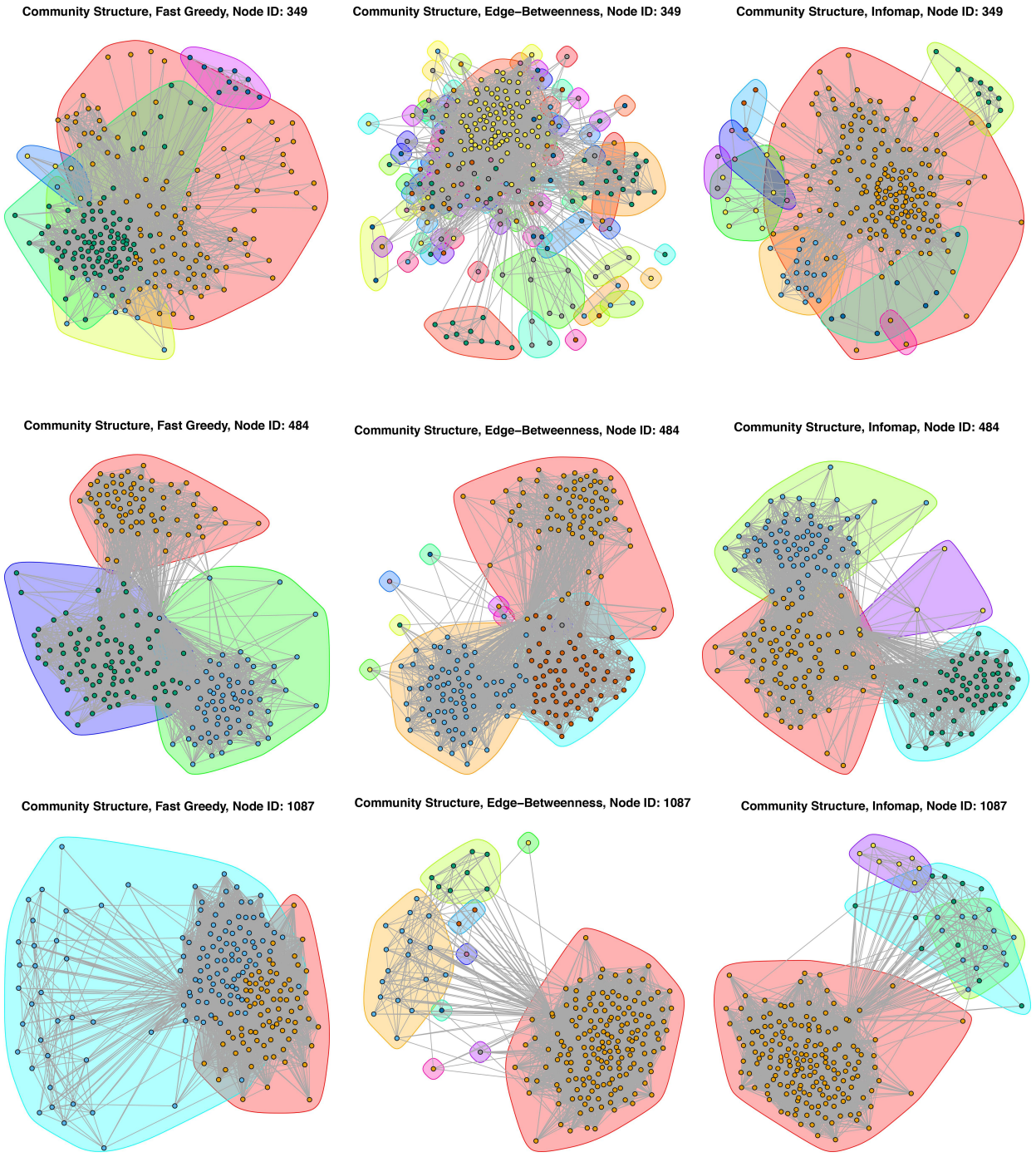


Figure 6: Community structures of personalized networks for 5 core nodes using 3 algorithms.

We can make several observations from Figure 6 and Table 1:

- The personalized network for node 484 achieves the highest average modularity score. Judging from Figure 6, we observe that the density of clusters are roughly homogenous and form distinguishable regions of high connectedness with sparse connectivity among the high-density regions.

- The personalized network for node 1087 has the lowest modularity for all three algorithms among the five personalized networks. This is because the network for node 1087 has the lowest node to edge ratio. Intuitively, a higher value of m indicates that an incoming node is connected to a larger number of older nodes. While this should result in strong intra-community connectedness, the global sparsity among different communities is lost due to the connectedness requirement brought on by high values of m, resulting in edges being formed among otherwise distinct clusters and hence weakening the community structures. Mathematically,

  $$Q(P) = \sum \frac{1}{2m} \sum_{i,j \in C_i} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$$

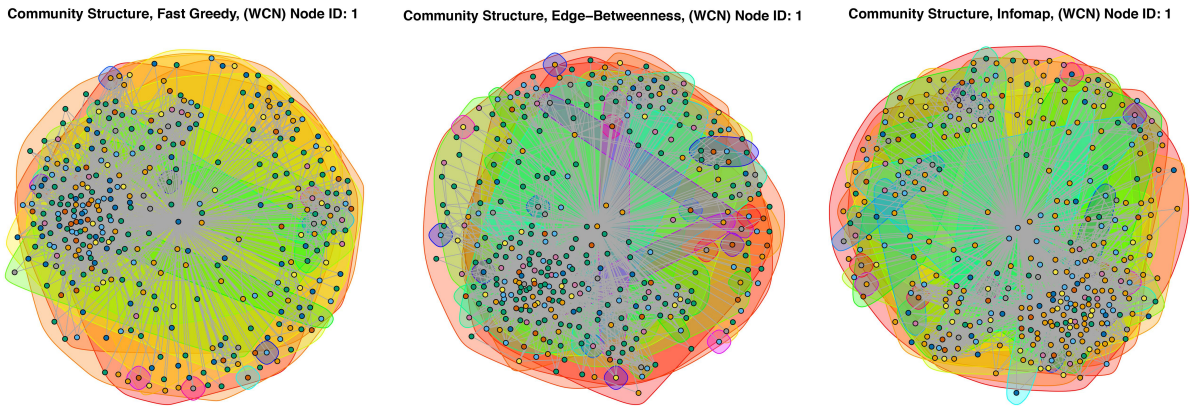
  If the number of edges in the network m (not to be confused with the actual m we are talking about, which is the number of old nodes the incoming node connects with) increases, then $Q(P)$ dtops. As a result, less clusters are formed in the overall graph.
- Edge-betweenness algorithm performs the worst on average among all three algorithms based on modularity score. This is partially observable from Figure 6, where we see that the edge-betweenness community detection algorithm tends to leave out a significant number of nodes within isolated communities of single members, leading to weak intra-community connectedness. This is because edge-betweenness generates communities via hierarchical decomposition by building a full dendrogram, assuming that there is only a single path to go from one group to another group and cutting the tree into groups via the modularity score of the partitions at each level of the tree. The process leaves out nodes with few vertices in their own communities, as the probability of them being included in a cut (community) at the initial stages of the algorithm is low. In addition, we observed that edge-betweenness is the slowest among the three methods. This is because of the computational complexity of calculations involved, with the scores being recalculated after every edge removal.
- Fast-greedy algorithm performs the best on average among all three algorithms based on modularity score, followed by Infomap. Compared to the edge-betweenness algorithm, fast-greedy is bottom-up instead of top-down, optimizing the modularity in a greedy manner such that the merges are locally optimal, yielding the largest possible increase in the modularity with each merge. In other words, the greedy algorithm is optimized to increase modularity, whereas edge-betweenness aims at decomposing the network based on the number of shortest paths, which does not always yield dense communities with sparse inter-community connections. The algorithm is also faster than edge-betweenness without requiring any parameter turning.
- Fast-greedy algorithm does not perform well on large networks, such as for core node 108. Furthermore, compared to Infomap or edge-betweenness, the fast-greedy algorithm often tends to merge nodes from otherwise visibly separate communities. This is because of the resolution limit problem of the fast-greedy method, with the tendency of merging nodes within a community below a given number of vertices to neighbouring nodes. The problem is amplified for large networks, where the greedy algorithm tends to form fewer communities compared to Infomap or edge-betweenness.
- Infomap and edge-betweenness algorithms perform similar to each other.

Question 10:

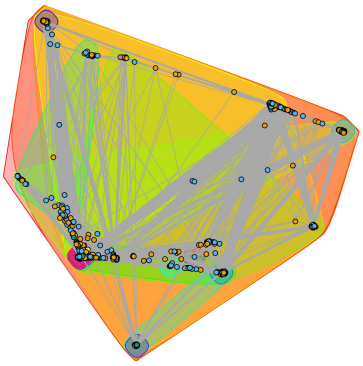
In this question, we are asked to find the community structures of the personalized networks of 5 core nodes (node IDs 1, 108, 349, 484 and 1087) using Fast-Greedy, Edge-Betweenness, and Infomap community detection algorithms but with the core node removed, as well as compare the resulting modularity scores with those obtained in Question 9. Table 2 shows the comparison of modularity scores of the 5 personalized networks obtained via the three community detection algorithms with and without the core node involved. Figure 7 shows the community structures for each of the core nodes for the three algorithms without the core nodes.

Table 2: Comparison of modularity scores with and without core node involved.

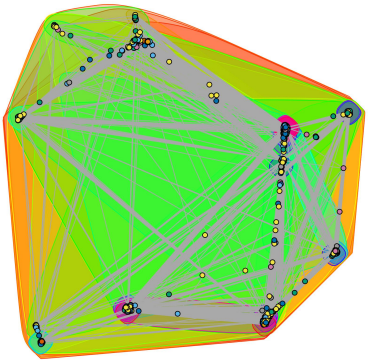
Core Node	Core Node Present?	Fast-Greedy	Edge-Betweenness	Infomap
1	Yes	0.413101	0.353302	0.394125
	No	0.441853	0.416146	0.418008
108	Yes	0.435929	0.506755	0.508223
	No	0.458127	0.521322	0.520989
349	Yes	0.251715	0.133528	0.095464
	No	0.245692	0.150566	0.244816
484	Yes	0.507002	0.489095	0.515279
	No	0.534214	0.515441	0.543444
1087	Yes	0.145531	0.027624	0.026907
	No	0.148196	0.032495	0.027372



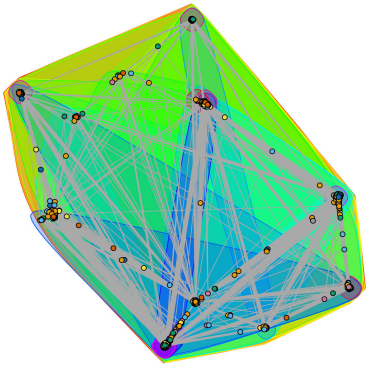
Community Structure, Fast Greedy, (WCN) Node ID: 108



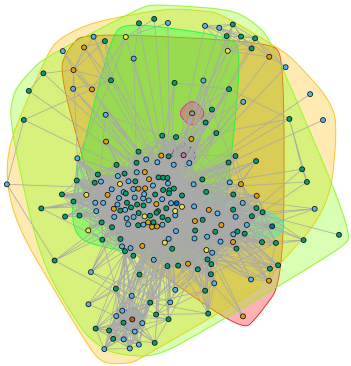
Community Structure, Edge–Betweenness, (WCN) Node ID: 108



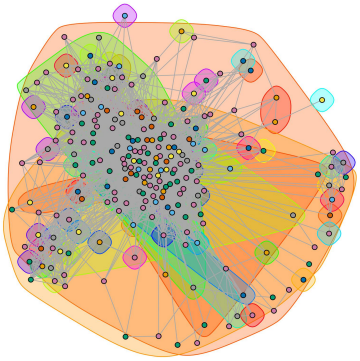
Community Structure, Infomap, (WCN) Node ID: 108



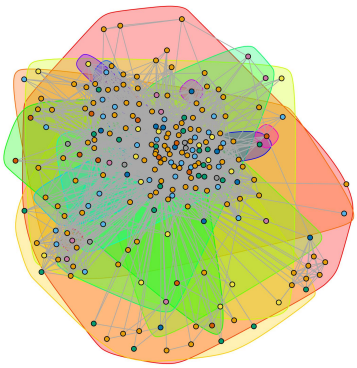
Community Structure, Fast Greedy, (WCN) Node ID: 349



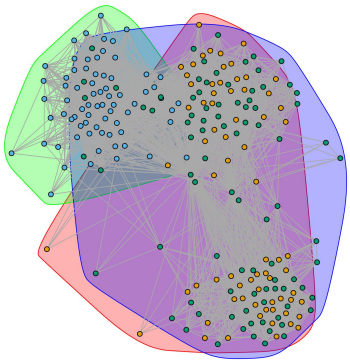
Community Structure, Edge–Betweenness, (WCN) Node ID: 349



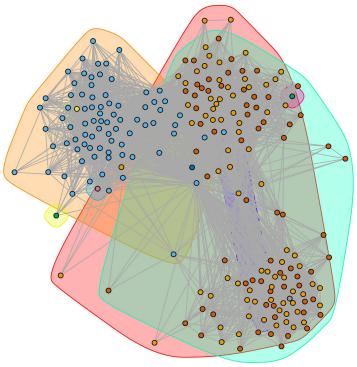
Community Structure, Infomap, (WCN) Node ID: 349



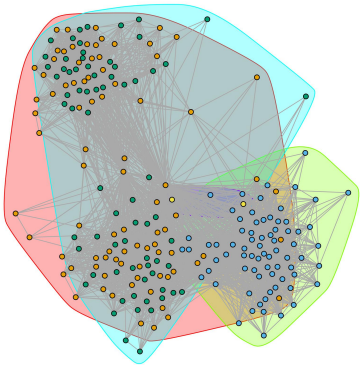
Community Structure, Fast Greedy, (WCN) Node ID: 484



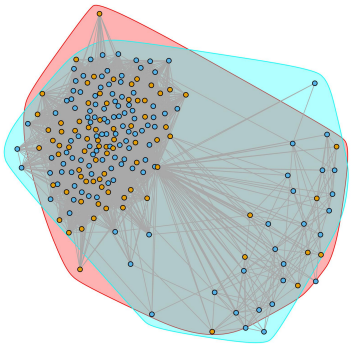
Community Structure, Edge–Betweenness, (WCN) Node ID: 484



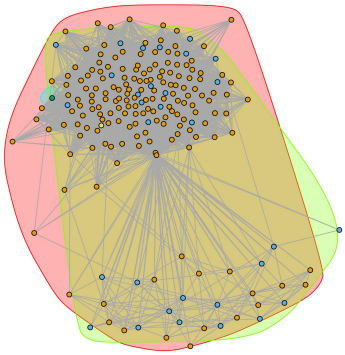
Community Structure, Infomap, (WCN) Node ID: 484



Community Structure, Fast Greedy, (WCN) Node ID: 1087



Community Structure, Edge–Betweenness, (WCN) Node ID: 1087



Community Structure, Infomap, (WCN) Node ID: 1087

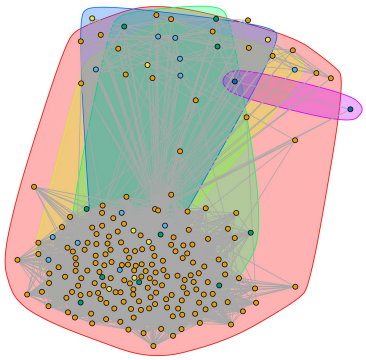


Figure 7: Community structures of personalized networks for 5 core nodes using 3 algorithms, with core node removed.

From Table 2, we see that the modularity scores have increased after removing the core node for all 5 personalized networks. This is because the core node acts as the bridge between all of the other nodes, which causes the network with the core node less capable of being partitioned into communities with strong intra-community connectedness and sparse inter-community connectedness compared to the network without the core node. With the presence of the core node, it is difficult to classify the core node into a single community as it is connected to every other node while also making it difficult to assign densely packed and sparsely inter-connected communities among the other nodes, resulting in a dense network with low modularity. In addition, with the core node present, the other nodes are not always directly connected to each other or the rest of the graph, resulting in many isolated communities (communities with a single node) for edge-betweenness as observed in Figure 6. With the core node removed, the edges from the core node to all other nodes are removed, allowing the community partition algorithms to find densely packed areas of high connectedness with sparse inter-community edges. The probability of strong intra-community connections is greater for networks with core nodes removed, and likewise, the extent of sparsity among communities are amplified in such networks. These facts are also evident from Figure 7, where we see that all the algorithms, especially edge-betweenness, have a decreased tendency of forming groups with isolated nodes. In addition, the community structures are still well-defined and well-partitioned, despite absence of the core nodes.

Question 11:

The embeddedness of a node v_i is defined as the number of mutual vertices a given node shares with the core node v_c. Mathematically, the embeddedness between v_c and v_i in the personalized network P is given by:

$$\text{Embeddedness}^{P}_{v_i,v_c} = deg(v_i)^P - 1$$

Note that $deg(v_i)^P \neq deg(v_i)$. $deg(v_i)^P$ is the degree of v_i in the personalized network and not in the original network. To get the embeddedness directly from the original network:

$$\text{Embeddedness}_{v_i,v_c} = deg(v_i) - N_{v_i} - 1 = |deg(v_c) \cap deg(v_i)| - 1$$

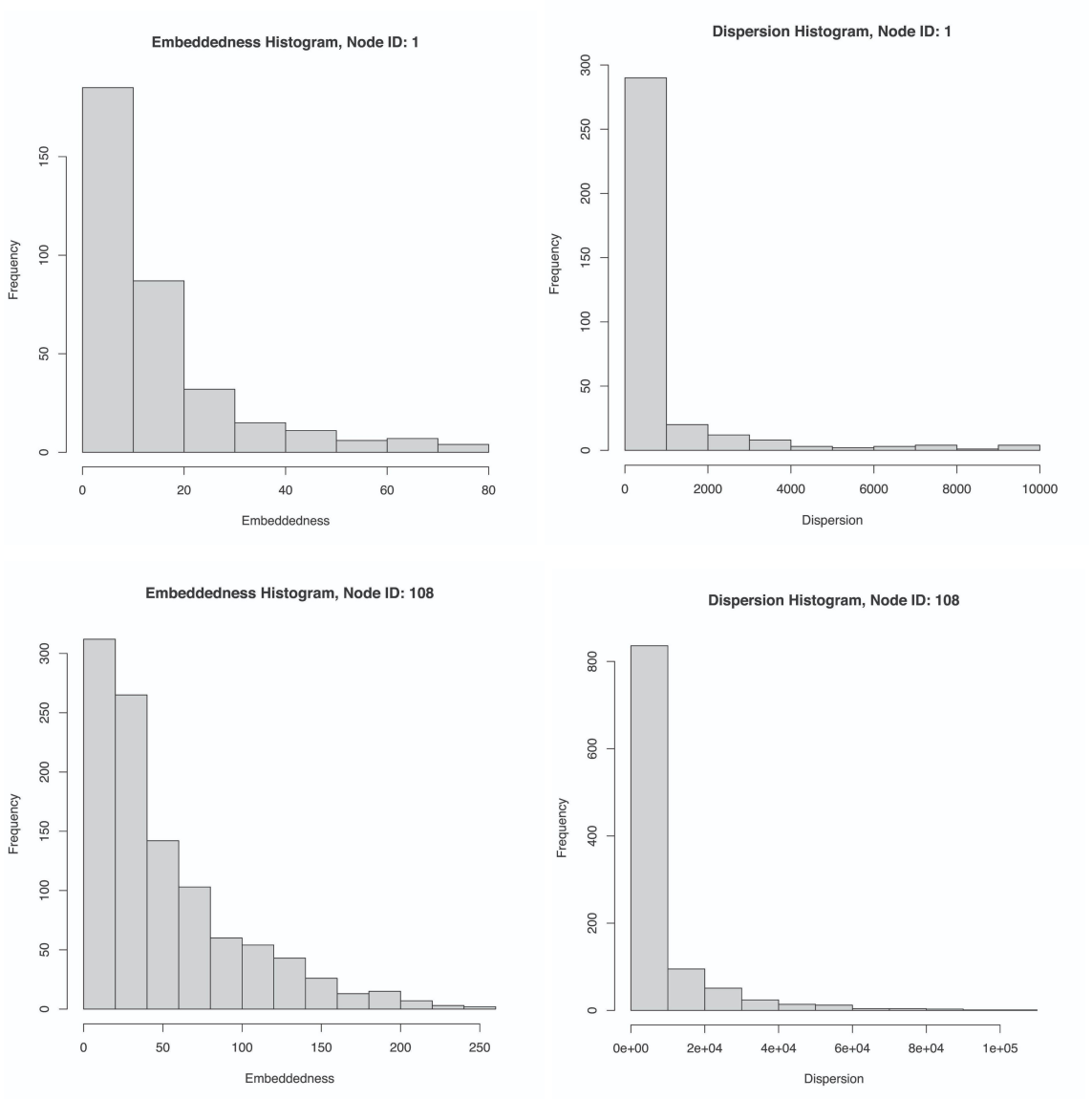
Where N_{v_i} is the number of neighbors of v_i absent in the personalized network. Thus, embeddedness is defined as the overlap in the social circle of two users.

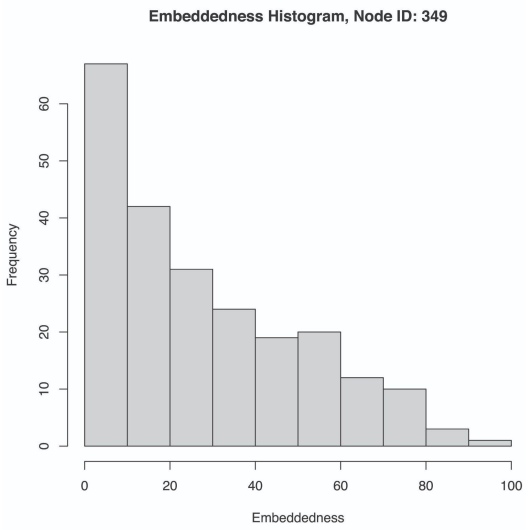
Question 12:

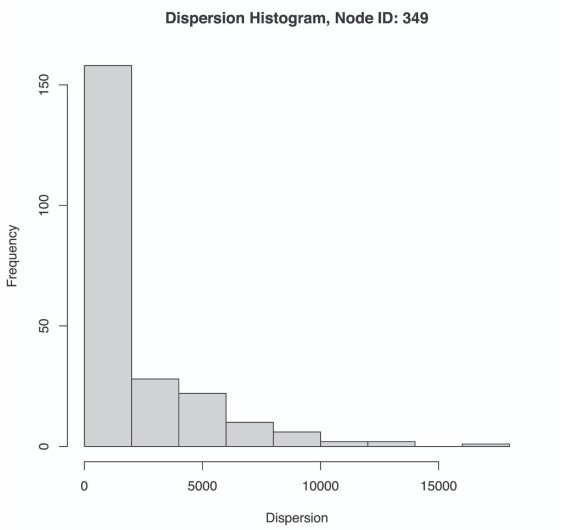
In this question, we are asked to plot the distribution histogram of embeddedness and dispersion for each of the core node's (node IDs 1, 108, 349, 484 and 1087) personalized networks. We already defined embeddedness in Question 11. Dispersion of a node is defined as the sum of distances between every pair of the mutual vertices the node shares with the core node, calculated in a modified subgraph graph with the target node and core node removed.

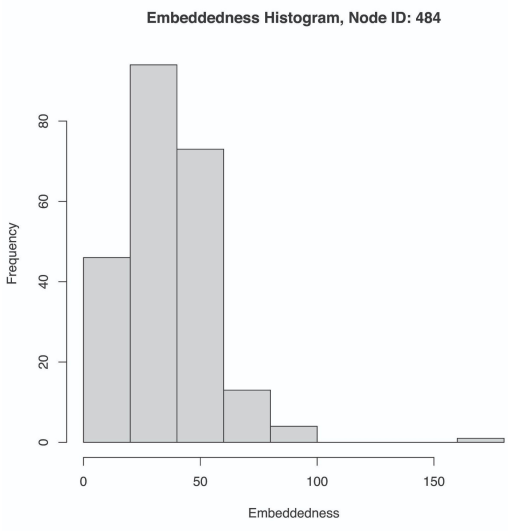
$$disp(u, v) = \sum_{s,t \in C_{uv}} d_v(s, t)$$

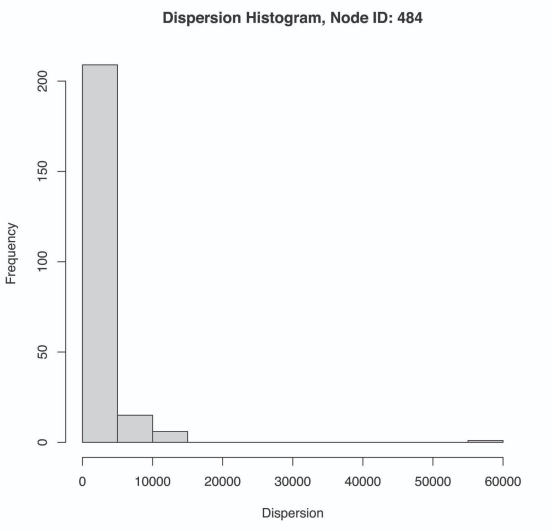
where, $C_{u,v} = u \cap v$, u and v denote the core and target nodes, d_v is the distance function. The embeddedness and dispersion plots are shown in Figure 8.

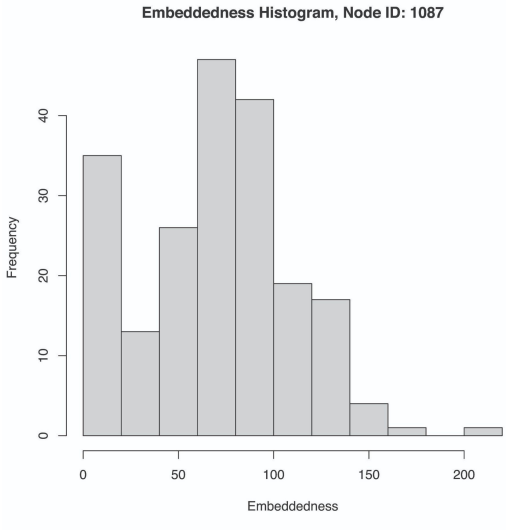



Embeddedness Histogram, Node ID: 349
Frequency
0
10
20
30
40
50
60
0
20
40
60
80
100
Embeddedness


Dispersion Histogram, Node ID: 349
Frequency
0
50
100
150
0
5000
10000
15000
Dispersion


Embeddedness Histogram, Node ID: 484
Frequency
0
20
40
60
80
0
50
100
150
Embeddedness


Dispersion Histogram, Node ID: 484
Frequency
0
50
100
150
200
0
10000
20000
30000
40000
50000
60000
Dispersion


Embeddedness Histogram, Node ID: 1087
Frequency
0
10
20
30
40
0
50
100
150
200
Embeddedness

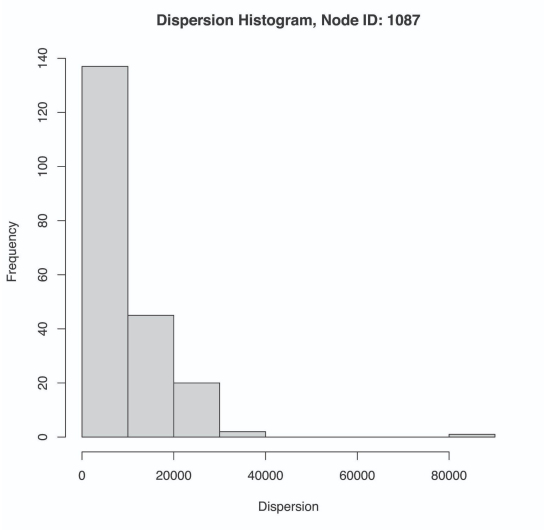

Dispersion Histogram, Node ID: 1087
Frequency
0
20
40
60
80
100
120
140
0
20000
40000
60000
80000
Dispersion

Figure 8: Embeddedness and dispersion histograms for each of the core node's personalized networks.

We can make several observations from Figure 8:

- The range (as well as expected value) of dispersions depends on the number of edges (degree) of the core node in the personalized network, the higher the number of edges, the higher is the dispersion. In addition, the expected value of embeddedness is dependent on the number of edges in the personalized network. This makes sense because:

$$disp(u, v) = \sum_{s,t \in C_{uv}} d_v(s, t)$$

$$\text{Embeddedness}^P_{v_i,v_c} = deg(v_i)^P - 1$$

  The higher the degree of the core node, the greater is $|C_{uv}|$, leading to a larger value of $disp(u, v)$. On the other hand, embeddedness does not have any explicit relation with either the number of mutual nodes or edges in the personalized network.
- Both embeddedness and dispersion roughly follow the power-law distribution due to their inherent relation with the degree distribution of the personalized networks, which follow power-law distribution stemming to weak preferential attachment:

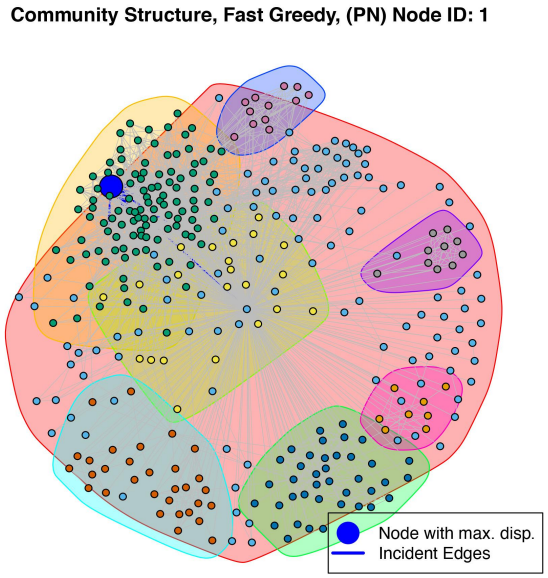
$$\text{Embeddedness}^P_{v_i,v_c} = deg(v_i)^P - 1$$

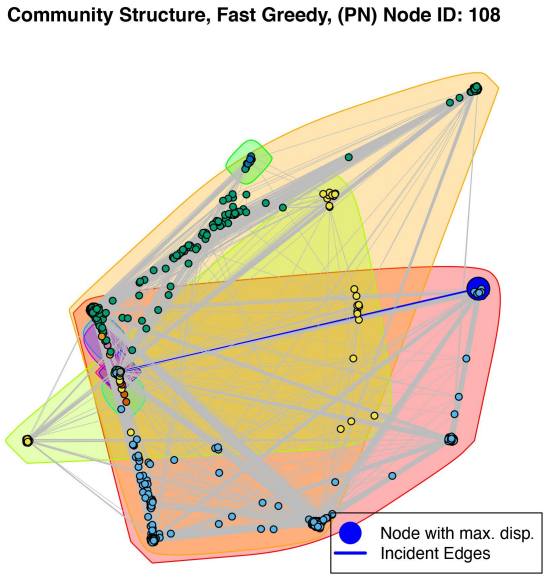
$$disp(u, v) = \sum_{s,t \in C_{uv}} d_v(s, t), \quad \begin{cases} |C_{u,v}| \propto deg(u) \wedge deg(v) \\ d_v(s, t) \propto deg(s) \wedge deg(t) \end{cases}$$

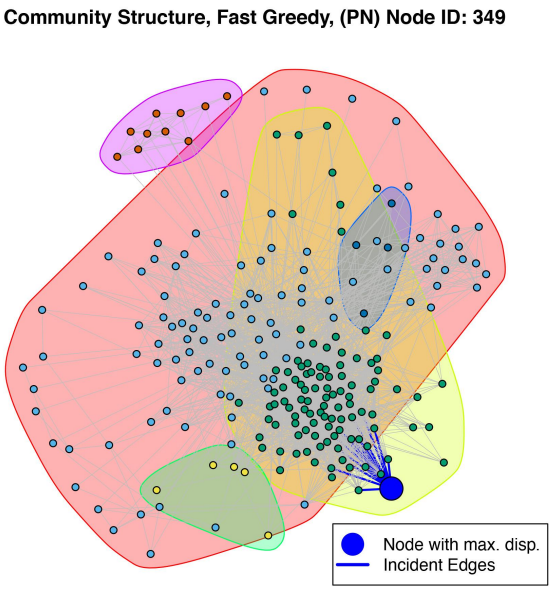
- Dispersion measures the extent to which two people's mutual friends are not themselves well-connected, in other words, the mutual nodes of two strongly connected nodes are not well connected and must display a dispersed structure. Hypothetically, networks with a low modularity should exhibit a dispersion distribution with lower skewness compared to networks with high modularity, as the probability of forming regions of high connectedness with sparse connectivity among clusters are lower for networks with ambiguous community structures. This is indeed observed for the dispersion distributions for the networks with core nodes with 349 and 1087, both of which have the lowest modularity scores.
- Embeddedness depends on the number of nodes in the personalized network. The higher the number of nodes and connectivity among them in the social circle, the greater is the embeddedness, even though embeddedness is not a good indicator of strong ties, as strongly tied nodes may have low degrees of embeddedness.

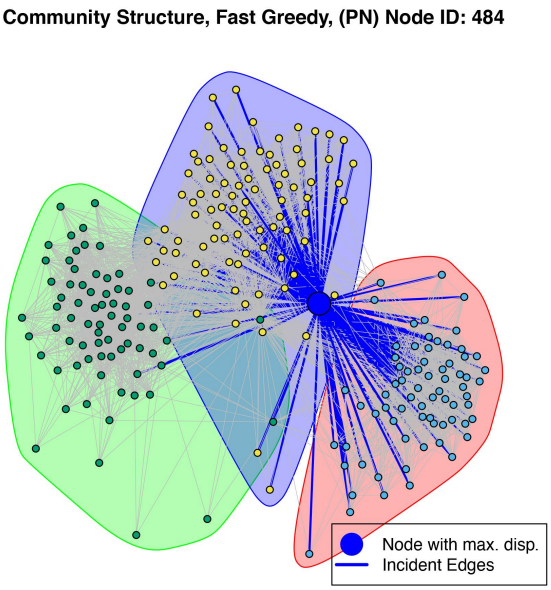
Question 13:

In this question, we are asked to plot the community structure (using Fast-Greedy) of the personalized network for each of the core nodes and highlight the node with maximum dispersion along with its incident edges. Figure 9 shows the plots. Nodes with maximum dispersion, along with its incident edges, are both highlighted in blue.









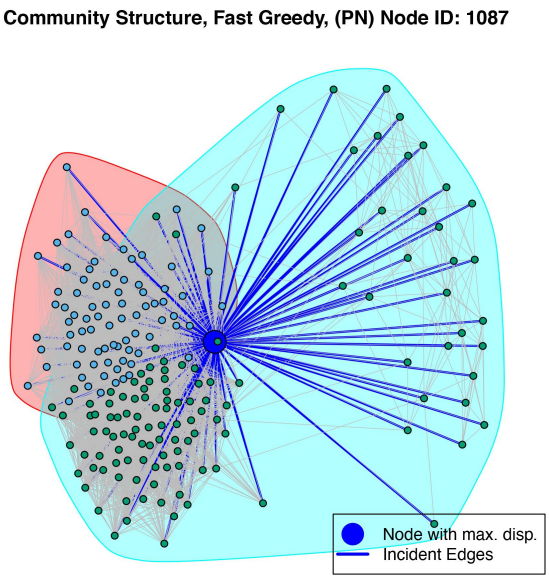
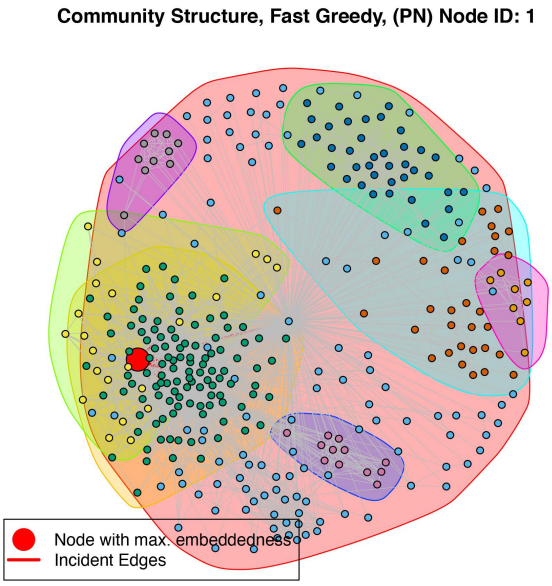
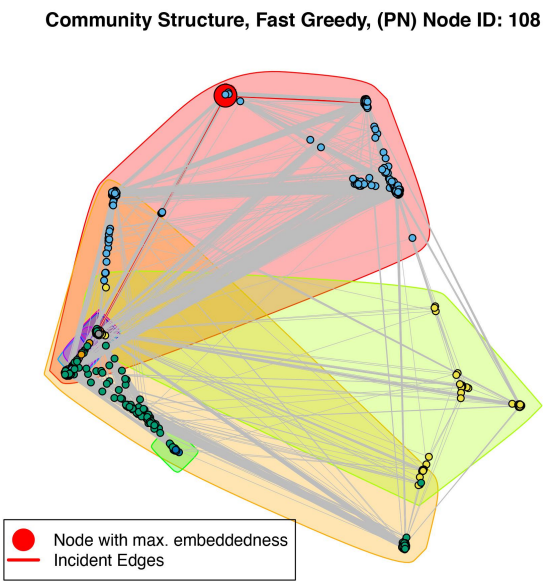


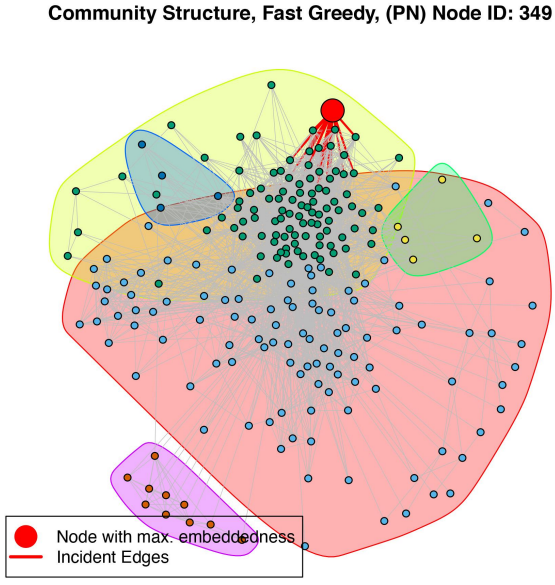
Figure 9: Community structure of personalized networks for core nodes, with maximum dispersion node and incident edges to that node highlighted.

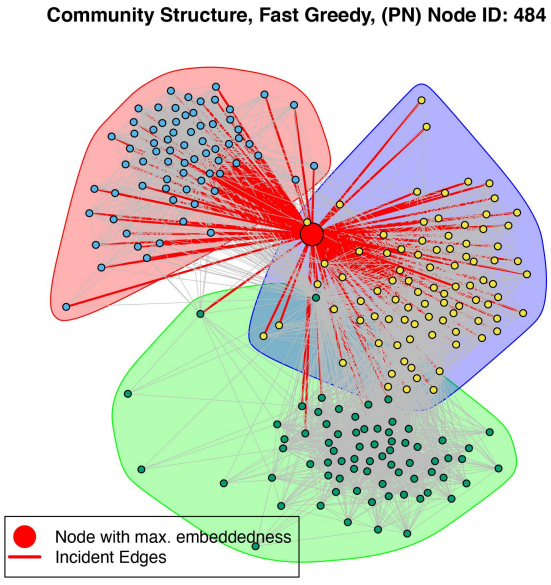
Question 14:

In this question, we are asked to plot the community structure (using Fast-Greedy) of the personalized network for each of the core nodes and highlight the node with maximum embeddedness along with its incident edges, as well as highlight the node with maximum dispersion/embeddedness (excluding the nodes having zero embeddedness if there are any) along with its incident edges. Figure 10 shows the plots with nodes of maximum embeddedness highlighted in red, while Figure 11 shows the plots with nodes of maximum dispersion/embeddedness highlighted in green.









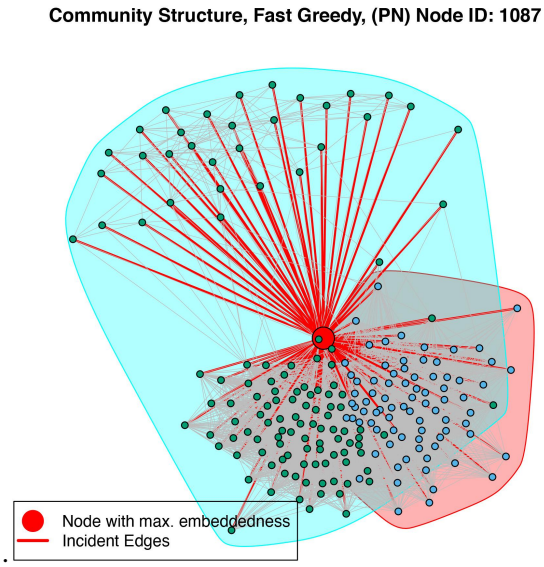
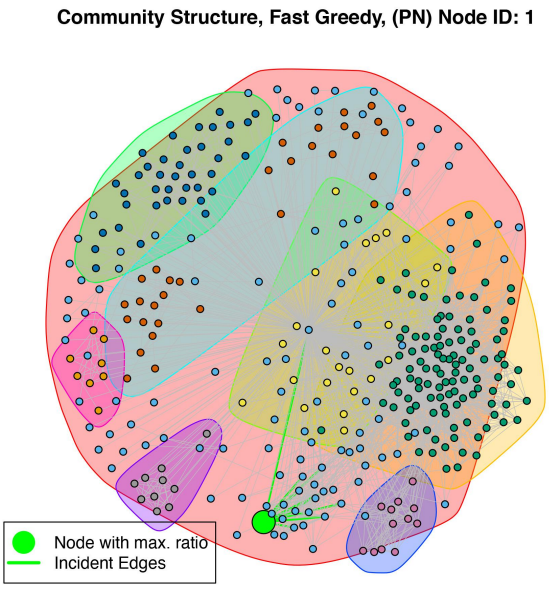
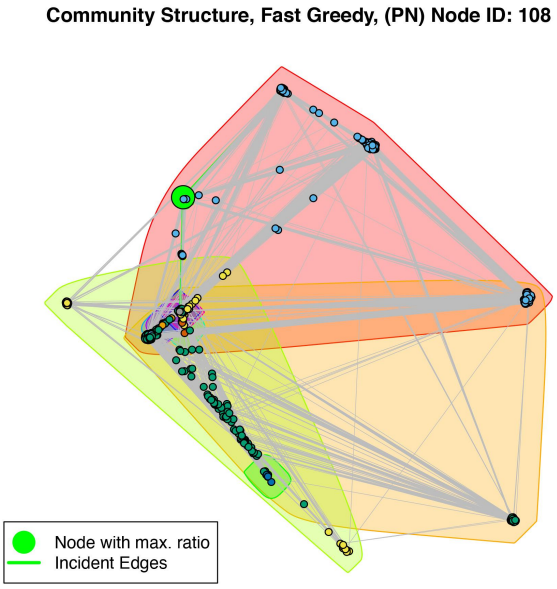


Figure 10: Community structure of personalized networks for core nodes, with maximum embeddedness node and incident edges to that node highlighted.





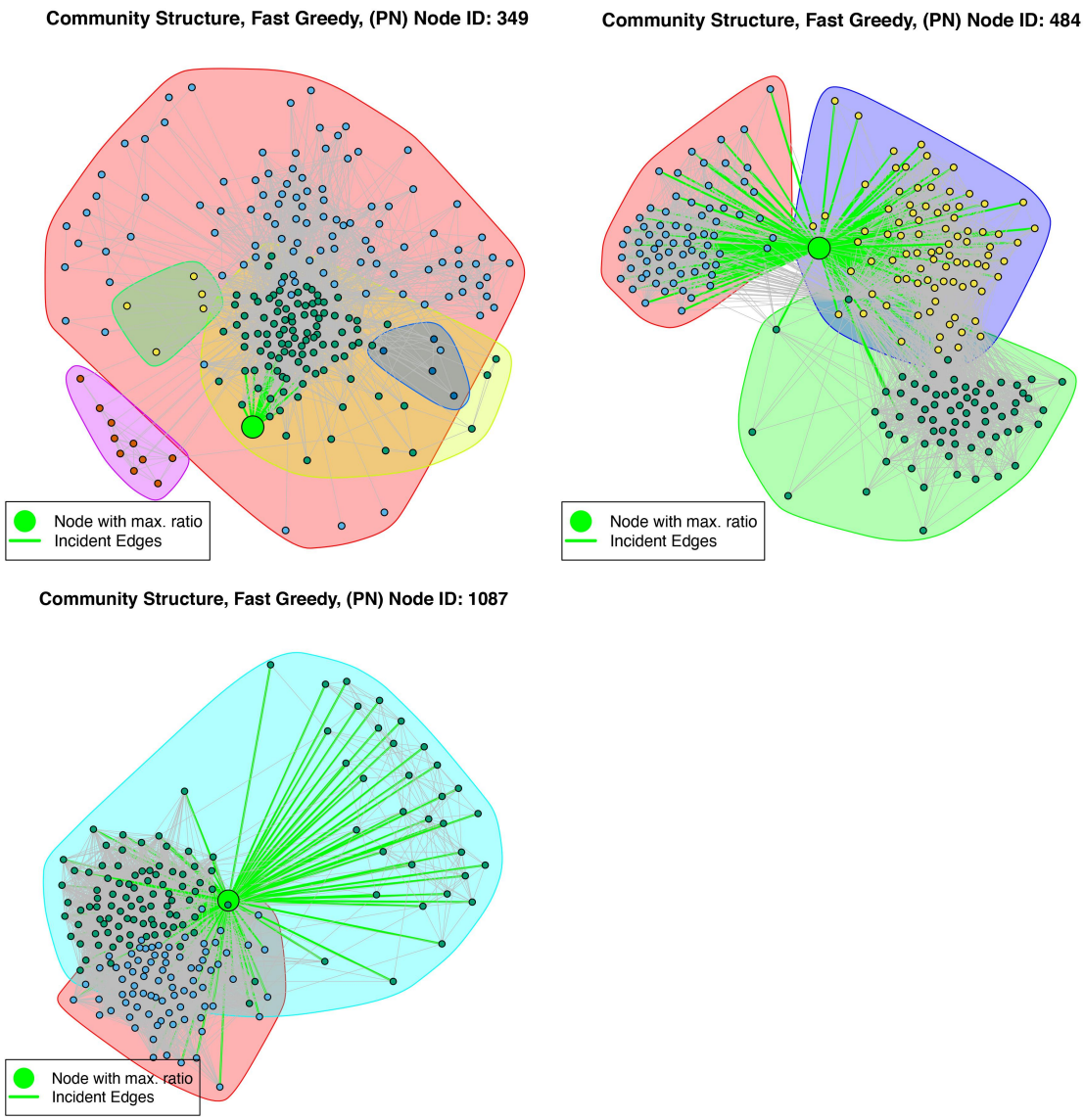


Figure 11: Community structure of personalized networks for core nodes, with maximum dispersion/embeddedness node and incident edges to that node highlighted.

Question 15:

We can make the following comments about a node revealed by each measure from Question 13 and Question 14:

Embeddedness:

We defined embeddedness of a node v_i is defined as the number of mutual vertices a given node shares with the core node v_c. Mathematically, the embeddedness between v_c and v_i in the personalized network P is given by:

$$\text{Embeddedness}^{P}_{v_i,v_c} = deg(v_i)^P - 1$$

To get the embeddedness directly from the original network:

$$\text{Embeddedness}_{v_i,v_c} = deg(v_i) - N_{v_i} - 1 = |deg(v_c) \cap deg(v_i)| - 1$$

Where N_{v_i} is the number of neighbors of v_i absent in the personalized network. Thus, embeddedness is defined as the overlap in the social circle of two users. Embeddedness depends on the number of nodes in the personalized network. The higher the number of nodes and connectivity among them in the social circle, the greater is the embeddedness. The expected value of embeddedness is dependent on the number of edges in the personalized network. In addition, large communities are likely to have a node with large value of embeddedness. In addition, compared to dispersion, the density of incident edges from the target node is much higher. This is by definition of embeddedness, which is proportional to the degree of the node. However, this is not a good measure of the strength of ties, as users with strong relationships are likely to be associated with users from various different communities.

Dispersion:

We defined the dispersion of a node is defined as the sum of distances between every pair of the mutual vertices the node shares with the core node, calculated in a modified subgraph graph with the target node and core node removed.

$$disp(u, v) = \sum_{s,t \in C_{uv}} d_v(s, t)$$

where, $C_{u,v} = u \cap v$, u and v denote the core and target nodes, d_v is the distance function. The range (as well as expected value) of dispersions depends on the number of edges (degree) of the core node in the personalized network, the higher the number of edges, the higher is the dispersion of a target node. In addition, dispersion measures the extent to which two people's mutual friends are not themselves well-connected, in other words, the mutual nodes of two strongly connected nodes are not well connected and must display a dispersed

structure. As a result, the density of incident edges from a node with maximum dispersion will likely be lower than a node with maximum embeddedness, since the network must display a dispersed structure with connectedness among users from various different communities. Dispersion is going to be higher for those nodes whose connections are farther apart (dispersed) than the nodes with dense connectedness. As a result, dispersion is a better measure of strength of ties or relationship over embeddedness.

Dispersion/Embeddedness:

The maximum value of dispersion/embeddedness (normalization of dispersion) occurs when dispersion of a node is very high while the embeddedness is very low. Such a node is likely to have mutual friends from different communities with stronger ties. The ratio is higher for smaller networks with dispersed structures, with friends that are themselves not strongly connected.

Question 16:

For this question, we create a personalized network with core node 415, using `make_ego_graph()` function on the original Facebook network. Afterwards, we find the list of all nodes, denoted as N_r with degree 24. **The value of $|N_r|$ was 11**.

Question 17:

In this question, we are asked to compute the average accuracy of friendship recommendation algorithms using three different neighbourhood based measures and find out which one is the best for the personalized network found in Question 16. Let S_i be the neighbour set of node i in the personalized network and S_j be the neighbour set of node j in the personalized network. The three measures are as follows:

- Common neighbours (i, j): $|S_i \cap S_j|$. Common neighbours is based on the assumption that if two nodes have many common neighbours, then the probability of them being connected in the future is high as well. The likelihood of future connectedness between two users is directly proportional to the number of mutual friends the users have. The score is based on the notion that people with common neighbours will be introduced to each other by that mutual friend (closing a triangle).
- Jaccard (i, j): $\frac{S_i \cap S_j}{S_i \cup S_j}$. Jaccard's coefficient is used to compute similarity of sample sets in statistics. In link prediction, all the friends of a node are denoted as a set and the prediction is done by ranking the similarity of the neighbour set for each pair of nodes. It is based on the notion that two users may have many mutual friends, but not all of them are due to strong ties when compared to the overall number of neighbours of each user. The coefficient ranges from 0 to 1. It takes into account the relative number of common neighbours, adjusted for degree of nodes.
- Adamic Adar (i, j): $\sum_{k \in S_i \cap S_j} \frac{1}{\log(|S_k|)}$. It is based on the notion that if a mutual friend of two users has lots of friends, then it is less likely that the mutual friend will introduce the two people to each other compared to the case when the mutual friend had fewer neighbours. The more friends a node has, the lower the score, weighing neighbours with fewer friends more heavily. In other words, someone with few friends are more likely to introduce his friends to each other than someone with a lot of friends. The mutual friend of a pair of users with few neighbours contributes more to the Adamic Adar score.

The steps to compute the average accuracy are as follows:

- Compute the average accuracy for each user in the list N_r. To do so, repeat the following steps 10 times:
  - Remove each friend of user i in the list N_r at random with probability 0.25 and store them in a new list referred to as R_i.
  - Recommend $|R_i|$ new friends to user i. To do so, use one of the three measures to compute the similarity between user i and friends that are not neighbours of user i, picking top $|R_i|$ nodes with highest scores. The recommendation list is denoted as P_i
  - The accuracy is given as: $\frac{|P_i \cap R_i|}{|R_i|}$

- Compute the average accuracy of the algorithm by averaging across the accuracies of the users in the list N_r.

We obtain the following accuracies for the three measures:

- **Common neighbours: 88.78%**
- **Jaccard: 85.30%**
- **Adamic Adar: 88.06%**

We see that the common neighbours algorithm performs the best, followed by Adamic Adar and Jaccard. Note that the accuracies are very close for all three networks (within 4% of each other). In addition, Adamic Adar and common neighbours perform very similarly, with < 1% difference in accuracy. We can make several inferences:

- *Common neighbours versus Jaccard*: Jaccard's coefficient is based on the assumption that only strongly tied nodes contribute to future friend recommendation (intersection over union). In other words, if the degree of the target nodes is high, then the Jaccard's coefficient will be low, with the implicit assumption that two users with too many friends may not have strong ties within their communities. However, if the nodes within a network are themselves not strongly tied in a network, then the Jaccard's coefficient will perform slightly worse than just taking the intersection over mutual friends (common neighbours). This is particularly prominent in small networks, where the probability of having strong ties is lower than that of larger networks.
- *Common neighbours versus Adamic Adar:* Theoretically, Adamic Adar should perform better than common neighbours, as it gives more importance to nodes with few neighbours (rare neighbours), particularly the cases where $S_i \cap S_j = 1$. However, for such a small network, it may well be the case that the occurrence of such cases is low (explained by the low Jaccard score), and most nodes have a large number of neighbors in common. This is why common neighbours and Adamic Adar perform very similar to each other.
- *Why are the accuracies of all three algorithms similar?* The accuracies of the algorithms are within 4% of each other. This is mainly because we are operating on a small personalized network with relatively fewer nodes and edges compared to large social networks. In addition, we are recommending friends to users with a limited degree. The reach of the network is not all-inclusive of implicit assumptions made by each of the measures and instead adhere to the most common assumptions, which does not yield significant differences in link prediction algorithm performance. In other words, we are computing on local features over global similarity features, which does not truly reflect how social network friendship recommendations work on a large scale.

Question 18:

There are 57 personal networks out of 132 ego-networks**.** We found this by reading all the .circles files in the dataset and counting the number of lines in each file. Personal networks are those networks for users with more than 2 circles.

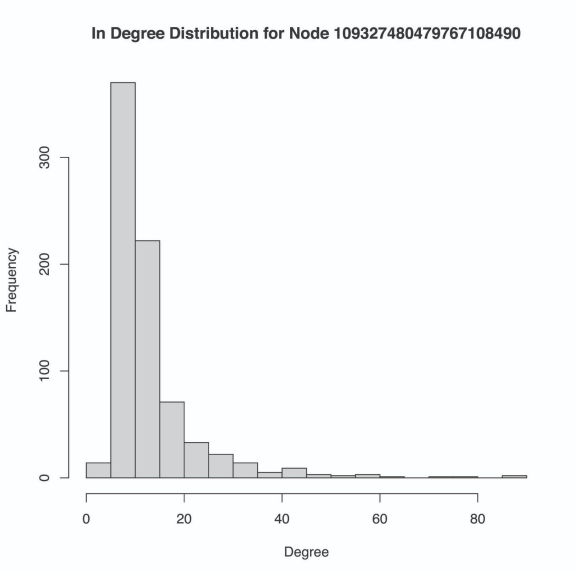
Question 19:

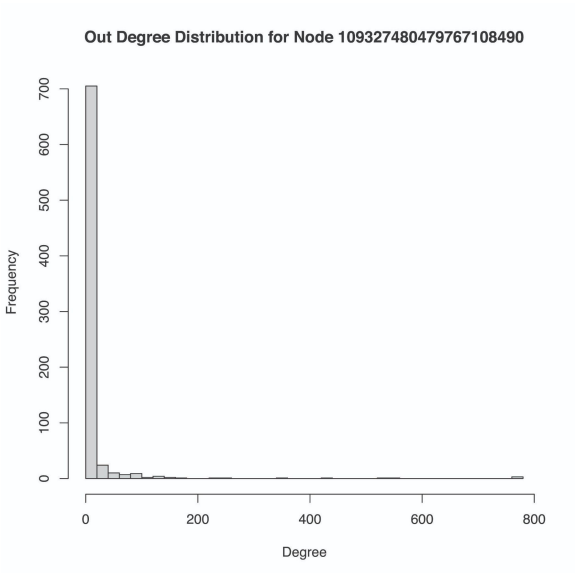
Figure 12 shows the in-degree and out-degree for 3 personalized networks with node IDs 109327480479767108490, 115625564993990145546 and 101373961279443806744. Mathematically, the in-degree of node i in a network is given by:

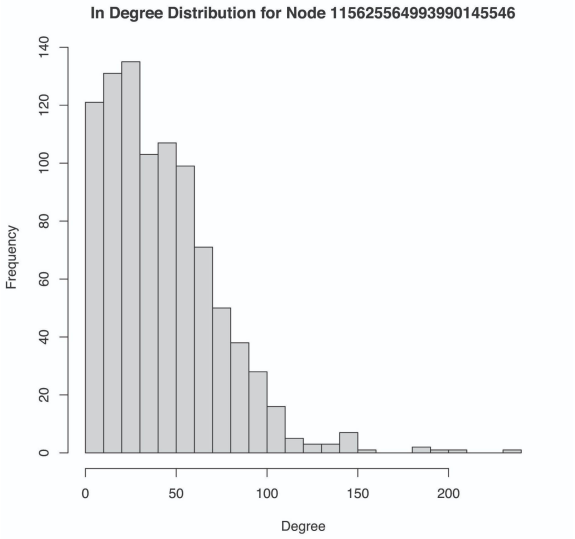
$$deg(i)^{\text{in}} = \sum_j A_{ij}$$

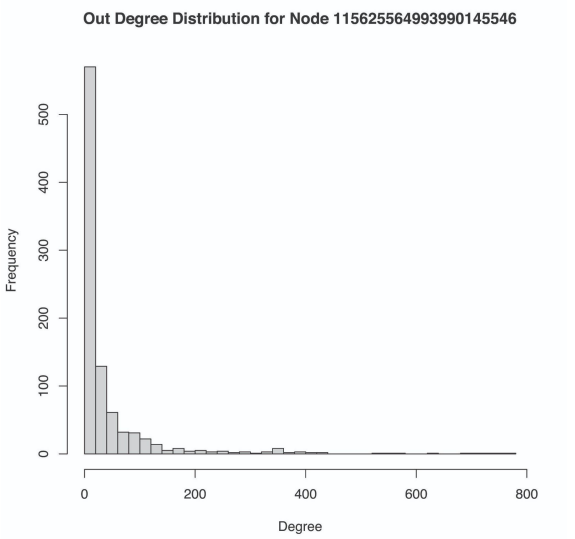
where, A_ij is defined as the node-node incidence matrix or adjacency matrix. Similarly, the out degree is given as:

$$deg(i)^{\text{out}} = \sum_j A_{ji}$$









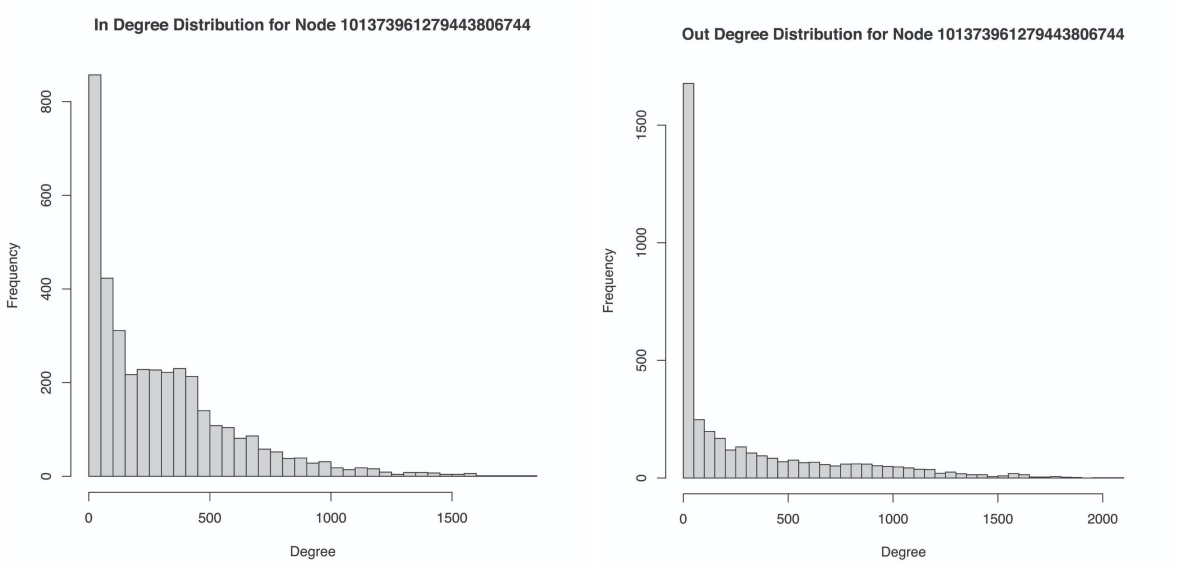


Figure 12: In and out-degree distributions for 3 personalized networks in the Google+ dataset.

From Figure 12, we see that the out-degree distributions for all three networks are very similar to each other, following power-law distribution. The in-degree distributions show some variation across the three networks and roughly follows the power-law (though not as closely as out-degree distributions). For each network, the in-degree and out-degree distributions are significantly different.

- The out-degree distribution of the personalized network of node 109327480479767108490 is most heavily skewed to the right among the three out-degree distributions.
- For the personalized networks with node 109327480479767108490 and 115625564993990145546, while the expected value from the out-degree distributions are similar, the expected in-degree values are different, with a higher value for node 115625564993990145546 over 109327480479767108490. In addition, the in-degree distribution of the network with node 115625564993990145546 is roughly linear. This means that for the network with node 115625564993990145546, there are more incoming connections to each of the nodes over the network with node 109327480479767108490. This means that the network with node 115625564993990145546 forms communities of strong connectedness with sparse inter-community connections rather than depending on a few popular users or hubs compared to the network with node 109327480479767108490.
- The out-degree distribution of the network with node 101373961279443806744 rolls off most slowly, indicating absence of clear community structures and low modularity.

Question 20:

In this question, we are asked to extract the community structures (using walktrap community detection) and modularity scores of the three personalized networks from Question 19. Figure 13 shows the community structures.

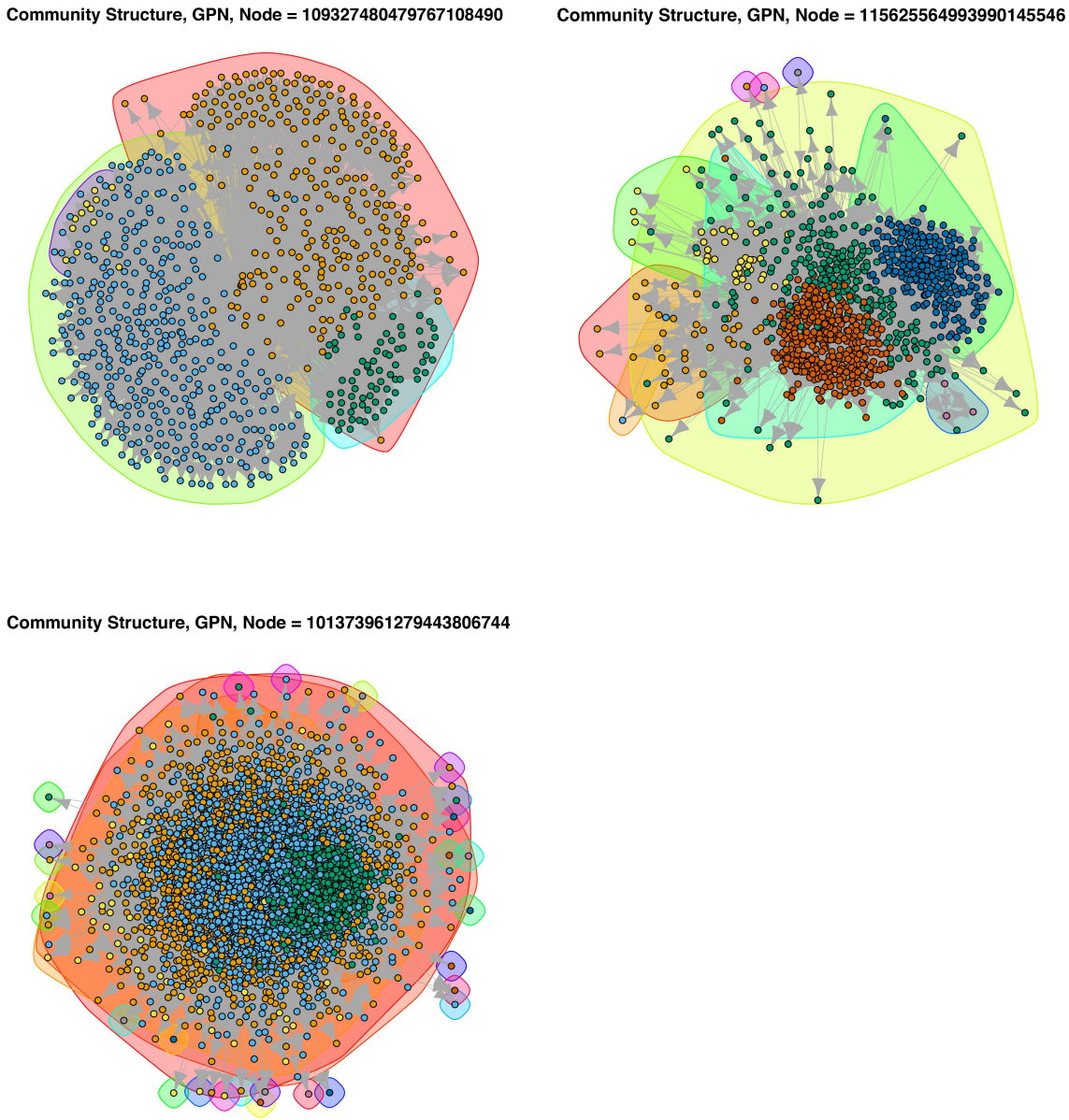


Figure 13: Community structure of three personalized networks from Google+ dataset

The modularity scores are :

- **Node 109327480479767108490: 0.252765387296677**

- **Node 11562556493939901455546: 0.319472551345825**
- **Node 10137396127944380674 4: 0.191090270876884**

From Figure 13 and the modularity scores, we see that the modularity scores are not similar. Node 10137396127944380674 4 has the lowest modularity score, indicating the network is least capable of being divided into densely packed modules with strong connectedness and sparse interconnections among communities. This is visible from Figure 13, where we observe that most of the nodes in the personalized network for node 10137396127944380674 4 are grouped in a single chunk in the centre of the plot. We also observed that the personalized network for node 10137396127944380674 4 has the highest number of edges m and nodes among all networks. Intuitively, a higher value of m indicates that an incoming node is connected to a larger number of older nodes. While this should result in strong intra-community connectedness, the global sparsity among different communities is lost due to the connectedness requirement brought on by high values of m, resulting in edges being formed among otherwise distinct clusters and hence weakening the community structures. Mathematically,

$$Q(P) = \sum \frac{1}{2m} \sum_{i,j \in C_i} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$$

If the number of edges in the network m (not to be confused with the actual m we are talking about, which is the number of old nodes the incoming node connects with) increases, then $Q(P)$ dtops. As a result, less clusters are formed in the overall graph.

The modularity score is highest for node 11562556493939901455546. From Figure 13, we can clearly see dense communities or clusters being formed in the community structure plot, with sparse interconnectivity among these clusters.

Question 21:

Homogeneity: A clustering result satisfies homogeneity if all of its clusters contain only data points which are members of a single class or circle. Mathematically, homogeneity *h* is defined in terms of conditional entropy of labels or circles *C* given cluster assignments or circles *K* and is independent of absolute value of ground truth labels.

$$h = 1 - \frac{H(C|K)}{H(C)}$$

$$H(C|K) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|}\frac{n_{c,k}}{n} \cdot \log\left(\frac{n_{c,k}}{n_k}\right)$$

$$H(X) = -\sum_{h=1}^{|H|}\frac{n_h}{n} \cdot \log\left(\frac{n_h}{n}\right)$$

$$n_{c,k} = n_k \cap n_c$$

H(X) denotes the entropy of partition *X*, where *X* denotes non-overlapping groups of sample points. The information entropy is maximized when the sizes of the partitions are equal and minimized when some group within the partition takes up all the data points. The homogeneity score is maximized when each cluster K_i contains samples only from C_i, i.e. $H(C|K) = 0$.

Completeness: A clustering result satisfies completeness if all the data points that are members of a given circle are elements of the same community. Mathematically, completeness is defined in terms of conditional entropy of clusters *K* given ground truth labels *C*.

$$c = 1 - \frac{H(K|C)}{H(K)}$$

Completeness if maximized when each ground truth class C_i is part of some cluster K_i. In the ideal case ($c = 1$), a single cluster should encompass all members of a circle.

Question 22:

The *h* and *c* values for the 3 personalized networks are given in Table 3.

Table 3: Homogeneity and Completeness scores for three personalized networks in Google+ network

Node ID	**Homogeneity**, *h*	**Completeness**, *c*
109327480479767108490	0.8518851	0.3298739
115625564993990145546	0.4518903	-3.423962
101373961279443806744	0.003866707	-1.504238

We can make several observations from Table 3:

- For the network with node ID 109327480479767108490, we observe a relatively high value of homogeneity. This indicates that given a community, most of its users belong to the same circle. The completeness score is low but positive, which indicates that some of the users belonging to one circle have not been classified into one community. We can observe these facts from Figure 13. In relation to completeness score, we observe that not all orange dots have been classified to the red community, some of them have been classified into the yellow community and green community. However, comparatively, the completeness score is still much higher than the other two networks. The scores indicate that the members of the personalized network of node ID 109327480479767108490 are well-segregated into individual communities with few overlaps among different circles.
- For the network with node ID 115625564993990145546, the homogeneity score is almost half of that of the network with node ID 109327480479767108490. This means that more users from different circles in this network have been wrongly classified into a single community. This is evident from the community structure plot in Figure 13. In addition, the completeness score is low and negative. This means that $H(K|C) > H(K)$. This can happen when some of the circles have not been assigned to any community and have been lumped together (each community has a few circles). In other words, a negative completeness indicates a sparse C and points towards a large mismatch between the number of circles and number of communities.
- For the network with node ID 101373961279443806744, the homogeneity score is the lowest of all three, indicating most users from different circles in this network have been wrongly classified into a single community. This is also evident from the low modularity scores, which indicates that this network is least capable of being divided into dense communities with sparse inter-community connections. In other words, the communities are not well-separated. The completeness score is again negative but higher than the network with node 115625564993990145546, indicating

$H(K|C) > H(K)$. The reasons for negative completeness have been discussed in the previous point.

Project2_Saha_Young.ipynb 16-27-37-689

April 28, 2021

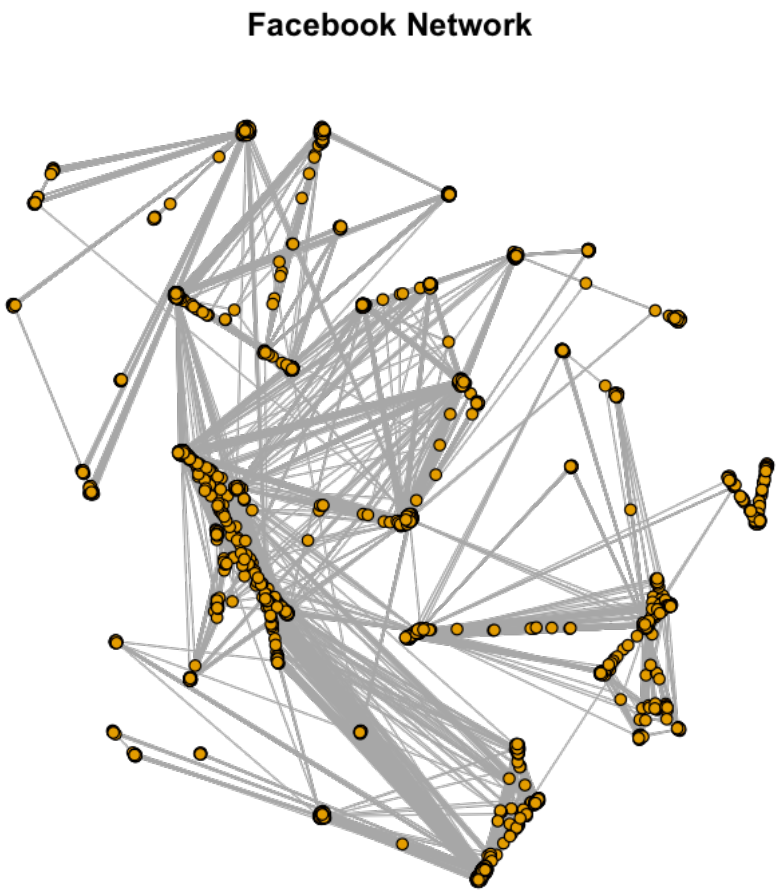
[35]:
```
library(igraph)
```

1 Facebook Network

1.0.1 Question 1.1

[36]:
```
el <- read.table("facebook_combined.txt")
g <- graph.data.frame(el, directed=FALSE)
plot(g,vertex.label="",vertex.size=3,main = "Facebook Network")
dev.copy2eps(file='Q1_1.eps')
```

pdf: 2



[37]:
```
print(sprintf("Number of nodes: %d",gorder(g)))
print(sprintf("Number of edges: %d", gsize(g)))
```

```
[1] "Number of nodes: 4039"
[1] "Number of edges: 88234"
```

1.0.2 Question 1.2

[38]:
```
is.connected(g)
```

TRUE

1.0.3 Question 2

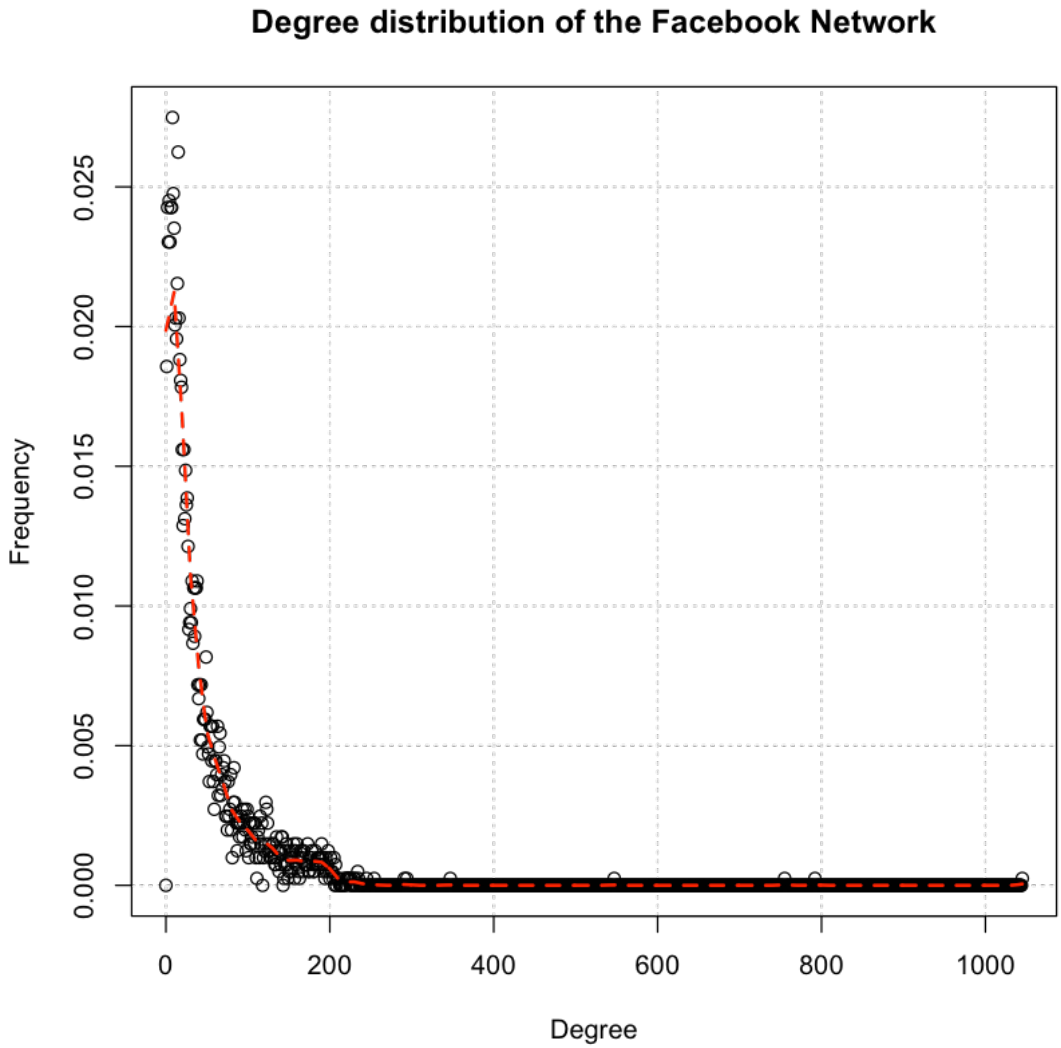
[39]:
```
print(diameter(g))
```

```
[1] 8
```

1.0.4 Question 3

[6]:
```
plot(seq_along(degree.distribution(g)) - 1, degree.distribution(g),
         main="Degree distribution of the Facebook␣
 ↪Network",xlab="Degree",ylab="Frequency",grid())
lines(lowess(seq_along(degree.distribution(g)) - 1,degree.distribution(g),f = 0.
 ↪027), col="red",lwd = 2,lty=2)
dev.copy2eps(file='Q3.eps')
```

pdf: 2



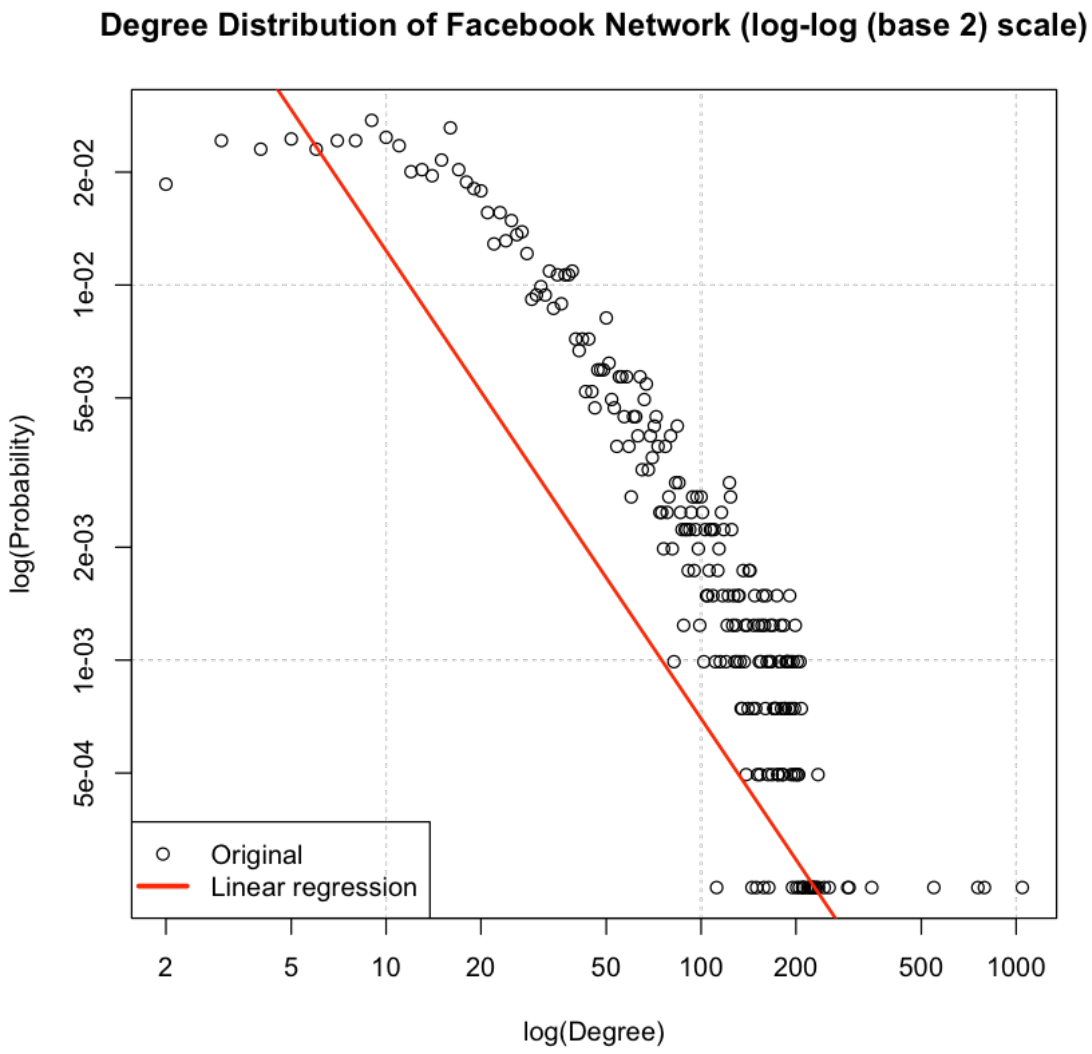
[7]:
```
print(sprintf("Average degree: %f", mean(degree(g))))
```

```
[1] "Average degree: 43.691013"
```

1.0.5 Question 4

[50]:
```
deg_dist = degree.distribution(g)
deg <- c(1:length(deg_dist))[which(deg_dist !=0, arr.ind = TRUE)]
dist <-deg_dist[which(deg_dist !=0, arr.ind = TRUE)]
plot(deg,dist,main="Degree Distribution of Facebook Network (log-log (base 2)␣
 ↪scale)",
     xlab="log(Degree)",ylab="log(Probability)",grid(),col="black",log="xy")
abline(lm(log(dist) ~ log(deg)),col="red",lwd=2)
legend('bottomleft', legend = c("Original", "Linear regression"),
       lty = c(0, 1), lwd = c(1,3), pch=c(1,NA),
       col = c('black','red'))
dev.copy2eps(file='Q4.eps')
```

pdf: 2



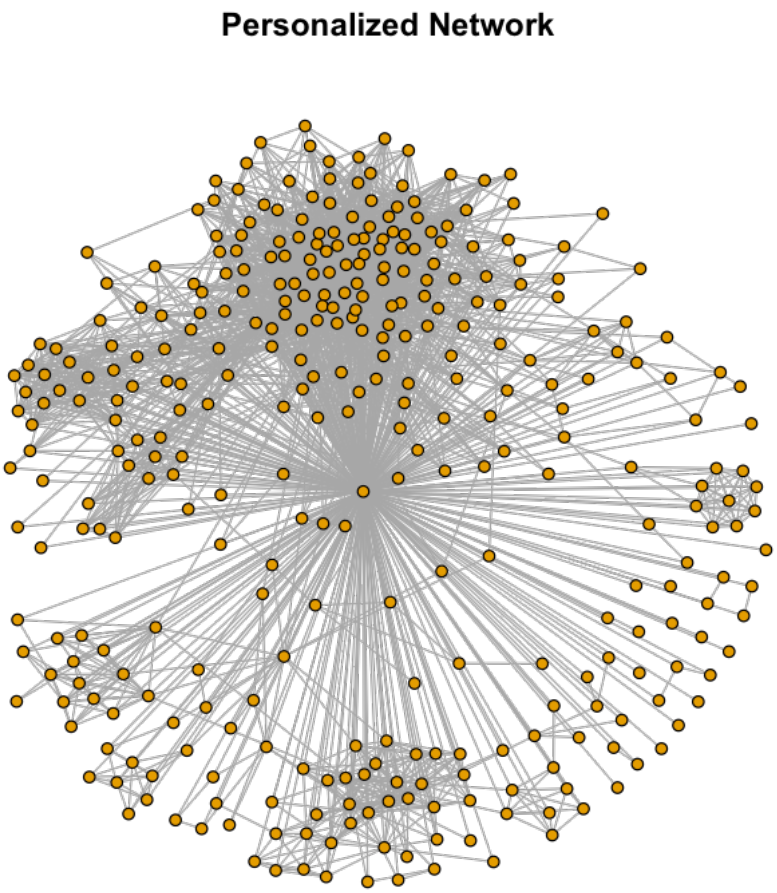
[49]:
```
print("Slope and intercept for the line")
print(lm(log(dist) ~ log(deg)))
```

```
[1] "Slope and intercept for the line"

Call:
lm(formula = log(dist) ~ log(deg))

Coefficients:
(Intercept)     log(deg)
    -0.6611      -1.2475
```

1.0.6 Question 5

[10]:
```
eg = make_ego_graph(g, order = 1, nodes = V(g), mindist = 0)[[1]]
plot(eg,vertex.label="",vertex.size=3,main = "Personalized Network")
dev.copy2eps(file='Q5.eps')
```

pdf: 2



[11]:
```
print(sprintf("Number of nodes: %d",gorder(eg)))
print(sprintf("Number of edges: %d", gsize(eg)))
```

```
[1] "Number of nodes: 348"
[1] "Number of edges: 2866"
```

1.0.7 Question 6

[12]:
```
print(diameter(eg))
```

```
[1] 2
```

1.0.8 Question 8

[13]:
```
coreNodes <- which(neighborhood.size(g,1,nodes=V(g))>201)
print(length(coreNodes))
degCoreNodes <- mean(degree(g, v=V(g)[coreNodes]))
print(degCoreNodes)
```

```
[1] 40
[1] 279.375
```

1.0.9 Question 9

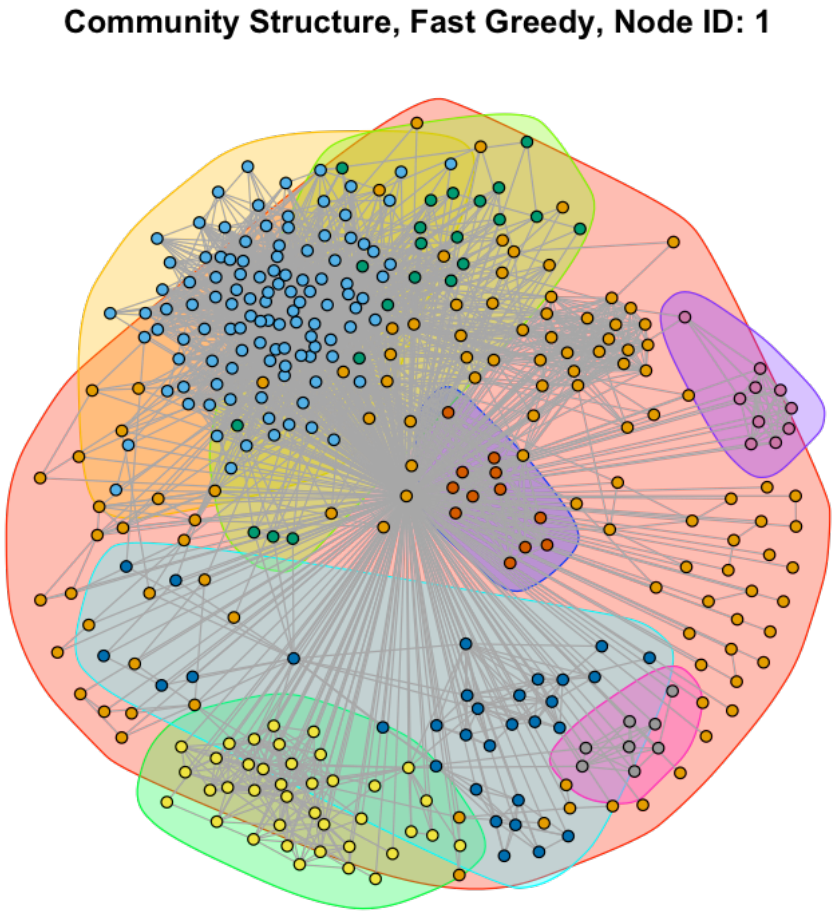
[28]:
```
node_list <- c(1,349,484,1087,108)
g <- read.graph("facebook_combined.txt", format="edgelist", directed=FALSE)
eg <- make_ego_graph(g, order = 1, nodes = V(g)[node_list])
```

[15]:
```
i = 1
net = eg[[i]]
fc <- cluster_fast_greedy(net)
eb <- cluster_edge_betweenness(net)
imc <- cluster_infomap(net)
print(sprintf("Modularity, Fast Greedy, Node ID: %d: %f",node_list[i],␣
 ↪modularity(fc)))
print(sprintf("Modularity, Edge-Betweenness, Node ID: %d: %f",node_list[i],␣
 ↪modularity(eb)))
print(sprintf("Modularity, Infomap, Node ID: %d: %f",node_list[i],␣
 ↪modularity(imc)))
plot(eg[[i]],mark.groups = fc,vertex.size=3,vertex.color=fc$membership,vertex.
 ↪label="",main = sprintf("Community Structure, Fast Greedy, Node ID: %d",␣
 ↪node_list[i]))
dev.copy2pdf(file=sprintf('Q9fc%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = eb,vertex.size=3,vertex.color=eb$membership,vertex.
 ↪label="",main = sprintf("Community Structure, Edge-Betweenness, Node ID:␣
 ↪%d", node_list[i]))
dev.copy2pdf(file=sprintf('Q9eb%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = imc,vertex.size=3,vertex.color=imc$membership,vertex.
 ↪label="",main = sprintf("Community Structure, Infomap, Node ID: %d",␣
 ↪node_list[i]))
dev.copy2pdf(file=sprintf('Q9imc%d.pdf',node_list[i]))
```

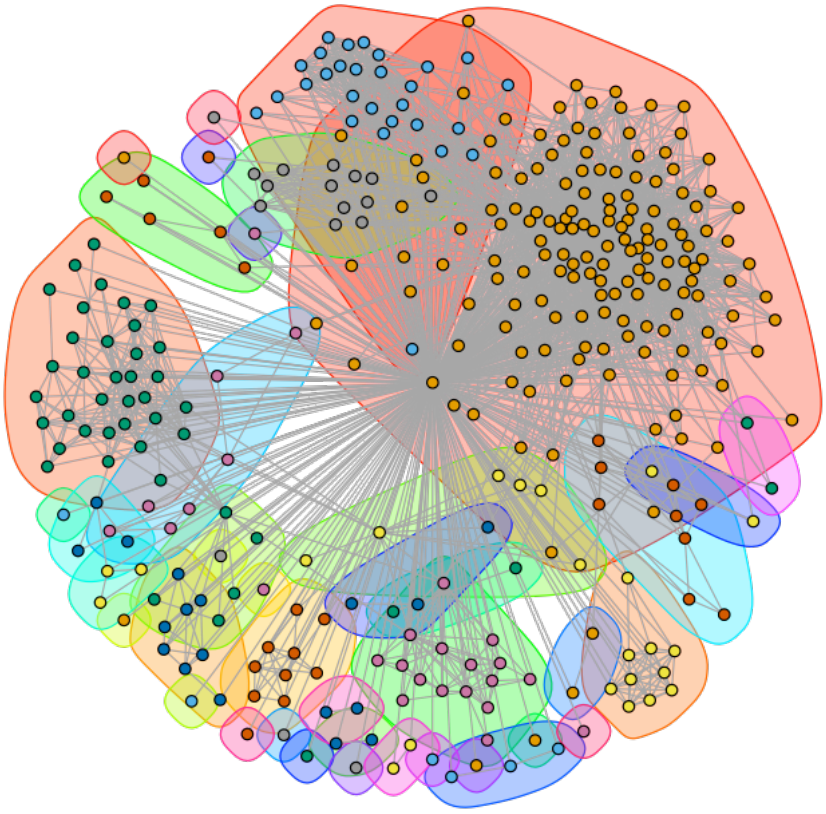
```
[1] "Modularity, Fast Greedy, Node ID: 1: 0.413101"
[1] "Modularity, Edge-Betweenness, Node ID: 1: 0.353302"
```

```
[1] "Modularity, Infomap, Node ID: 1: 0.394125"
```

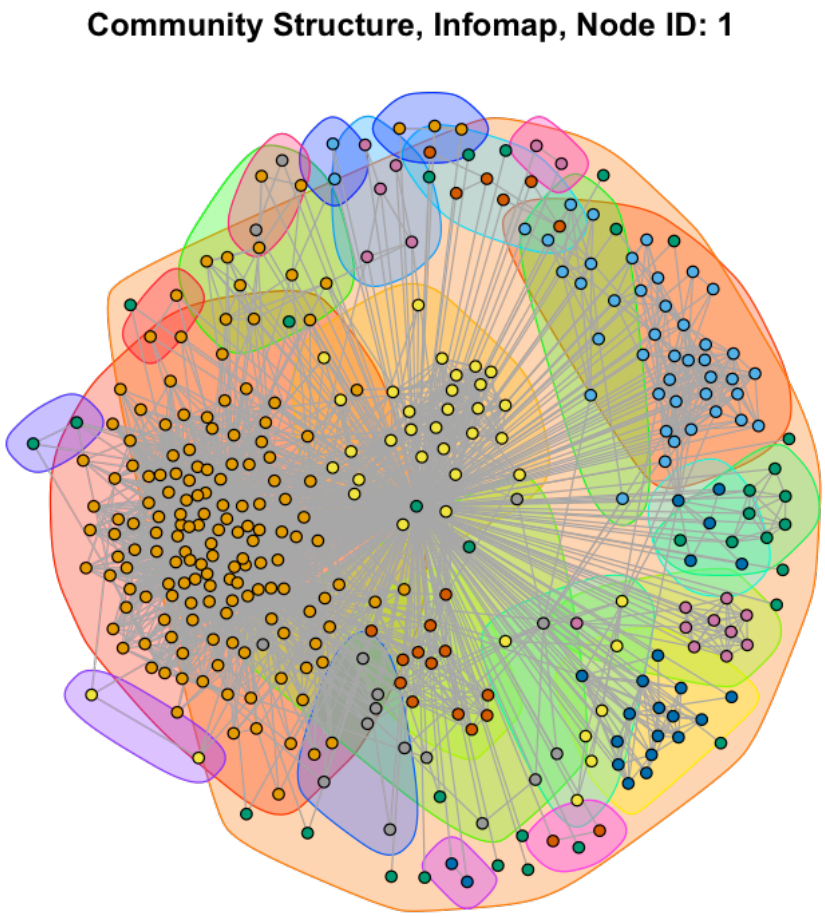
pdf: 2



pdf: 2



pdf: 2

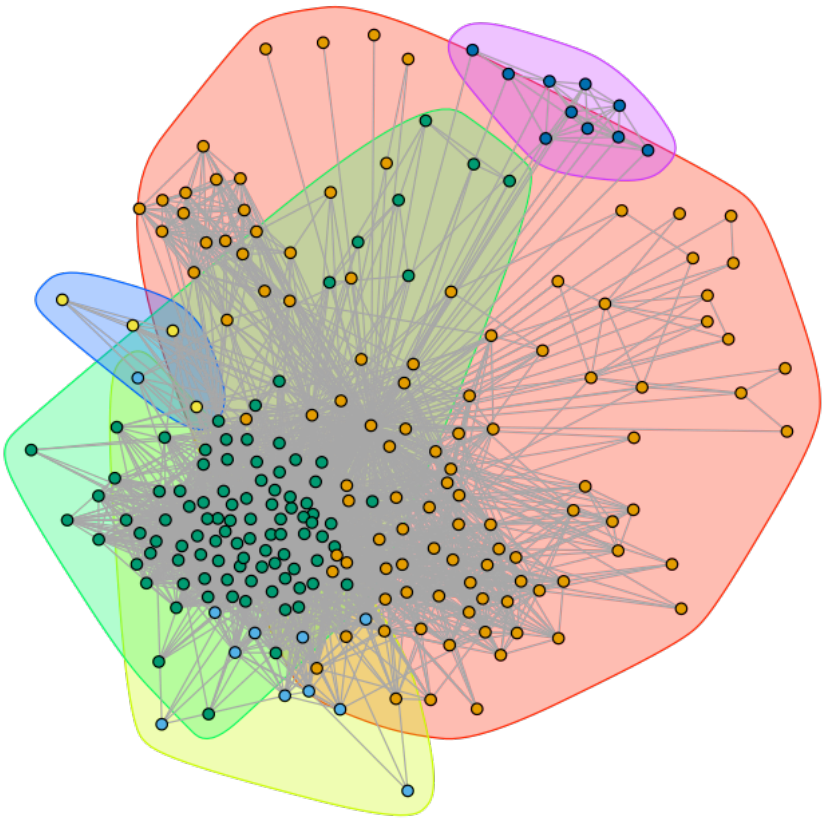


[16]:
```
i = 2
net = eg[[i]]
fc <- cluster_fast_greedy(net)
eb <- cluster_edge_betweenness(net)
imc <- cluster_infomap(net)
print(sprintf("Modularity, Fast Greedy, Node ID: %d: %f",node_list[i],␣
 ↪modularity(fc)))
print(sprintf("Modularity, Edge-Betweenness, Node ID: %d: %f",node_list[i],␣
 ↪modularity(eb)))
print(sprintf("Modularity, Infomap, Node ID: %d: %f",node_list[i],␣
 ↪modularity(imc)))
```

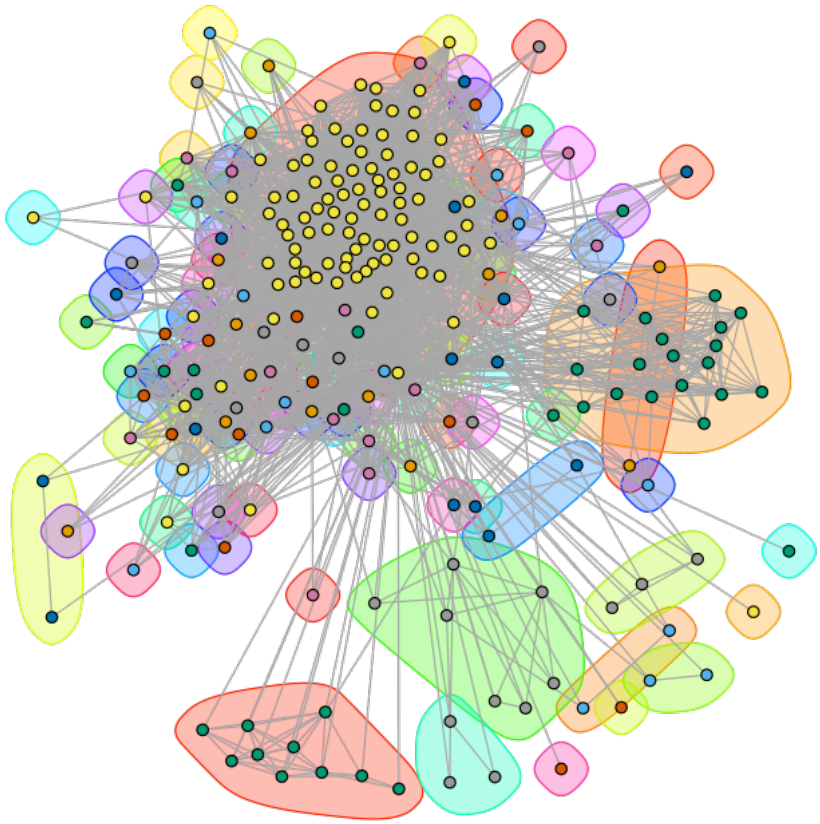
```
plot(eg[[i]],mark.groups = fc,vertex.size=3,vertex.color=fc$membership,vertex.
 ↪label="",main = sprintf("Community Structure, Fast Greedy, Node ID: %d",␣
 ↪node_list[i]))
dev.copy2pdf(file=sprintf('Q9fc%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = eb,vertex.size=3,vertex.color=eb$membership,vertex.
 ↪label="",main = sprintf("Community Structure, Edge-Betweenness, Node ID:␣
 ↪%d", node_list[i]))
dev.copy2pdf(file=sprintf('Q9eb%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = imc,vertex.size=3,vertex.color=imc$membership,vertex.
 ↪label="",main = sprintf("Community Structure, Infomap, Node ID: %d",␣
 ↪node_list[i]))
dev.copy2pdf(file=sprintf('Q9imc%d.pdf',node_list[i]))
```

```
[1] "Modularity, Fast Greedy, Node ID: 349: 0.251715"
[1] "Modularity, Edge-Betweenness, Node ID: 349: 0.133528"
[1] "Modularity, Infomap, Node ID: 349: 0.095464"
```

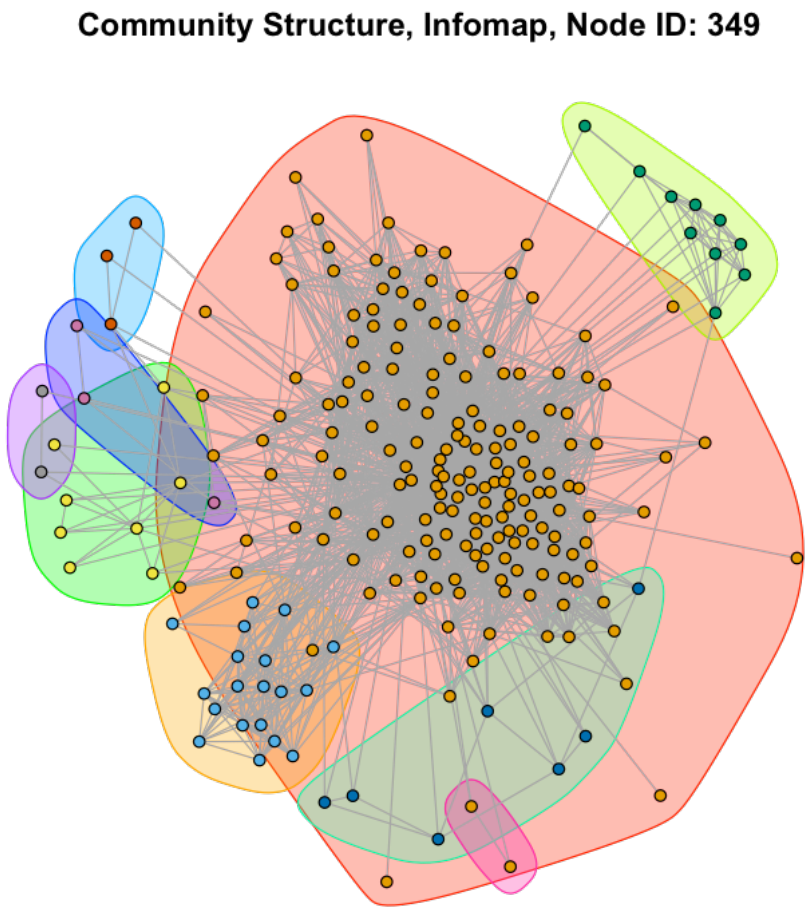
pdf: 2



pdf: 2



pdf: 2

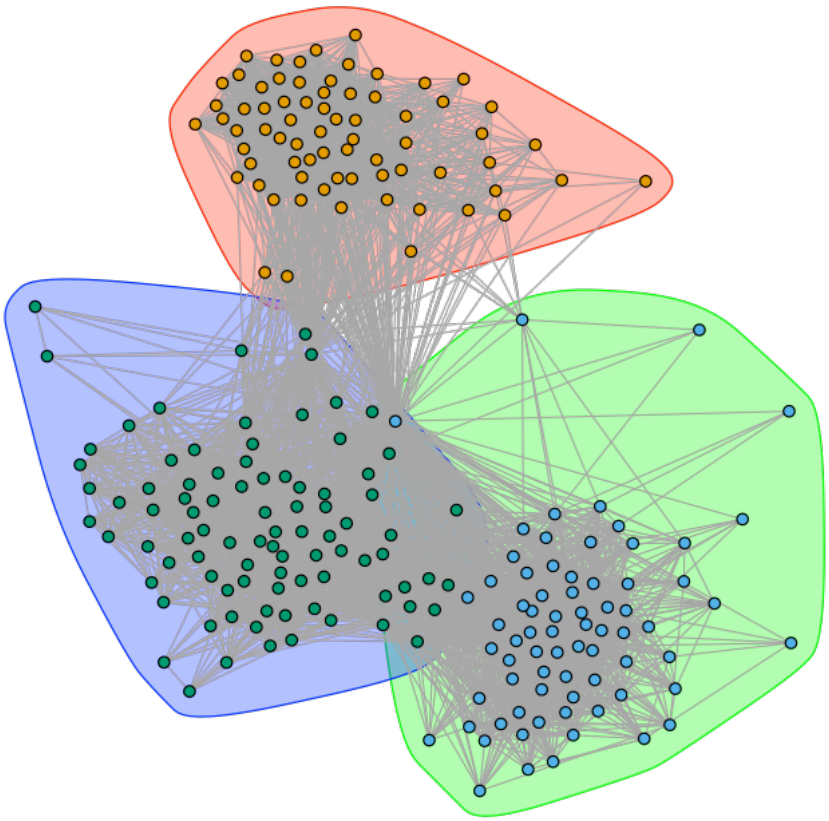


[17]:
```
i = 3
net = eg[[i]]
fc <- cluster_fast_greedy(net)
eb <- cluster_edge_betweenness(net)
imc <- cluster_infomap(net)
print(sprintf("Modularity, Fast Greedy, Node ID: %d: %f",node_list[i],␣
 ↪modularity(fc)))
print(sprintf("Modularity, Edge-Betweenness, Node ID: %d: %f",node_list[i],␣
 ↪modularity(eb)))
print(sprintf("Modularity, Infomap, Node ID: %d: %f",node_list[i],␣
 ↪modularity(imc)))
```

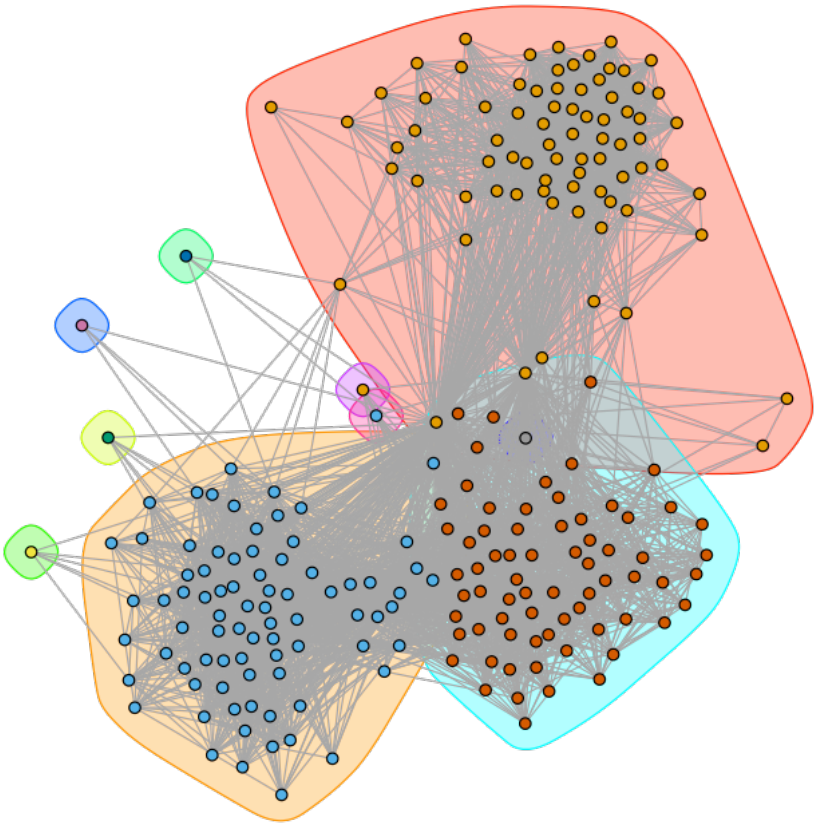
```
plot(eg[[i]],mark.groups = fc,vertex.size=3,vertex.color=fc$membership,vertex.
 ↪label="",main = sprintf("Community Structure, Fast Greedy, Node ID: %d",␣
 ↪node_list[i]))
dev.copy2pdf(file=sprintf('Q9fc%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = eb,vertex.size=3,vertex.color=eb$membership,vertex.
 ↪label="",main = sprintf("Community Structure, Edge-Betweenness, Node ID:␣
 ↪%d", node_list[i]))
dev.copy2pdf(file=sprintf('Q9eb%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = imc,vertex.size=3,vertex.color=imc$membership,vertex.
 ↪label="",main = sprintf("Community Structure, Infomap, Node ID: %d",␣
 ↪node_list[i]))
dev.copy2pdf(file=sprintf('Q9imc%d.pdf',node_list[i]))
```

```
[1] "Modularity, Fast Greedy, Node ID: 484: 0.507002"
[1] "Modularity, Edge-Betweenness, Node ID: 484: 0.489095"
[1] "Modularity, Infomap, Node ID: 484: 0.515279"
```

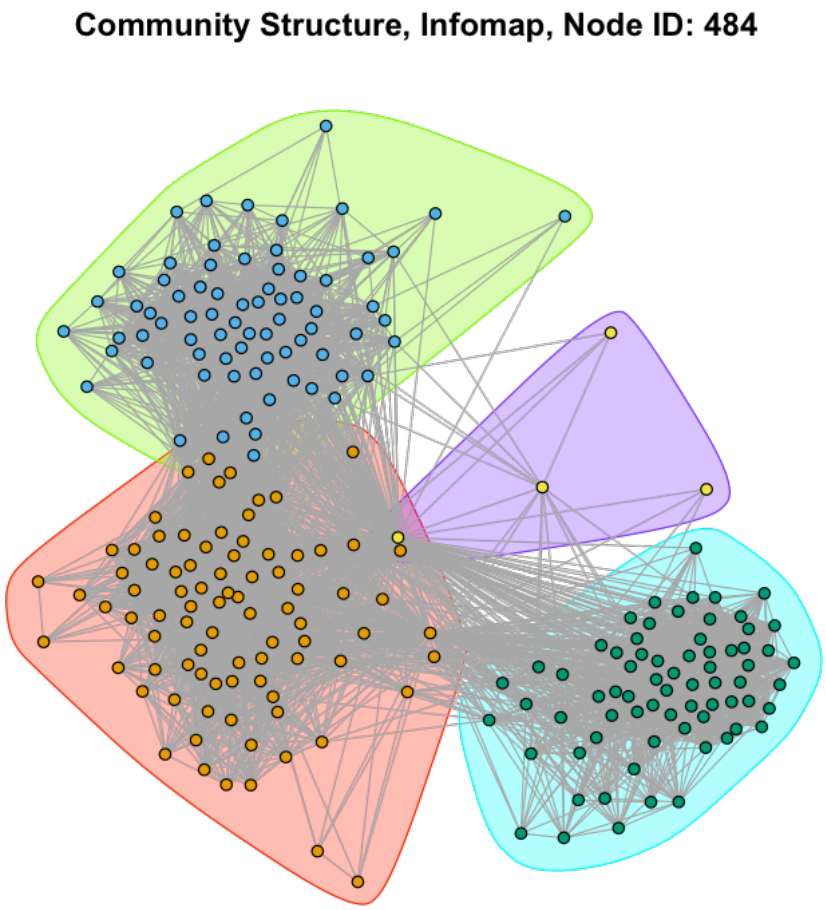
pdf: 2



pdf: 2



pdf: 2

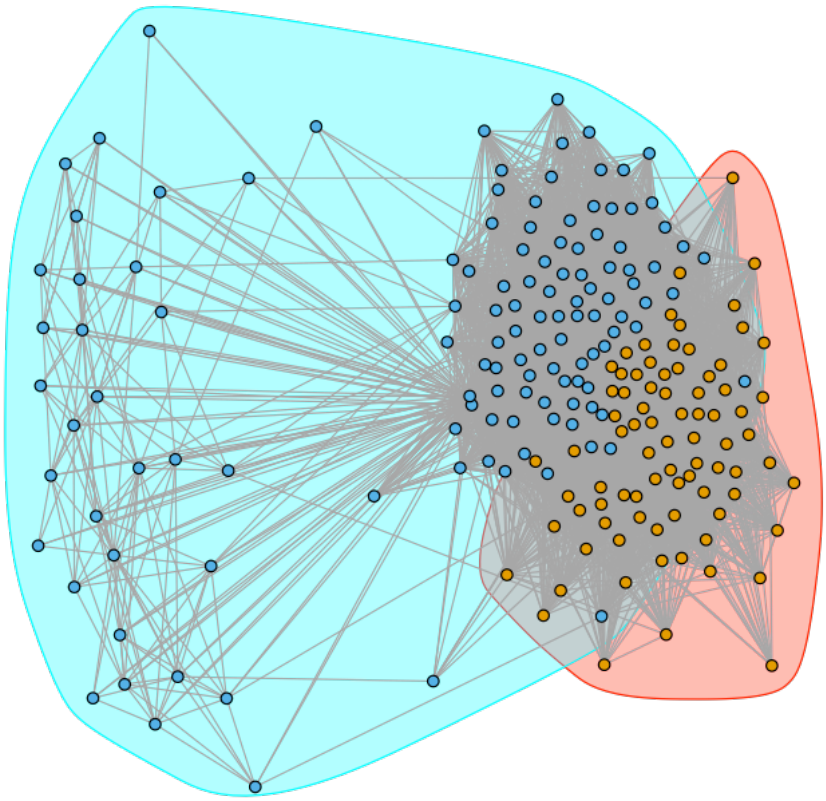


[18]:
```
i = 4
net = eg[[i]]
fc <- cluster_fast_greedy(net)
eb <- cluster_edge_betweenness(net)
imc <- cluster_infomap(net)
print(sprintf("Modularity, Fast Greedy, Node ID: %d: %f",node_list[i],␣
 ↪modularity(fc)))
print(sprintf("Modularity, Edge-Betweenness, Node ID: %d: %f",node_list[i],␣
 ↪modularity(eb)))
print(sprintf("Modularity, Infomap, Node ID: %d: %f",node_list[i],␣
 ↪modularity(imc)))
```

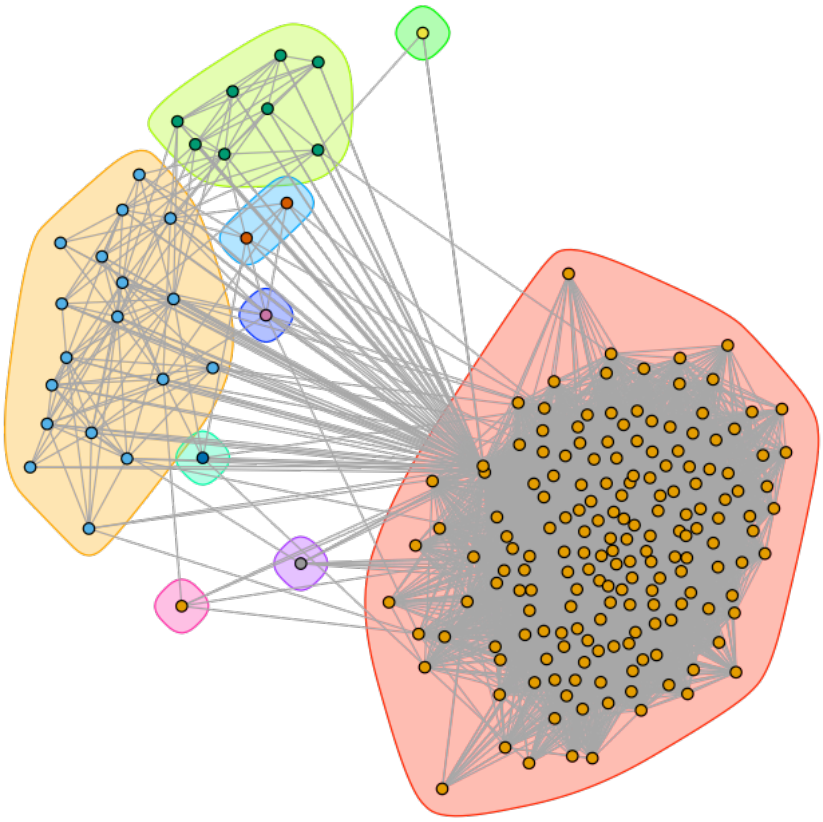
```
plot(eg[[i]],mark.groups = fc,vertex.size=3,vertex.color=fc$membership,vertex.
 ↪label="",main = sprintf("Community Structure, Fast Greedy, Node ID: %d",␣
 ↪node_list[i]))
dev.copy2pdf(file=sprintf('Q9fc%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = eb,vertex.size=3,vertex.color=eb$membership,vertex.
 ↪label="",main = sprintf("Community Structure, Edge-Betweenness, Node ID:␣
 ↪%d", node_list[i]))
dev.copy2pdf(file=sprintf('Q9eb%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = imc,vertex.size=3,vertex.color=imc$membership,vertex.
 ↪label="",main = sprintf("Community Structure, Infomap, Node ID: %d",␣
 ↪node_list[i]))
dev.copy2pdf(file=sprintf('Q9imc%d.pdf',node_list[i]))
```

```
[1] "Modularity, Fast Greedy, Node ID: 1087: 0.145531"
[1] "Modularity, Edge-Betweenness, Node ID: 1087: 0.027624"
[1] "Modularity, Infomap, Node ID: 1087: 0.026907"
```

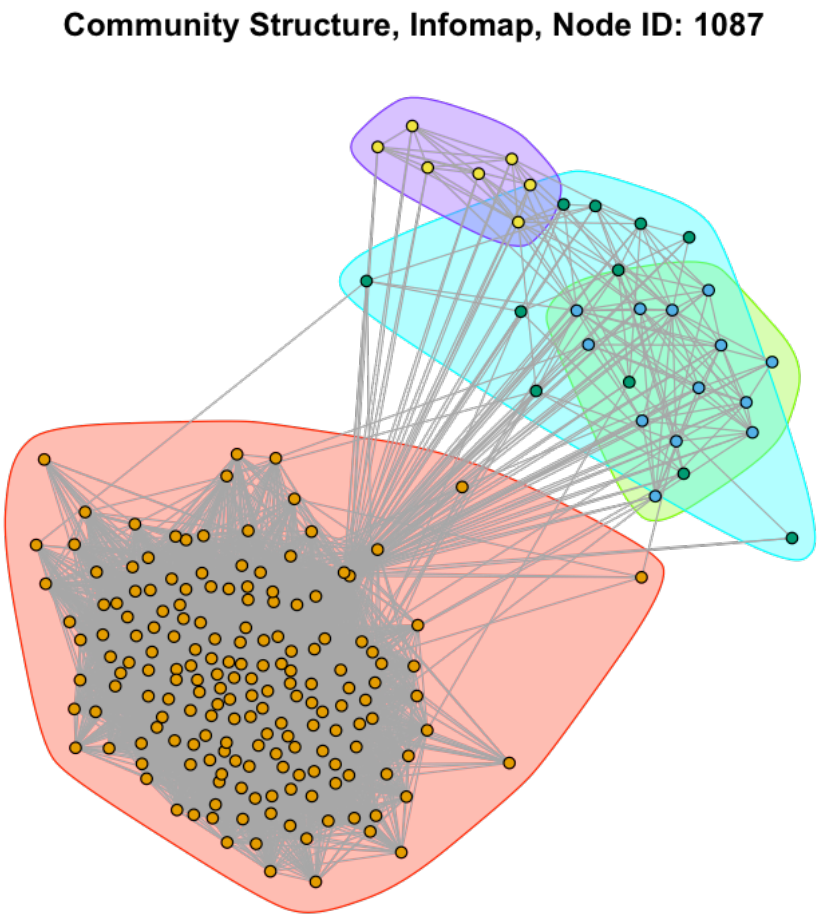
pdf: 2



pdf: 2



pdf: 2

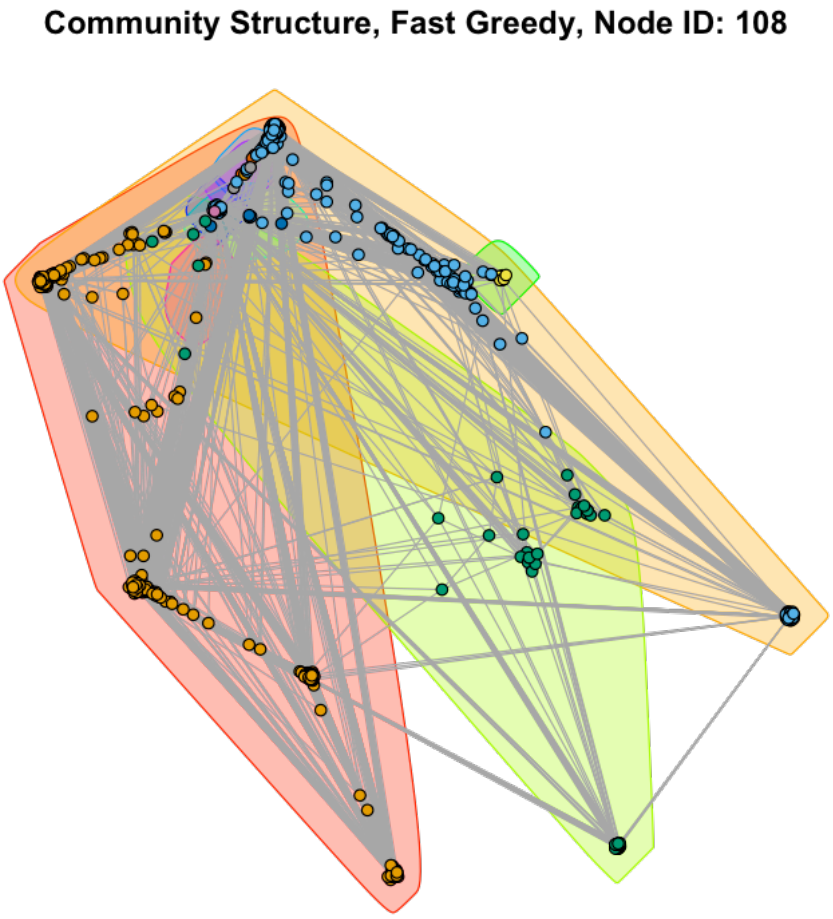


[29]:
```
i = 5
net = eg[[i]]
fc <- cluster_fast_greedy(net)
eb <- cluster_edge_betweenness(net)
imc <- cluster_infomap(net)
print(sprintf("Modularity, Fast Greedy, Node ID: %d: %f",node_list[i],␣
 ↪modularity(fc)))
print(sprintf("Modularity, Edge-Betweenness, Node ID: %d: %f",node_list[i],␣
 ↪modularity(eb)))
print(sprintf("Modularity, Infomap, Node ID: %d: %f",node_list[i],␣
 ↪modularity(imc)))
```

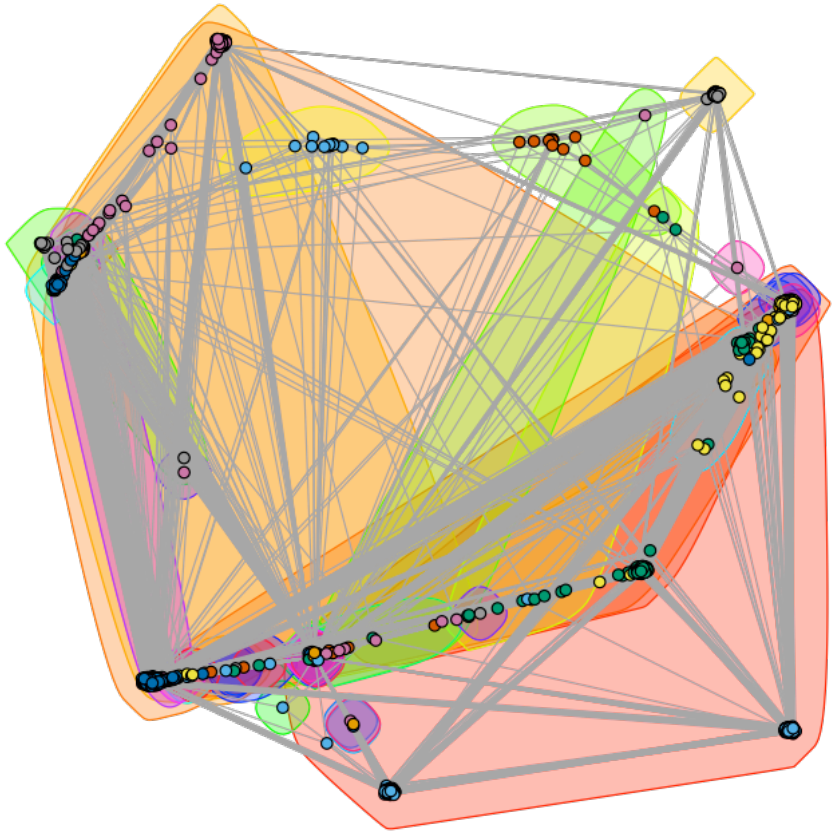
```
plot(eg[[i]],mark.groups = fc,vertex.size=3,vertex.color=fc$membership,vertex.
 ↪label="",main = sprintf("Community Structure, Fast Greedy, Node ID: %d",␣
 ↪node_list[i]))
dev.copy2pdf(file=sprintf('Q9fc%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = eb,vertex.size=3,vertex.color=eb$membership,vertex.
 ↪label="",main = sprintf("Community Structure, Edge-Betweenness, Node ID:␣
 ↪%d", node_list[i]))
dev.copy2pdf(file=sprintf('Q9eb%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = imc,vertex.size=3,vertex.color=imc$membership,vertex.
 ↪label="",main = sprintf("Community Structure, Infomap, Node ID: %d",␣
 ↪node_list[i]))
dev.copy2pdf(file=sprintf('Q9imc%d.pdf',node_list[i]))
```

```
[1] "Modularity, Fast Greedy, Node ID: 108: 0.435929"
[1] "Modularity, Edge-Betweenness, Node ID: 108: 0.506755"
[1] "Modularity, Infomap, Node ID: 108: 0.508223"
```

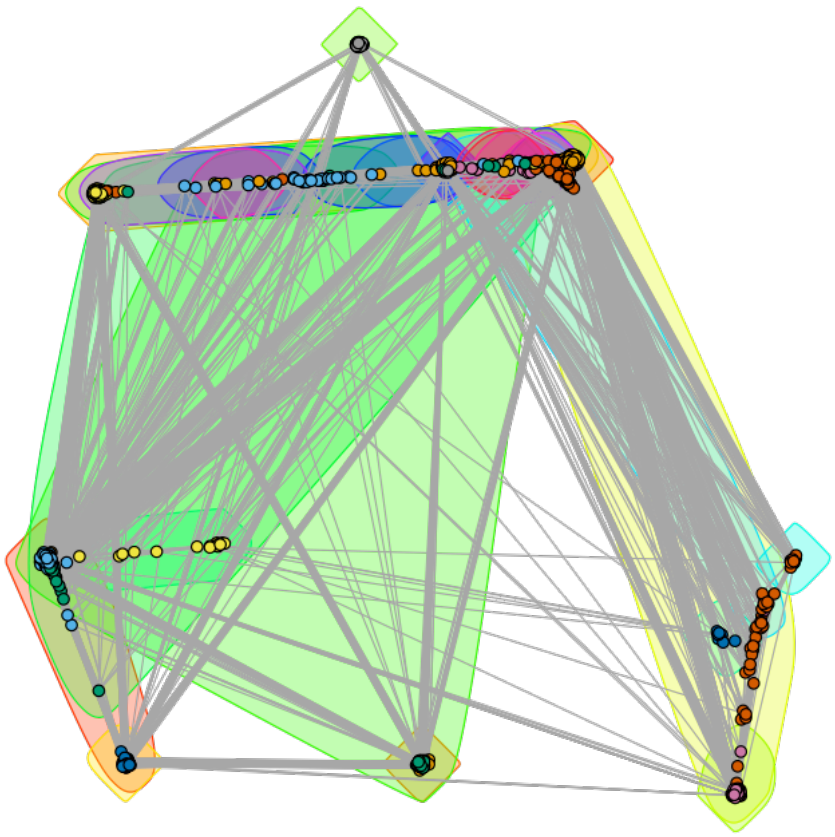
pdf: 2



pdf: 2



pdf: 2



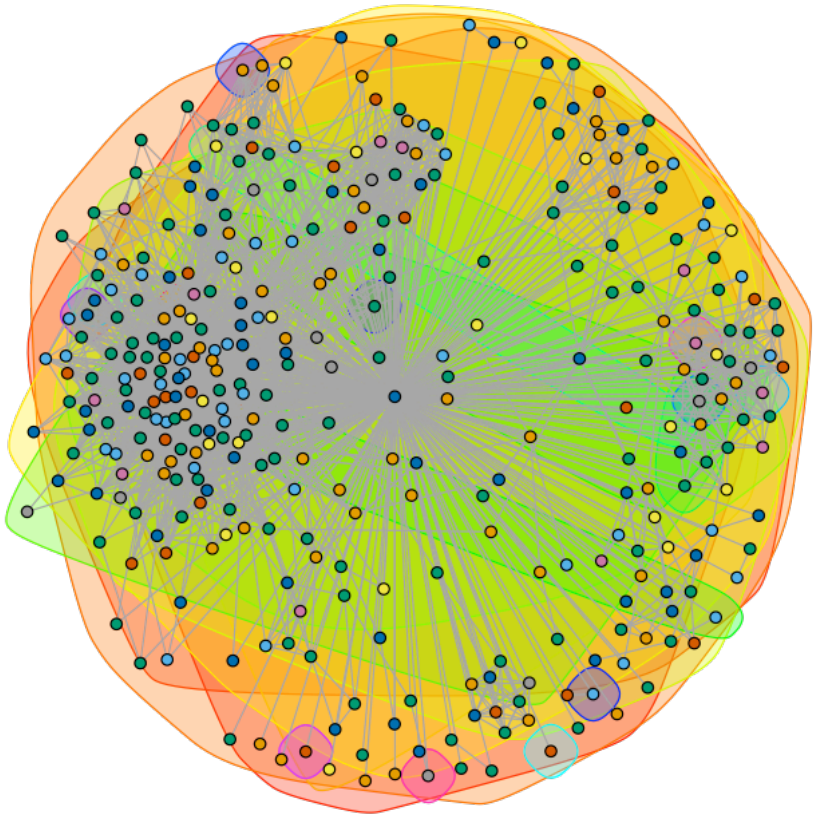
1.0.10 Question 10

[19]:
```
i = 1
net <- induced_subgraph(g, neighbors(g, node_list[i]))
fc <- cluster_fast_greedy(net)
eb <- cluster_edge_betweenness(net)
imc <- cluster_infomap(net)
print(sprintf("Modularity, Fast Greedy, Node ID (without core node): %d:␣
 ↪%f",node_list[i], modularity(fc)))
print(sprintf("Modularity, Edge-Betweenness, Node ID (without core node): %d:␣
 ↪%f",node_list[i], modularity(eb)))
```

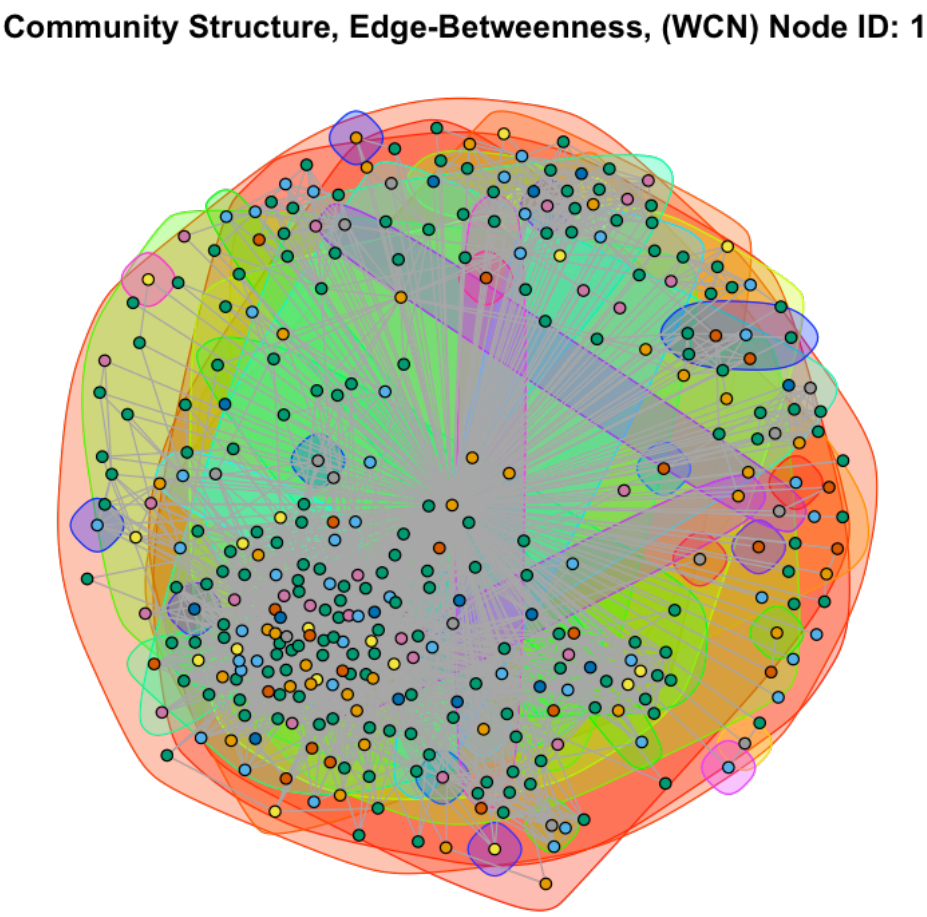
```
print(sprintf("Modularity, Infomap, Node ID (without core node): %d:␣
 ↪%f",node_list[i], modularity(imc)))
plot(eg[[i]],mark.groups = fc,vertex.size=3,edge.arrow.size=.5,vertex.
 ↪color=fc$membership,vertex.label="",main = sprintf("Community Structure,␣
 ↪Fast Greedy, (WCN) Node ID: %d", node_list[i]))
dev.copy2pdf(file=sprintf('Q10fc%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = eb,vertex.size=3,edge.arrow.size=.5,vertex.
 ↪color=eb$membership ,vertex.label="",main = sprintf("Community Structure,␣
 ↪Edge-Betweenness, (WCN) Node ID: %d", node_list[i]))
dev.copy2pdf(file=sprintf('Q10eb%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = imc,vertex.size=3,edge.arrow.size=.5,vertex.
 ↪color=imc$membership,vertex.label="",main = sprintf("Community Structure,␣
 ↪Infomap, (WCN) Node ID: %d", node_list[i]))
dev.copy2pdf(file=sprintf('Q10imc%d.pdf',node_list[i]))
```

```
[1] "Modularity, Fast Greedy, Node ID (without core node): 1: 0.441853"
[1] "Modularity, Edge-Betweenness, Node ID (without core node): 1: 0.416146"
[1] "Modularity, Infomap, Node ID (without core node): 1: 0.418008"
```

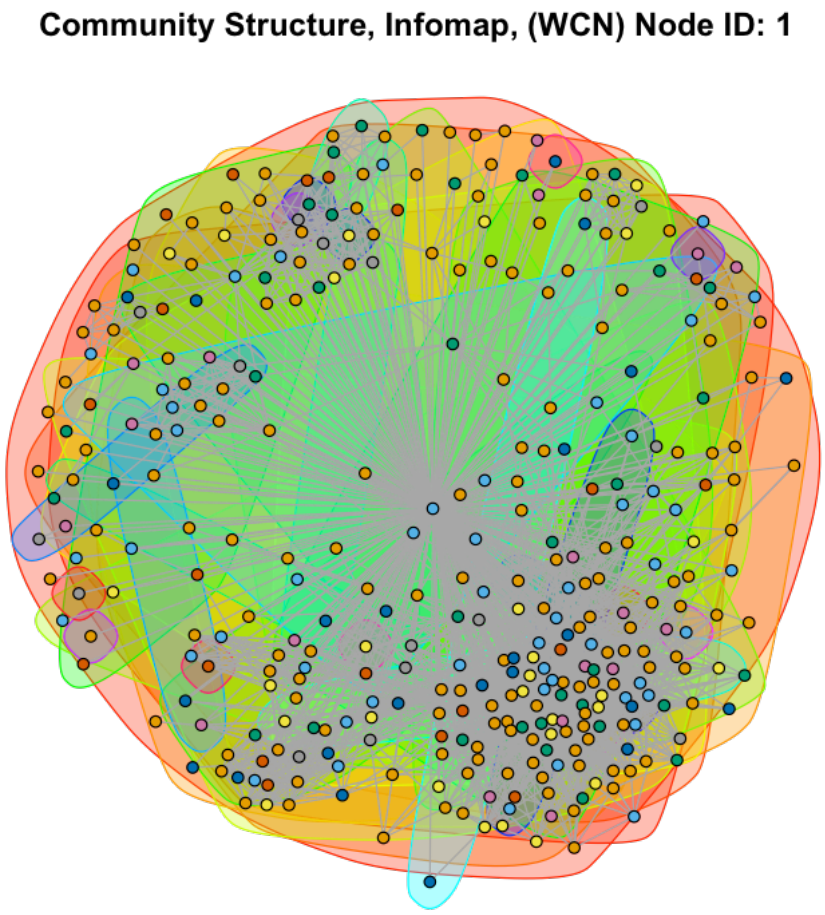
pdf: 2



pdf: 2



pdf: 2

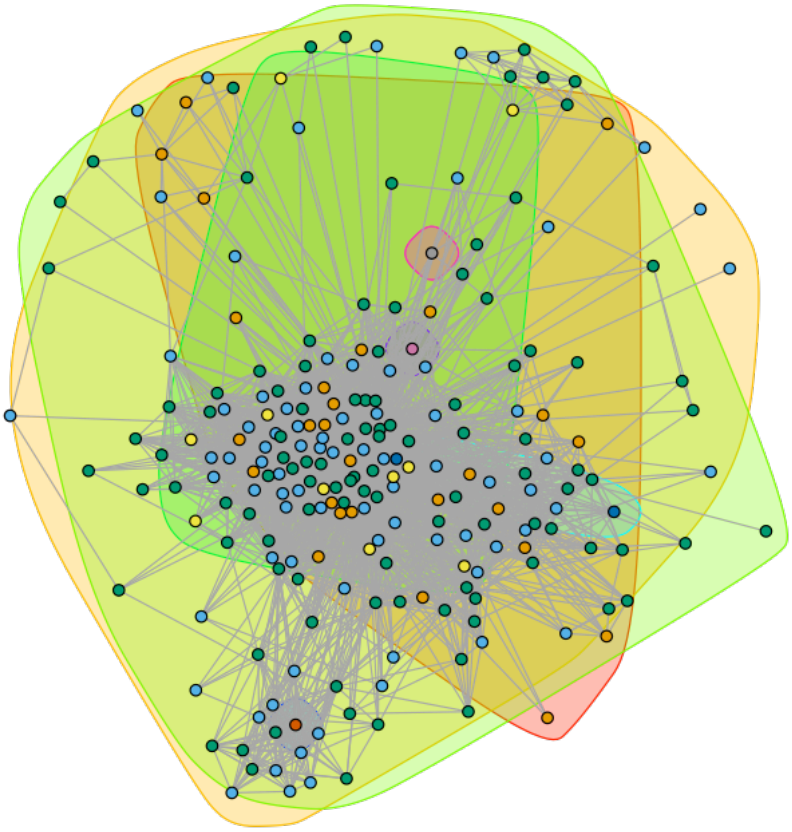


[20]:
```
i = 2
net <- induced_subgraph(g, neighbors(g, node_list[i]))
fc <- cluster_fast_greedy(net)
eb <- cluster_edge_betweenness(net)
imc <- cluster_infomap(net)
print(sprintf("Modularity, Fast Greedy, Node ID (without core node): %d:␣
 ↪%f",node_list[i], modularity(fc)))
print(sprintf("Modularity, Edge-Betweenness, Node ID (without core node): %d:␣
 ↪%f",node_list[i], modularity(eb)))
print(sprintf("Modularity, Infomap, Node ID (without core node): %d:␣
 ↪%f",node_list[i], modularity(imc)))
```

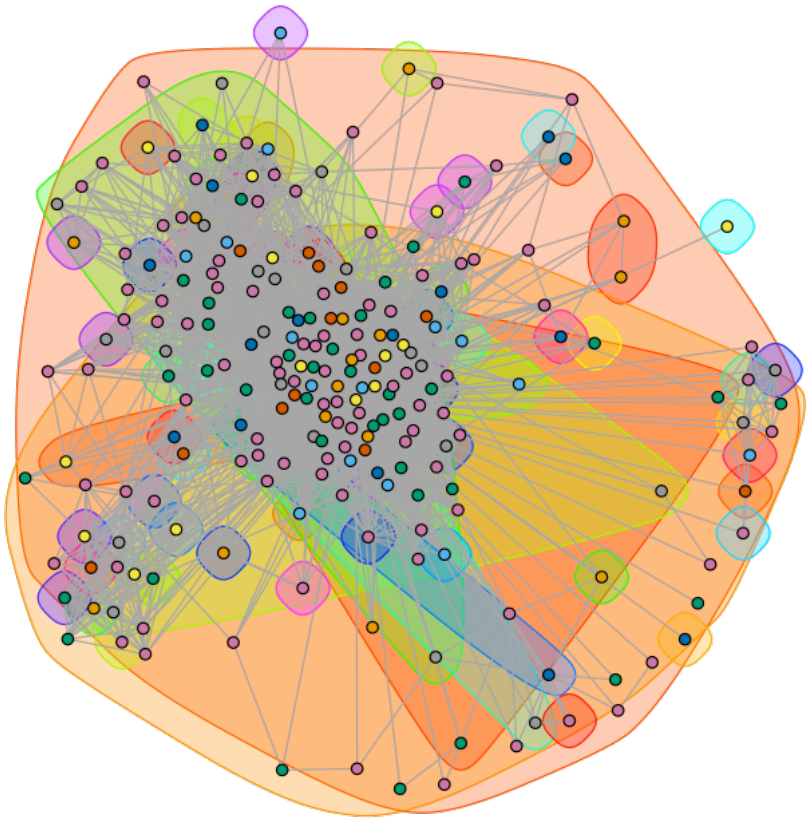
```
plot(eg[[i]],mark.groups = fc,vertex.size=3,edge.arrow.size=.5,vertex.
 ↪color=fc$membership,vertex.label="",main = sprintf("Community Structure,␣
 ↪Fast Greedy, (WCN) Node ID: %d", node_list[i]))
dev.copy2pdf(file=sprintf('Q10fc%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = eb,vertex.size=3,edge.arrow.size=.5,vertex.
 ↪color=eb$membership ,vertex.label="",main = sprintf("Community Structure,␣
 ↪Edge-Betweenness, (WCN) Node ID: %d", node_list[i]))
dev.copy2pdf(file=sprintf('Q10eb%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = imc,vertex.size=3,edge.arrow.size=.5,vertex.
 ↪color=imc$membership,vertex.label="",main = sprintf("Community Structure,␣
 ↪Infomap, (WCN) Node ID: %d", node_list[i]))
dev.copy2pdf(file=sprintf('Q10imc%d.pdf',node_list[i]))
```

```
[1] "Modularity, Fast Greedy, Node ID (without core node): 349: 0.245692"
[1] "Modularity, Edge-Betweenness, Node ID (without core node): 349: 0.150566"
[1] "Modularity, Infomap, Node ID (without core node): 349: 0.244816"
```

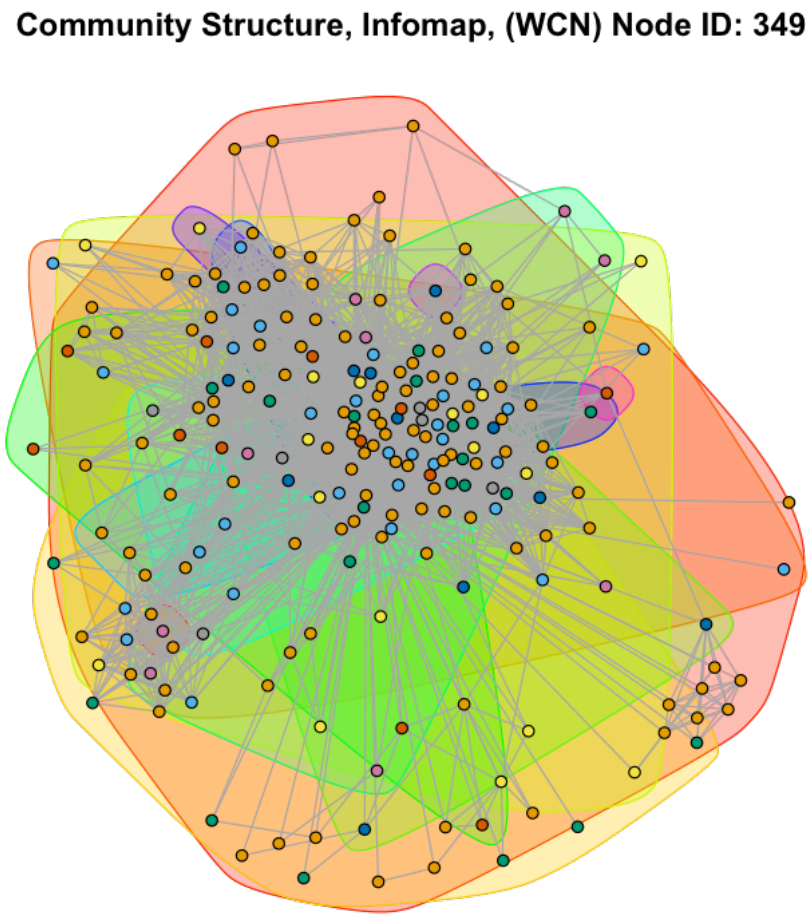
pdf: 2



pdf: 2



pdf: 2

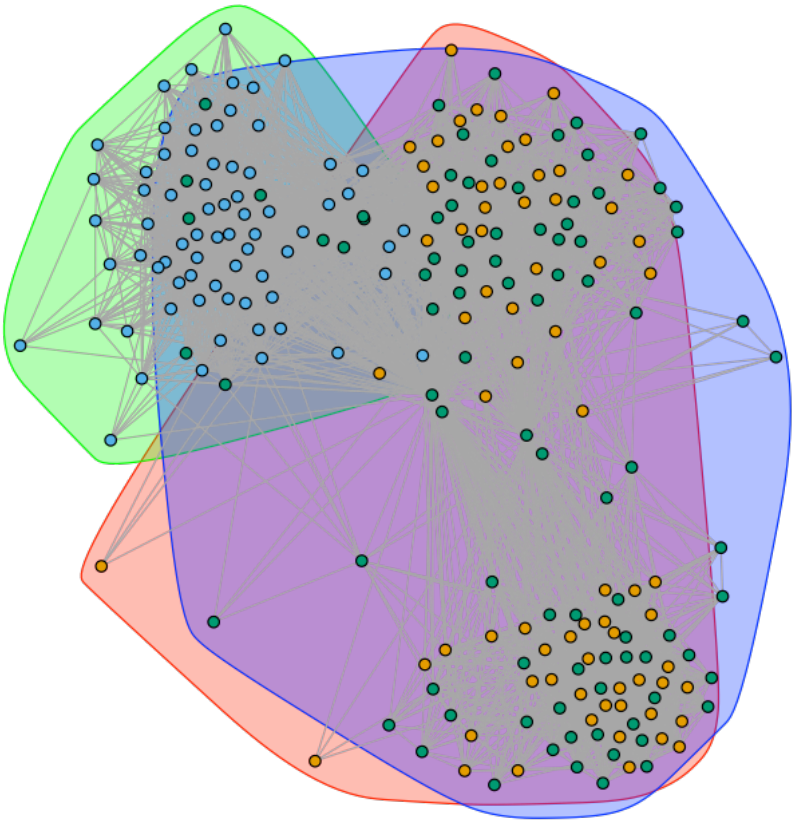


[21]:
```
i = 3
net <- induced_subgraph(g, neighbors(g, node_list[i]))
fc <- cluster_fast_greedy(net)
eb <- cluster_edge_betweenness(net)
imc <- cluster_infomap(net)
print(sprintf("Modularity, Fast Greedy, Node ID (without core node): %d:␣
 ↪%f",node_list[i], modularity(fc)))
print(sprintf("Modularity, Edge-Betweenness, Node ID (without core node): %d:␣
 ↪%f",node_list[i], modularity(eb)))
print(sprintf("Modularity, Infomap, Node ID (without core node): %d:␣
 ↪%f",node_list[i], modularity(imc)))
```

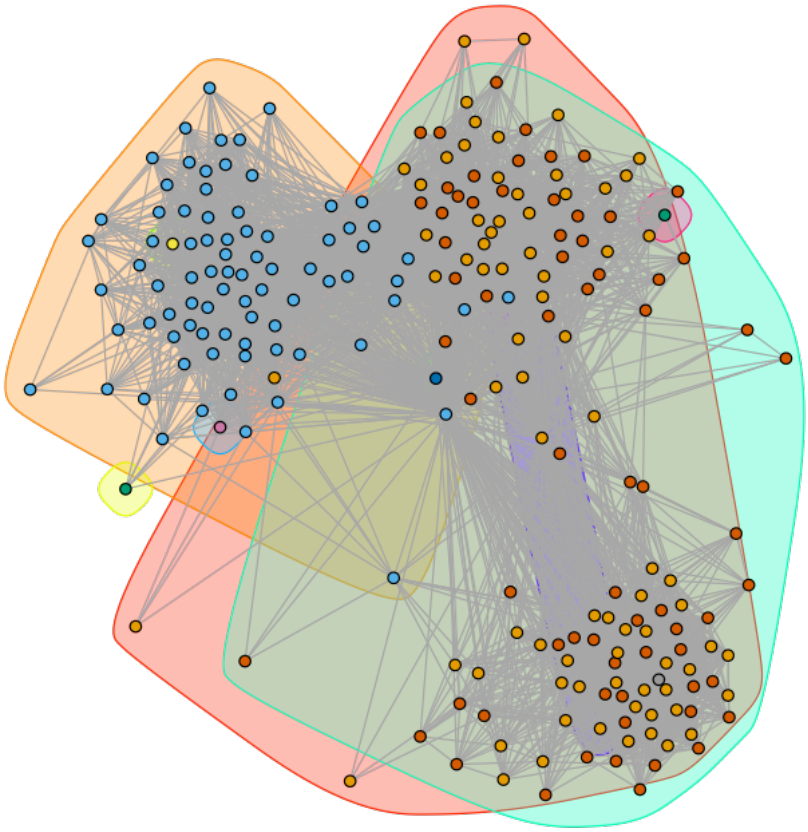
```
plot(eg[[i]],mark.groups = fc,vertex.size=3,edge.arrow.size=.5,vertex.
 ↪color=fc$membership,vertex.label="",main = sprintf("Community Structure,␣
 ↪Fast Greedy, (WCN) Node ID: %d", node_list[i]))
dev.copy2pdf(file=sprintf('Q10fc%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = eb,vertex.size=3,edge.arrow.size=.5,vertex.
 ↪color=eb$membership ,vertex.label="",main = sprintf("Community Structure,␣
 ↪Edge-Betweenness, (WCN) Node ID: %d", node_list[i]))
dev.copy2pdf(file=sprintf('Q10eb%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = imc,vertex.size=3,edge.arrow.size=.5,vertex.
 ↪color=imc$membership,vertex.label="",main = sprintf("Community Structure,␣
 ↪Infomap, (WCN) Node ID: %d", node_list[i]))
dev.copy2pdf(file=sprintf('Q10imc%d.pdf',node_list[i]))
```

```
[1] "Modularity, Fast Greedy, Node ID (without core node): 484: 0.534214"
[1] "Modularity, Edge-Betweenness, Node ID (without core node): 484: 0.515441"
[1] "Modularity, Infomap, Node ID (without core node): 484: 0.543444"
```

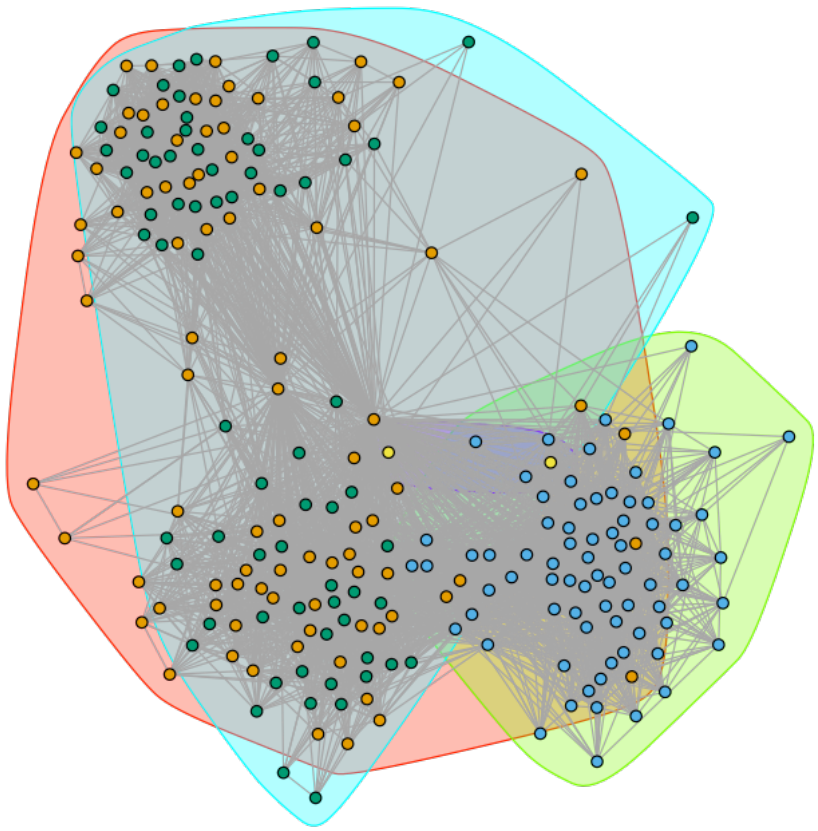
pdf: 2



pdf: 2



pdf: 2

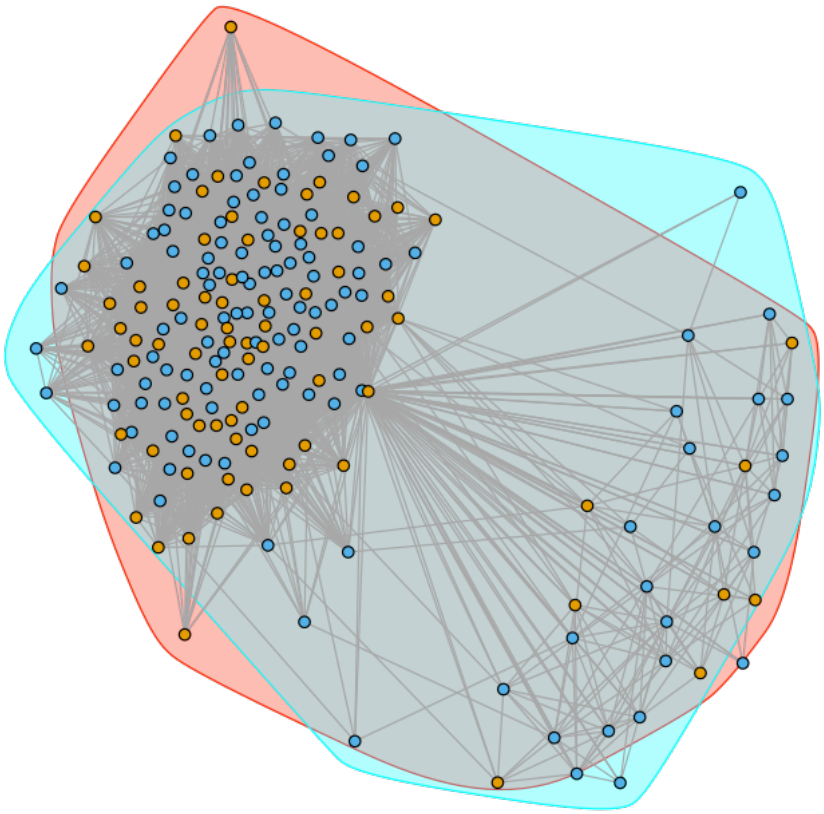


[22]:
```
i = 4
net <- induced_subgraph(g, neighbors(g, node_list[i]))
fc <- cluster_fast_greedy(net)
eb <- cluster_edge_betweenness(net)
imc <- cluster_infomap(net)
print(sprintf("Modularity, Fast Greedy, Node ID (without core node): %d:␣
 ↪%f",node_list[i], modularity(fc)))
print(sprintf("Modularity, Edge-Betweenness, Node ID (without core node): %d:␣
 ↪%f",node_list[i], modularity(eb)))
print(sprintf("Modularity, Infomap, Node ID (without core node): %d:␣
 ↪%f",node_list[i], modularity(imc)))
```

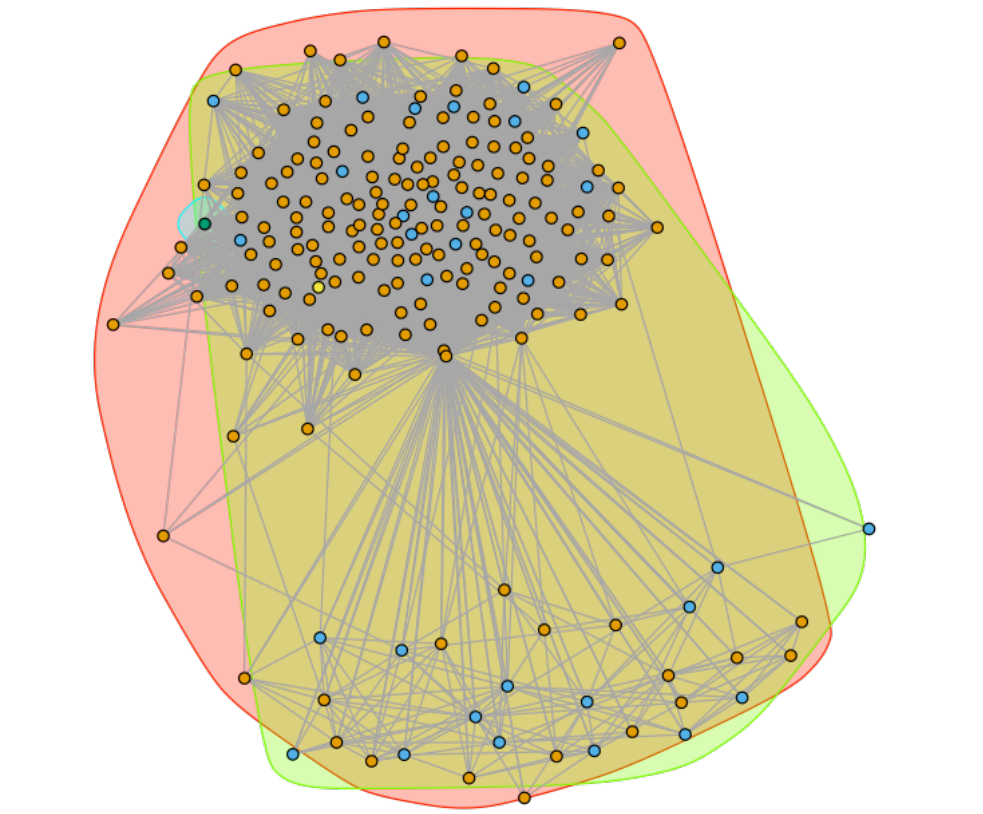
```
plot(eg[[i]],mark.groups = fc,vertex.size=3,edge.arrow.size=.5,vertex.
 ↪color=fc$membership,vertex.label="",main = sprintf("Community Structure,␣
 ↪Fast Greedy, (WCN) Node ID: %d", node_list[i]))
dev.copy2pdf(file=sprintf('Q10fc%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = eb,vertex.size=3,edge.arrow.size=.5,vertex.
 ↪color=eb$membership ,vertex.label="",main = sprintf("Community Structure,␣
 ↪Edge-Betweenness, (WCN) Node ID: %d", node_list[i]))
dev.copy2pdf(file=sprintf('Q10eb%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = imc,vertex.size=3,edge.arrow.size=.5,vertex.
 ↪color=imc$membership,vertex.label="",main = sprintf("Community Structure,␣
 ↪Infomap, (WCN) Node ID: %d", node_list[i]))
dev.copy2pdf(file=sprintf('Q10imc%d.pdf',node_list[i]))
```

```
[1] "Modularity, Fast Greedy, Node ID (without core node): 1087: 0.148196"
[1] "Modularity, Edge-Betweenness, Node ID (without core node): 1087: 0.032495"
[1] "Modularity, Infomap, Node ID (without core node): 1087: 0.027372"
```

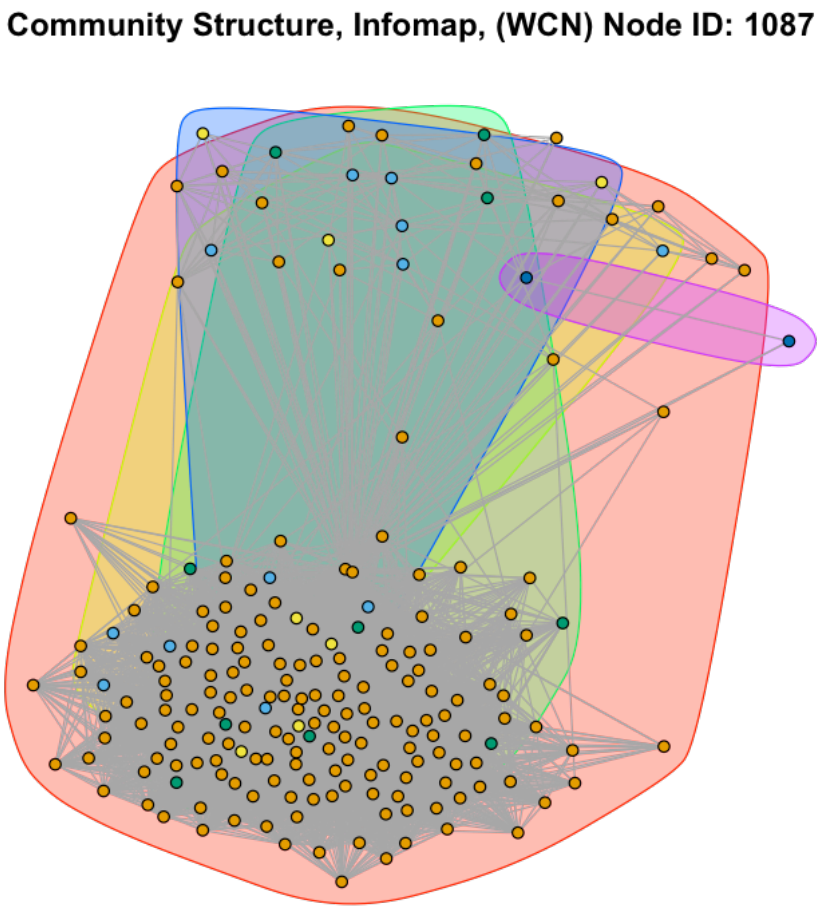
pdf: 2



pdf: 2



pdf: 2

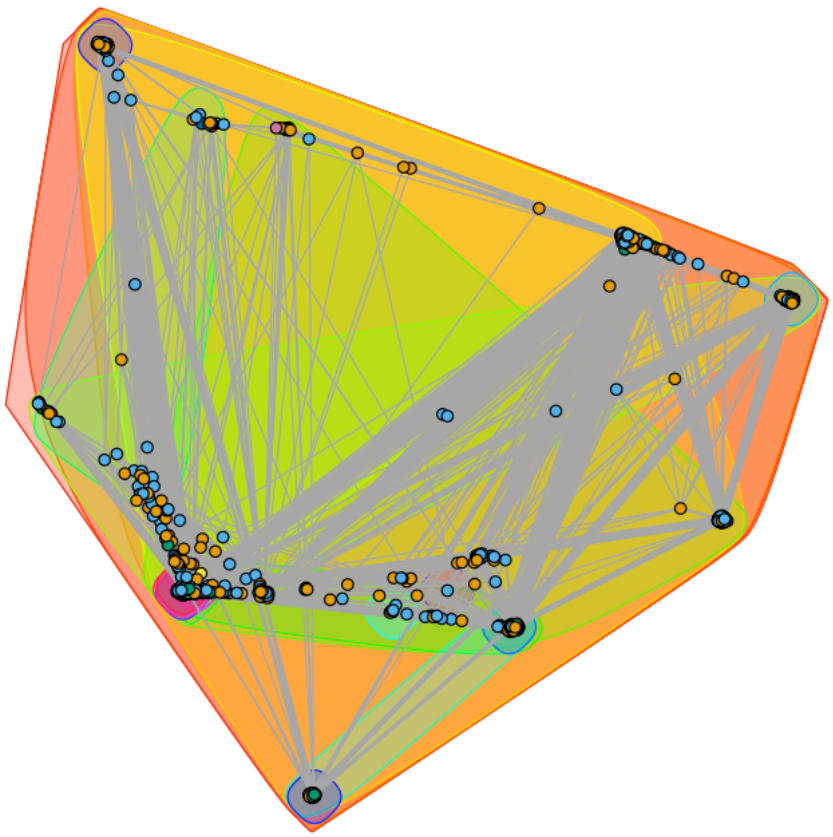


[30]:
```
i = 5
net <- induced_subgraph(g, neighbors(g, node_list[i]))
fc <- cluster_fast_greedy(net)
eb <- cluster_edge_betweenness(net)
imc <- cluster_infomap(net)
print(sprintf("Modularity, Fast Greedy, Node ID (without core node): %d:␣
 ↪%f",node_list[i], modularity(fc)))
print(sprintf("Modularity, Edge-Betweenness, Node ID (without core node): %d:␣
 ↪%f",node_list[i], modularity(eb)))
print(sprintf("Modularity, Infomap, Node ID (without core node): %d:␣
 ↪%f",node_list[i], modularity(imc)))
```

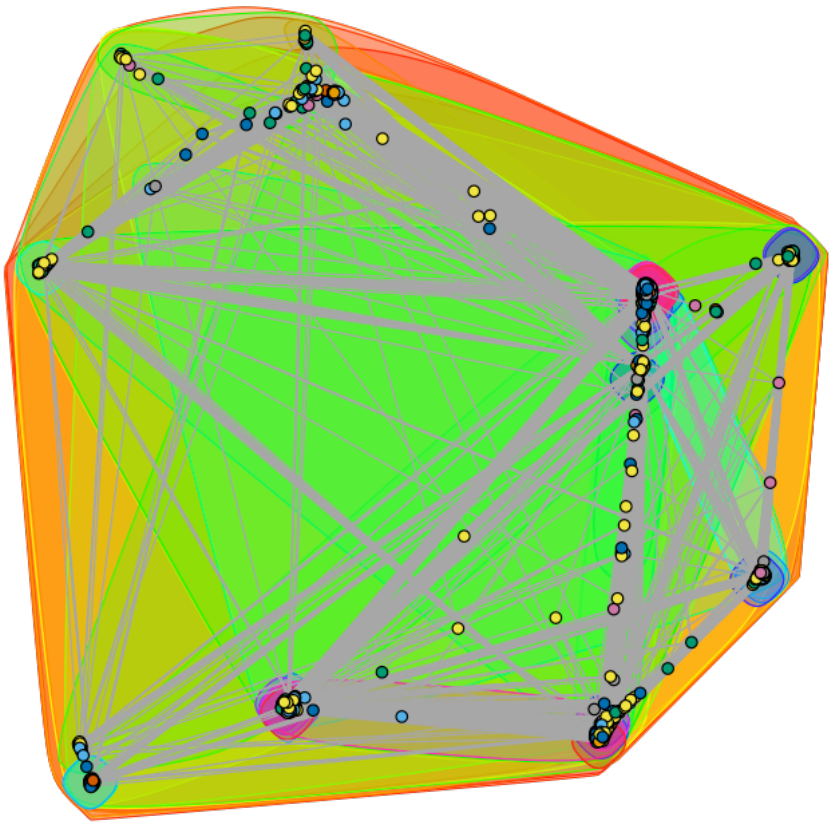
```
plot(eg[[i]],mark.groups = fc,vertex.size=3,edge.arrow.size=.5,vertex.
 ↪color=fc$membership,vertex.label="",main = sprintf("Community Structure,␣
 ↪Fast Greedy, (WCN) Node ID: %d", node_list[i]))
dev.copy2pdf(file=sprintf('Q10fc%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = eb,vertex.size=3,edge.arrow.size=.5,vertex.
 ↪color=eb$membership ,vertex.label="",main = sprintf("Community Structure,␣
 ↪Edge-Betweenness, (WCN) Node ID: %d", node_list[i]))
dev.copy2pdf(file=sprintf('Q10eb%d.pdf',node_list[i]))
plot(eg[[i]],mark.groups = imc,vertex.size=3,edge.arrow.size=.5,vertex.
 ↪color=imc$membership,vertex.label="",main = sprintf("Community Structure,␣
 ↪Infomap, (WCN) Node ID: %d", node_list[i]))
dev.copy2pdf(file=sprintf('Q10imc%d.pdf',node_list[i]))
```

```
[1] "Modularity, Fast Greedy, Node ID (without core node): 108: 0.458127"
[1] "Modularity, Edge-Betweenness, Node ID (without core node): 108: 0.521322"
[1] "Modularity, Infomap, Node ID (without core node): 108: 0.520989"
```

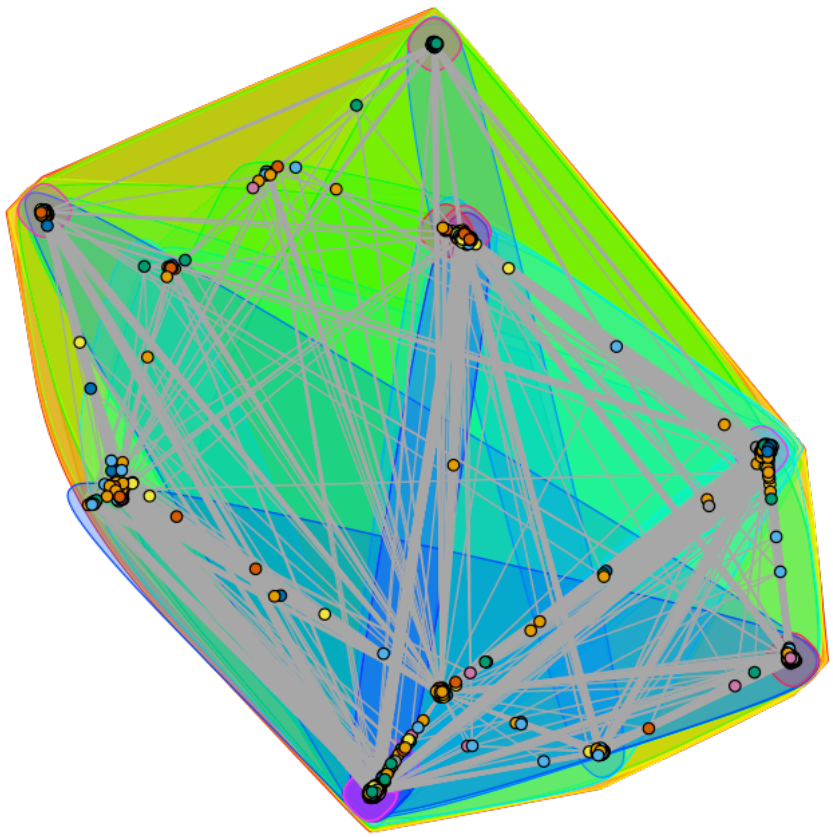
pdf: 2



pdf: 2



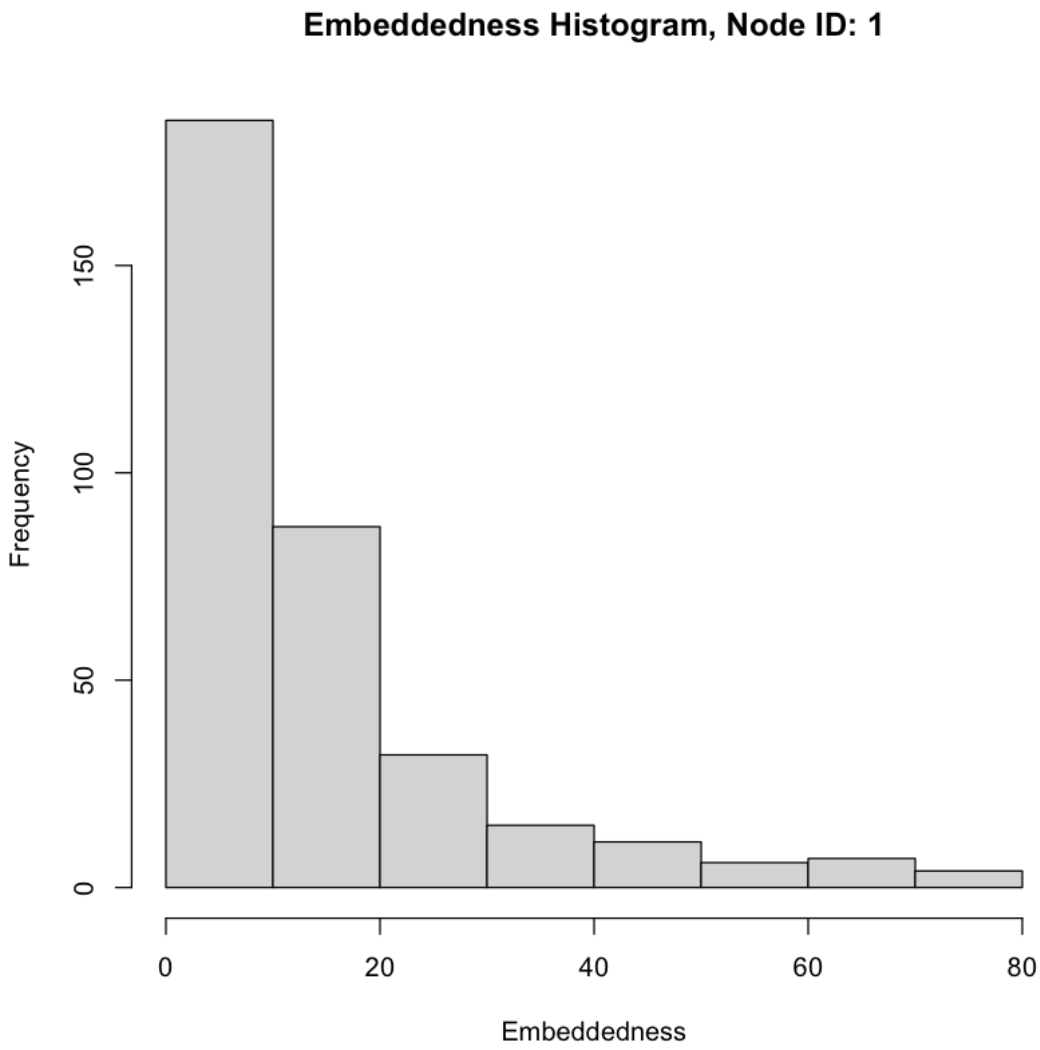
pdf: 2

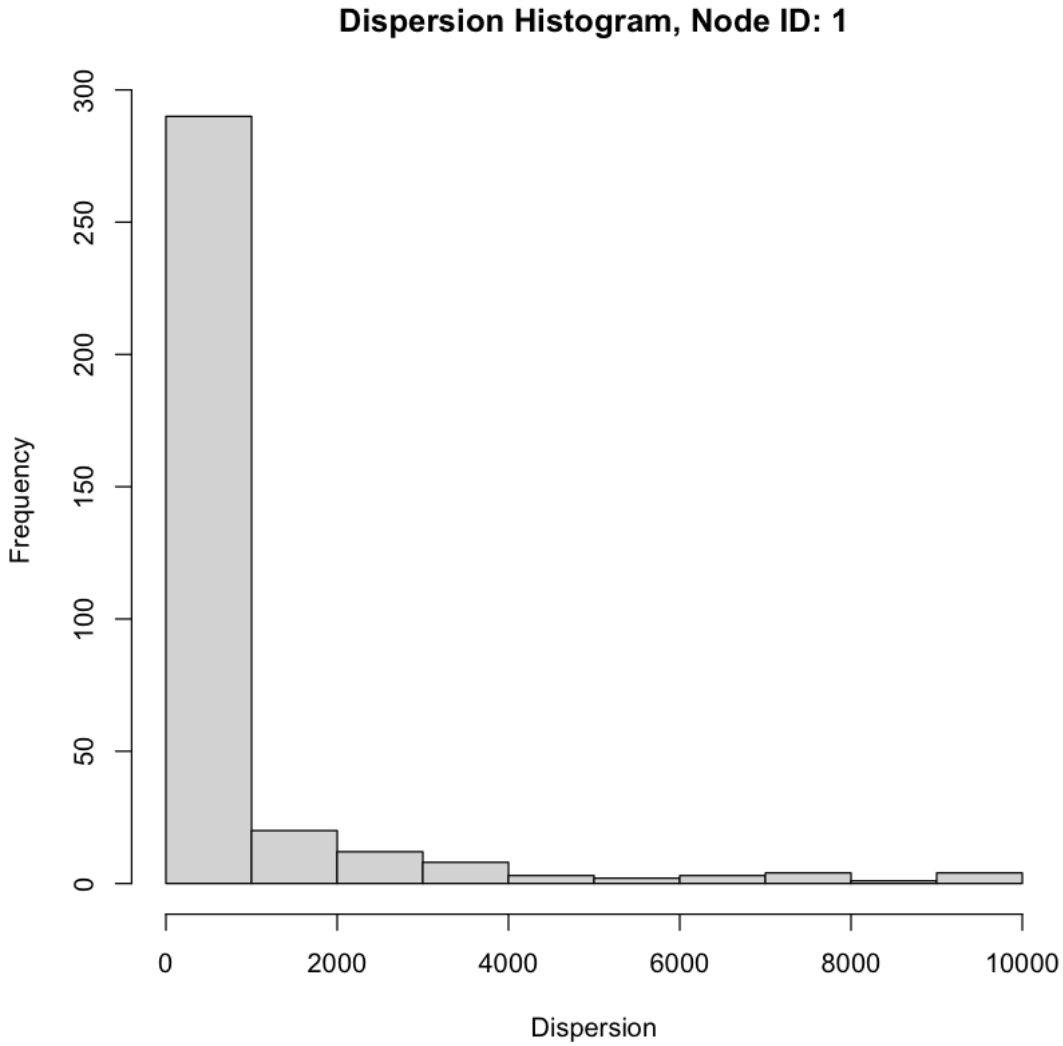


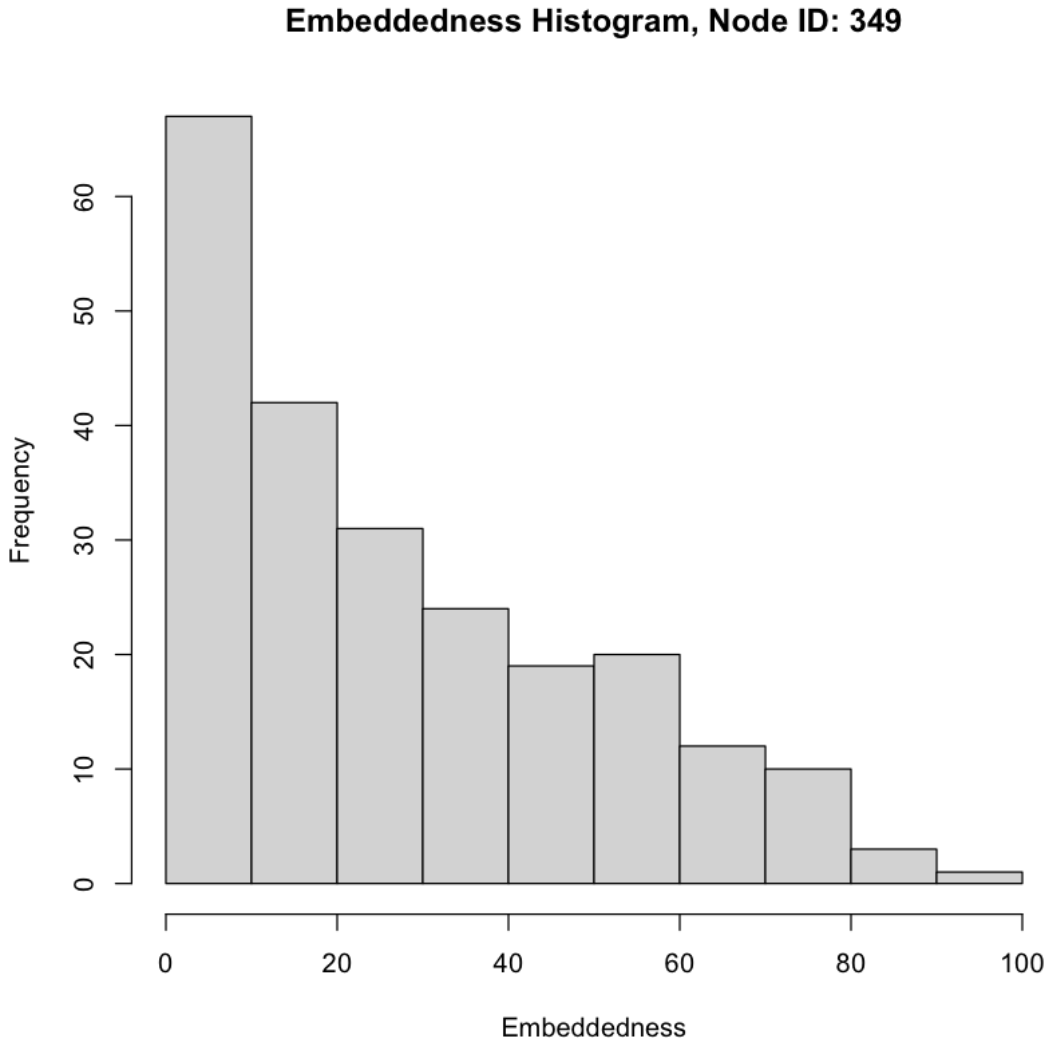
1.0.11 Question 12

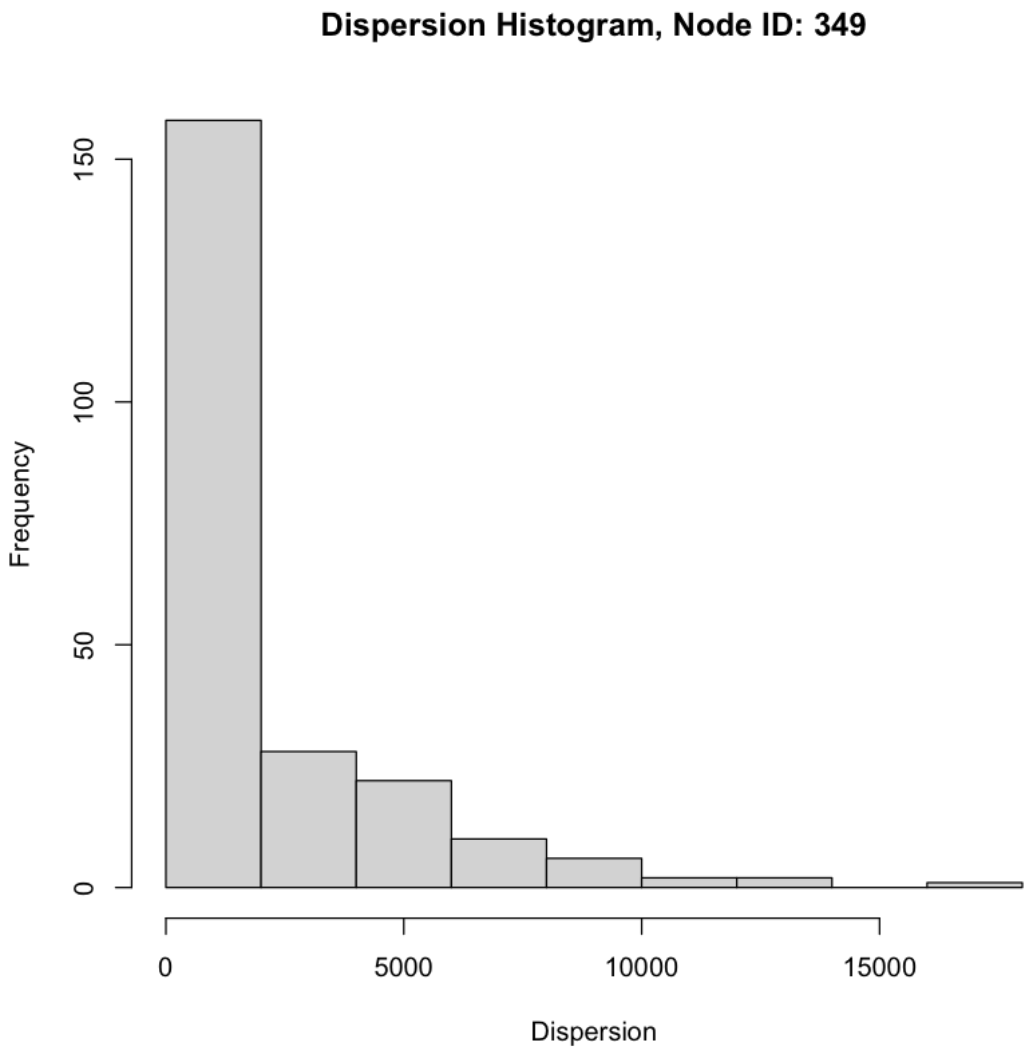
[31]:
```
el <- read.table("facebook_combined.txt",header=FALSE)
fb <- graph.data.frame(el, directed=FALSE)
node_list_str = c("0","348","483","1086","107")
for(j in c(1:5)){

    nodelist = unlist(ego(fb, order=1, nodes=node_list_str[j]))
    pn = induced.subgraph(fb,nodelist)
    pn$name = sort(nodelist)
    embeddedness <- c()
    disp <- c()
    deg <- c()
```

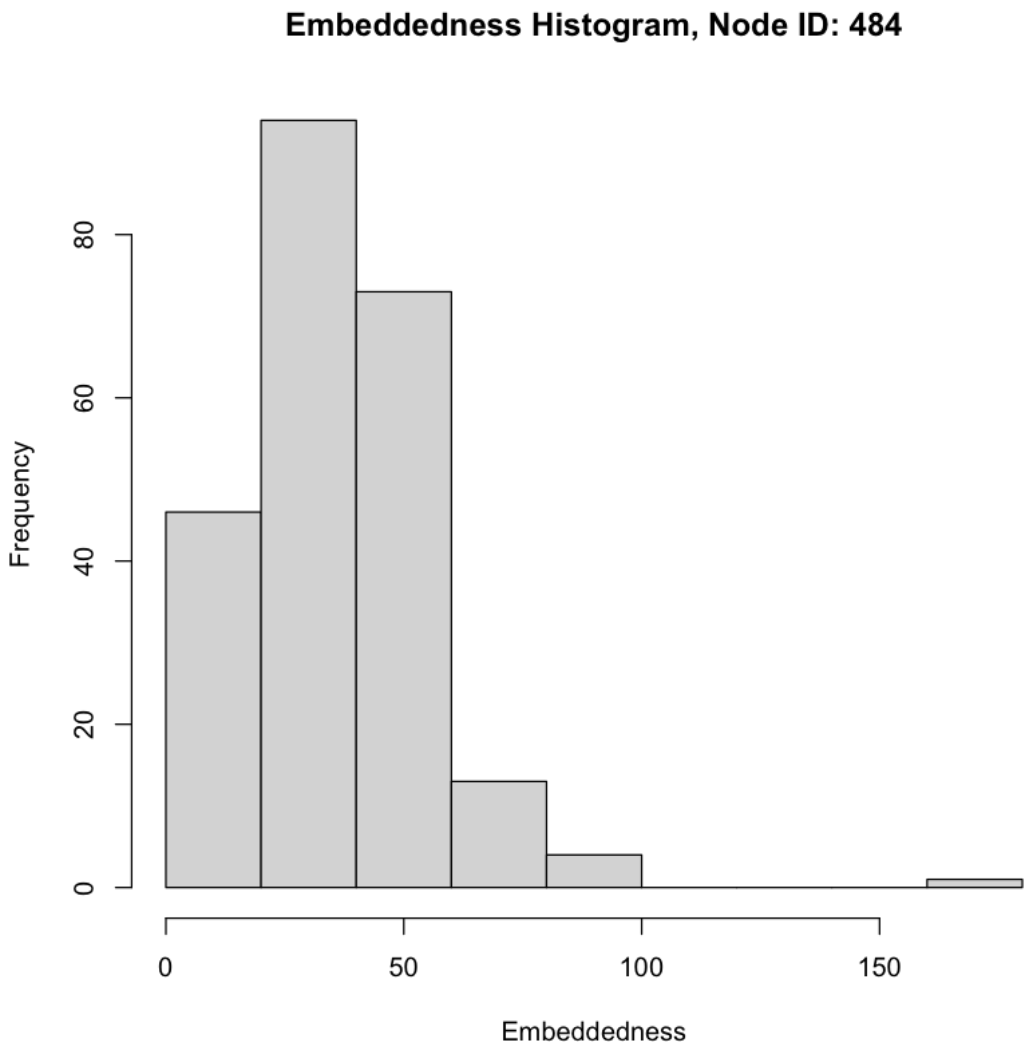
```
    i=1
    for(v in vertex_attr(pn)$name){
        if(v==node_list_str[j])
            next

        disp[i] = 0
        neh_ver = neighbors(pn, node_list_str[j])
        neh_core = neighbors(pn,v)
        inter = intersection(neh_ver, neh_core)
        embeddedness[i] = length(inter)

        deg[i] = degree(pn,v)
        eg2 = delete.vertices(pn, c(node_list_str[j], v))
        if(embeddedness[i]>1){
            ver = c()
            for(m in 1:length(inter)){
                ver =  c(ver,vertex_attr(pn)$name[inter[m]])
            }
            ver1=c()
            for(m in 1:length(ver)){
                ver1=c(ver1,which(vertex_attr(eg2)$name==ver[m]))
            }
            disp_mat = distances(eg2,v=ver1, to=ver1)
            disp_mat[disp_mat==Inf]<-diameter(eg2)+1
            disp[i] = sum(disp_mat)
        }
        i=i+1

    }
    hist(embeddedness,main=sprintf("Embeddedness Histogram, Node ID:␣
↪%d",strtoi(node_list_str[j])+1), xlab="Embeddedness",ylab="Frequency")
    dev.copy2eps(file=sprintf('Q12e%d.eps',strtoi(node_list_str[j])+1))
    hist(disp,main=sprintf("Dispersion Histogram, Node ID:␣
↪%d",strtoi(node_list_str[j])+1), xlab="Dispersion",ylab="Frequency")
    dev.copy2eps(file=sprintf('Q12d%d.eps',strtoi(node_list_str[j])+1))
}
```

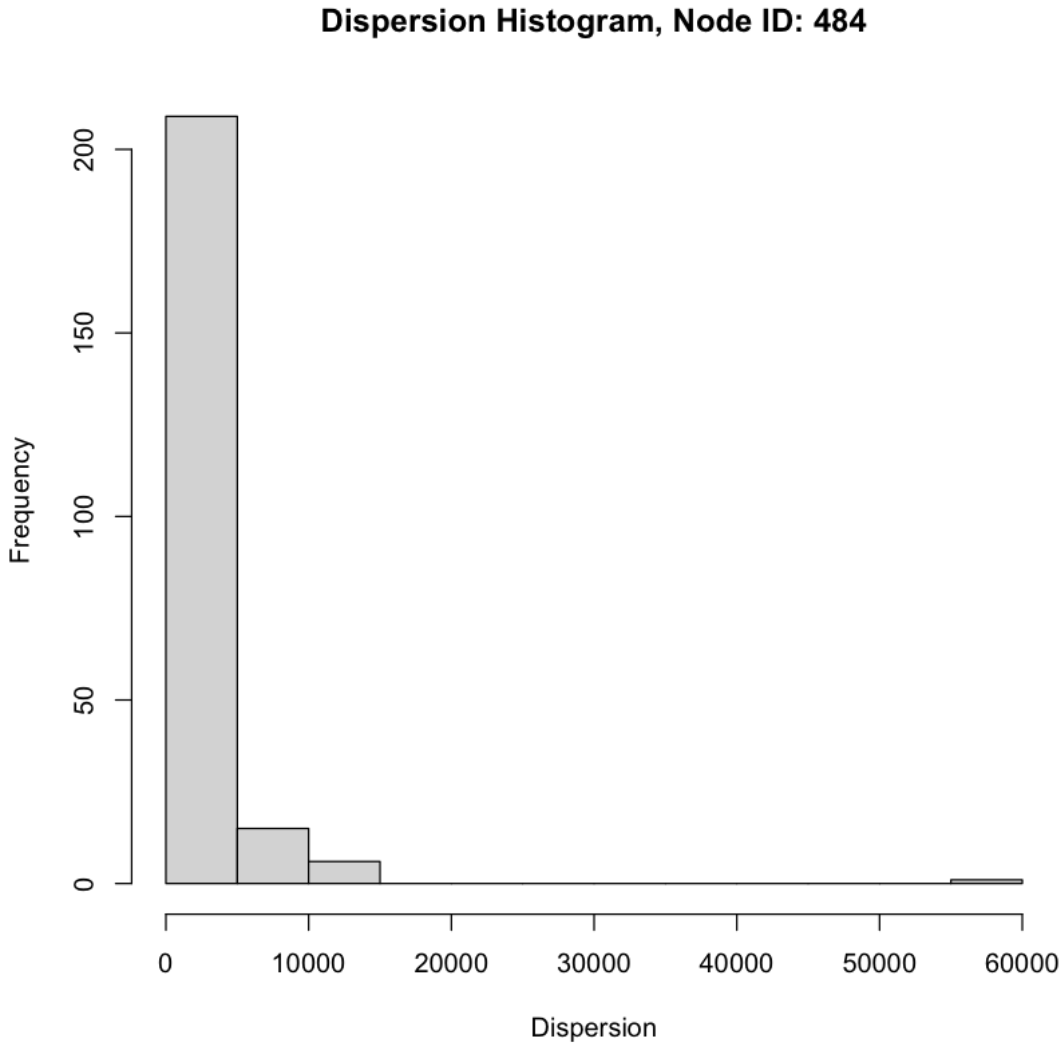

Embeddedness Histogram, Node ID: 1
Frequency
0
50
100
150
0
20
40
60
80
Embeddedness

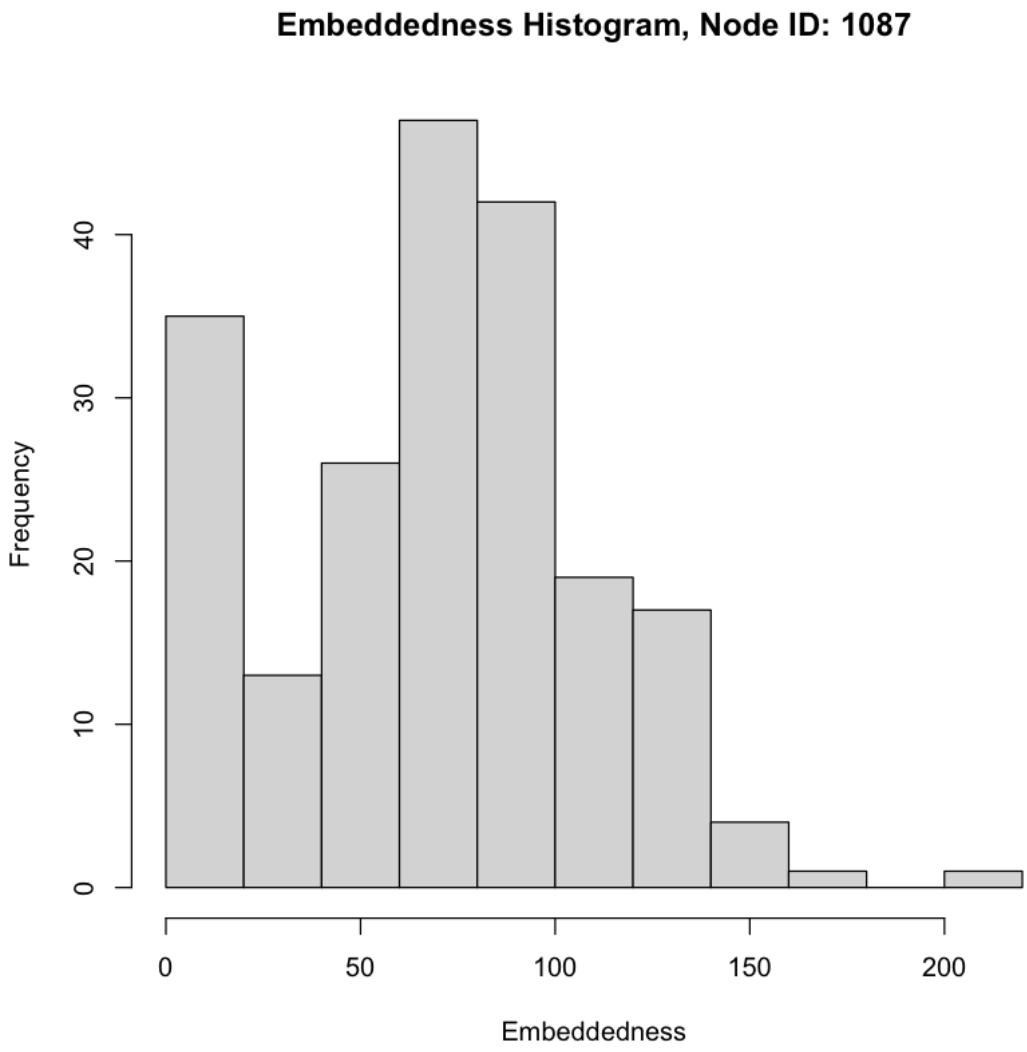

Dispersion Histogram, Node ID: 1
Frequency
0
50
100
150
200
250
300
0
2000
4000
6000
8000
10000
Dispersion

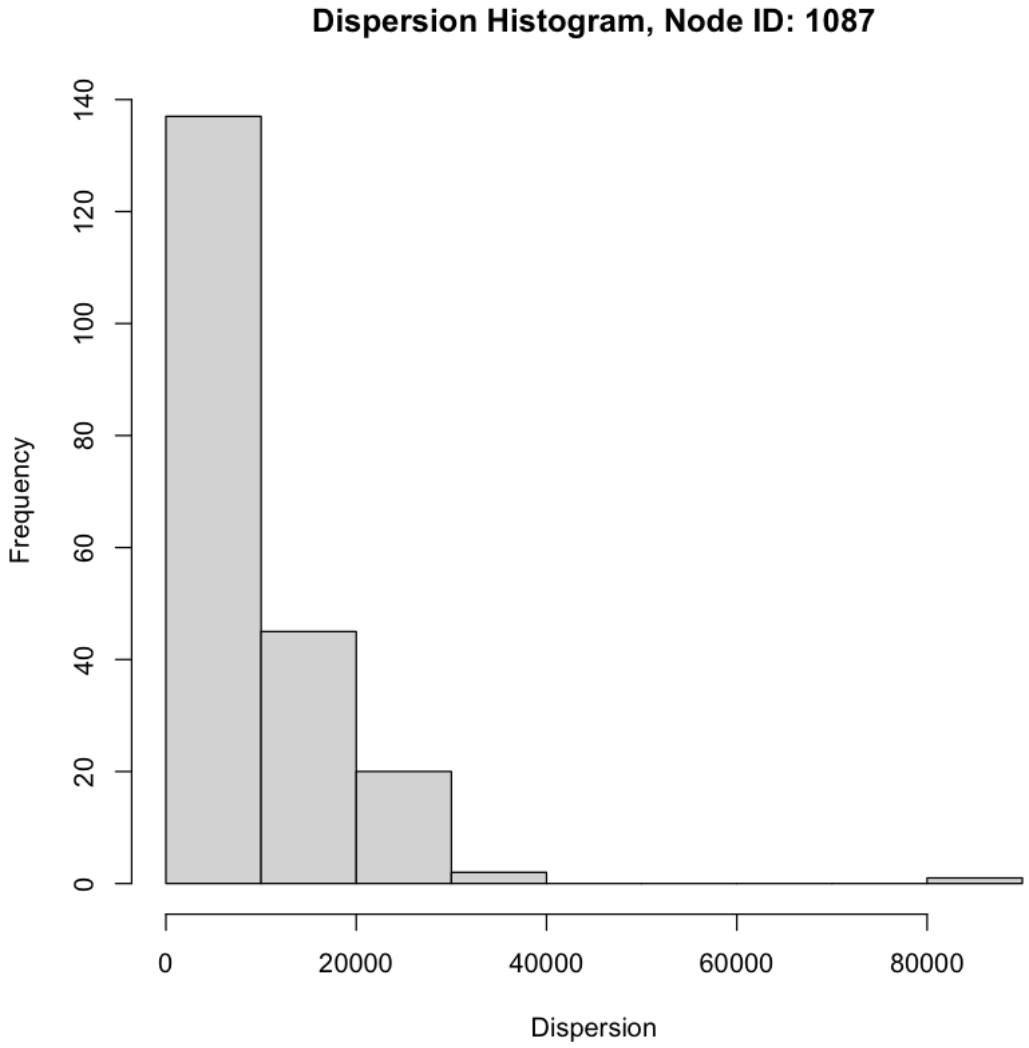

Embeddedness Histogram, Node ID: 349
Frequency
0
10
20
30
40
50
60
0
20
40
60
80
100
Embeddedness

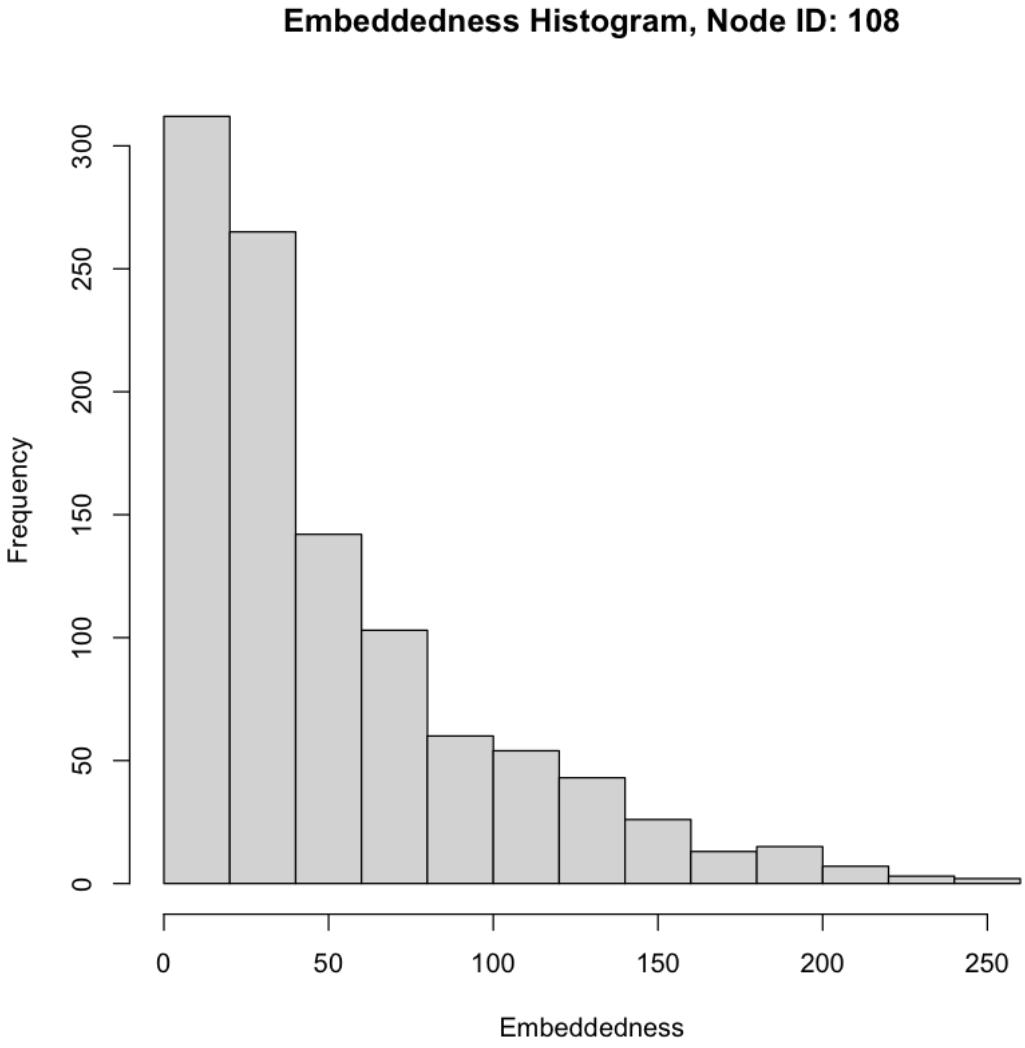

Dispersion Histogram, Node ID: 349
Frequency
0
50
100
150
0
5000
10000
15000
Dispersion

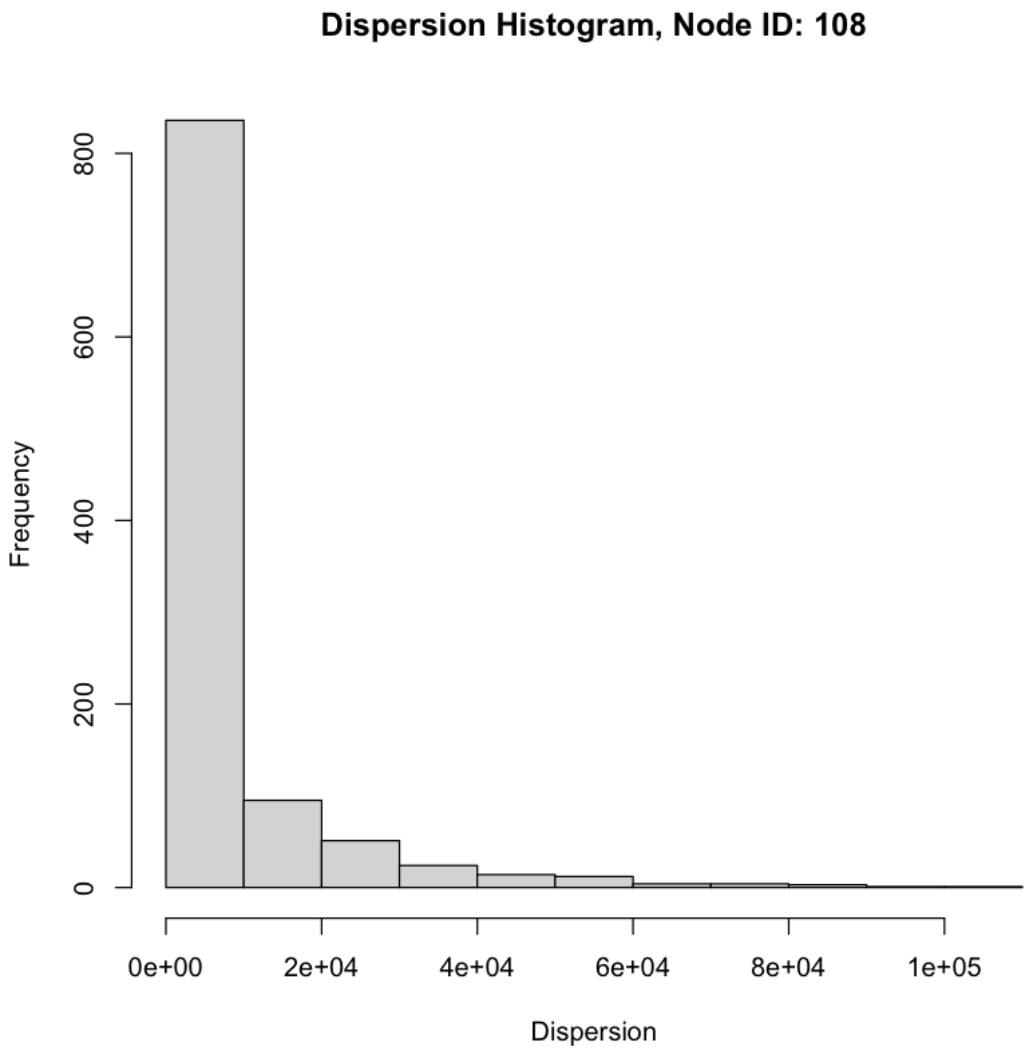

Embeddedness Histogram, Node ID: 484
Frequency
0
20
40
60
80
0
50
100
150
Embeddedness


Dispersion Histogram, Node ID: 484
Frequency
0
50
100
150
200
0
10000
20000
30000
40000
50000
60000
Dispersion


Embeddedness Histogram, Node ID: 1087
Frequency
0
10
20
30
40
0
50
100
150
200
Embeddedness


Dispersion Histogram, Node ID: 1087
Frequency
0
20
40
60
80
100
120
140
0
20000
40000
60000
80000
Dispersion


Embeddedness Histogram, Node ID: 108
Frequency
0
50
100
150
200
250
300
0
50
100
150
200
250
Embeddedness

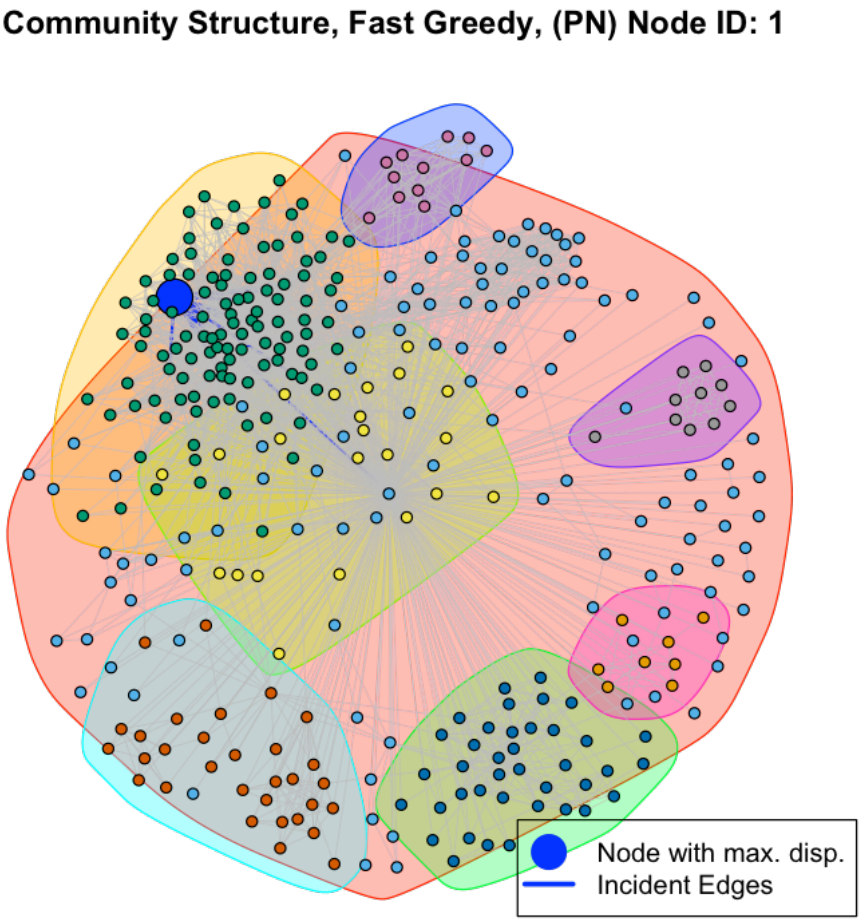


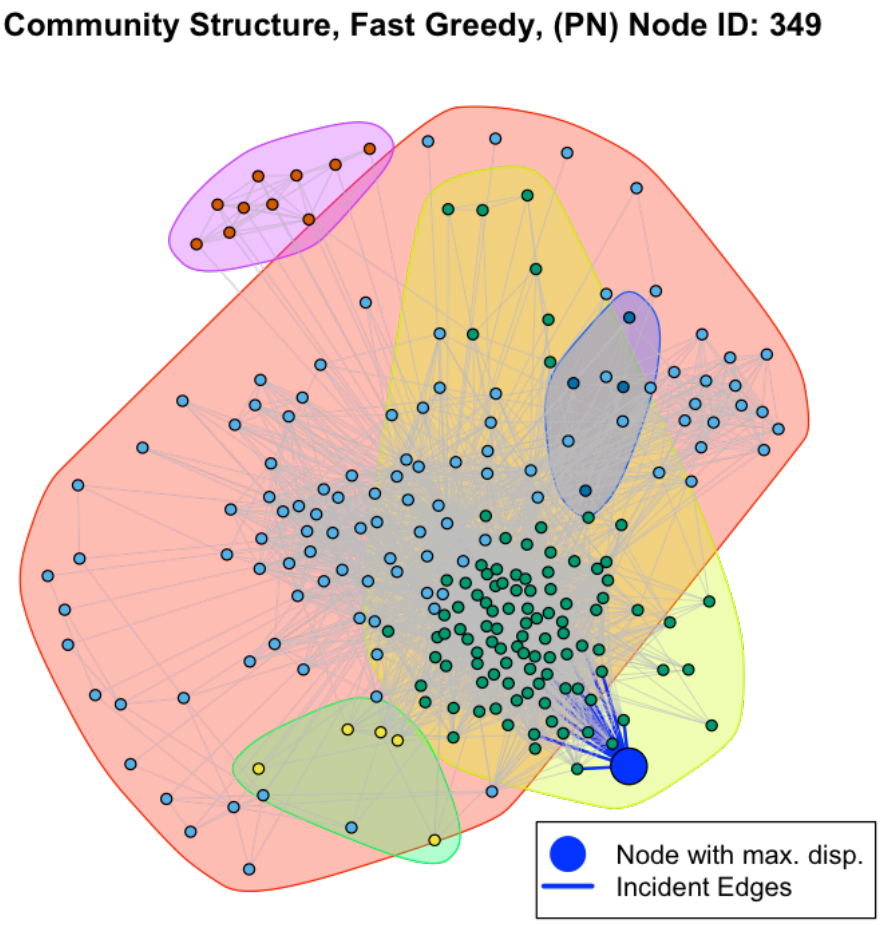
1.0.12 Question 13

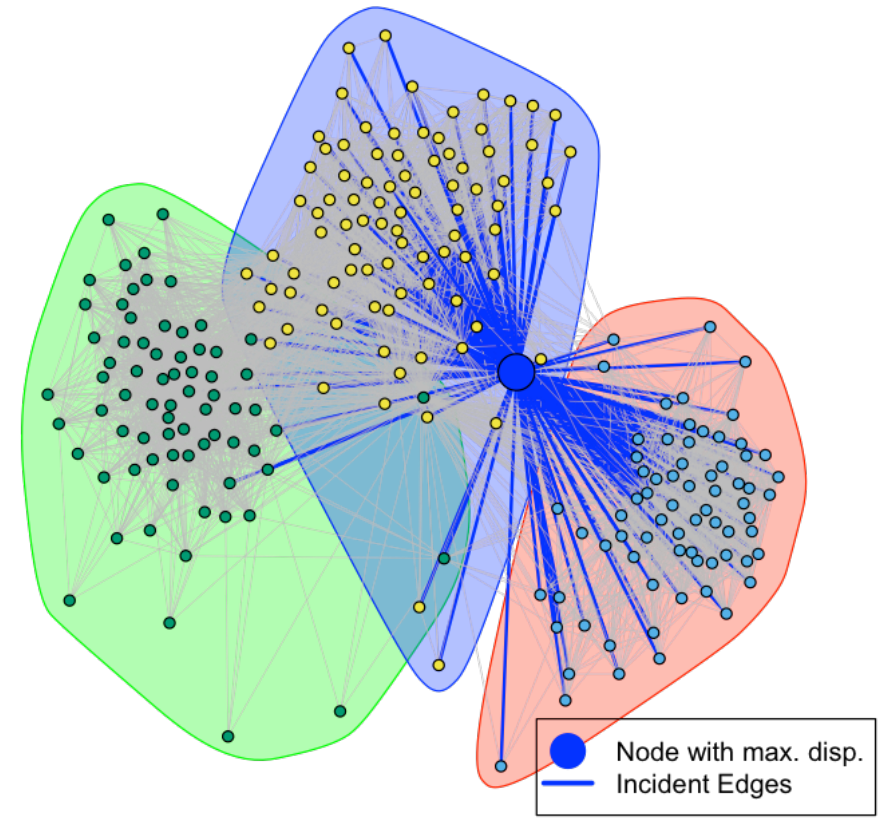
[32]:
```
el <- read.table("facebook_combined.txt",header=FALSE)
fb <- graph.data.frame(el, directed=FALSE)
node_list_str = c("0","348","483","1086","107")
for(j in c(1:5)){

    nodelist = unlist(ego(fb, order=1, nodes=node_list_str[j]))
    pn = induced.subgraph(fb,nodelist)
    pn$name = sort(nodelist)
    embeddedness <- c()
    disp <- c()
    deg <- c()
```

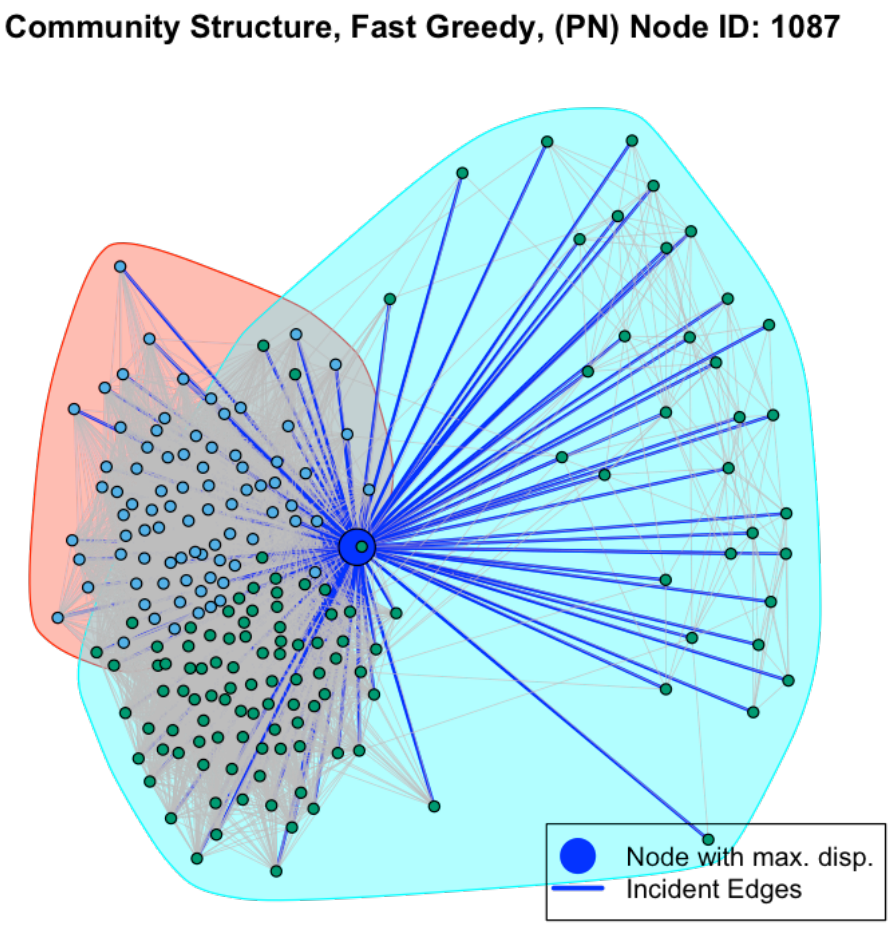
```
    i=1
    for(v in vertex_attr(pn)$name){
        if(v==node_list_str[j])
            next

        disp[i] = 0
        neh_ver = neighbors(pn, node_list_str[j])
        neh_core = neighbors(pn,v)
        inter = intersection(neh_ver, neh_core)
        embeddedness[i] = length(inter)

        deg[i] = degree(pn,v)
        eg2 = delete.vertices(pn, c(node_list_str[j], v))
        if(embeddedness[i]>1){
            ver = c()
            for(m in 1:length(inter)){
                ver =  c(ver,vertex_attr(pn)$name[inter[m]])
            }
            ver1=c()
            for(m in 1:length(ver)){
                ver1=c(ver1,which(vertex_attr(eg2)$name==ver[m]))
            }
            disp_mat = distances(eg2,v=ver1, to=ver1)
            disp_mat[disp_mat==Inf]<-diameter(eg2)+1
            disp[i] = sum(disp_mat)
        }
        i=i+1

    }

    maxdisp = pn$name[which(disp==max(disp))]
    maxnode = which(pn$name==maxdisp)

    fc <- cluster_fast_greedy(pn)
    vert_col = fc$membership+1
    vert_col[maxnode] = "blue"
    vert_size = rep(3, length(vert_col))
    vert_size[maxnode] = 10
    edge_col = rep("grey", length(E(pn)))
    edge_col[which(get.edgelist(pn, name = FALSE)[,1] == maxnode | get.
↪edgelist(pn, name = FALSE)[,2] == maxnode)] = "blue"
    edge_wid = rep(0.5,length(E(pn)))
    edge_wid[which(get.edgelist(pn, name = FALSE)[,1] == maxnode |  get.
↪edgelist(pn, name = FALSE)[,2] == maxnode)] = 2;
    plot(pn,mark.groups = fc,vertex.size=vert_size,edge.arrow.size=.5,vertex.
↪color=vert_col,edge.color = edge_col,edge.width=edge_wid,
```

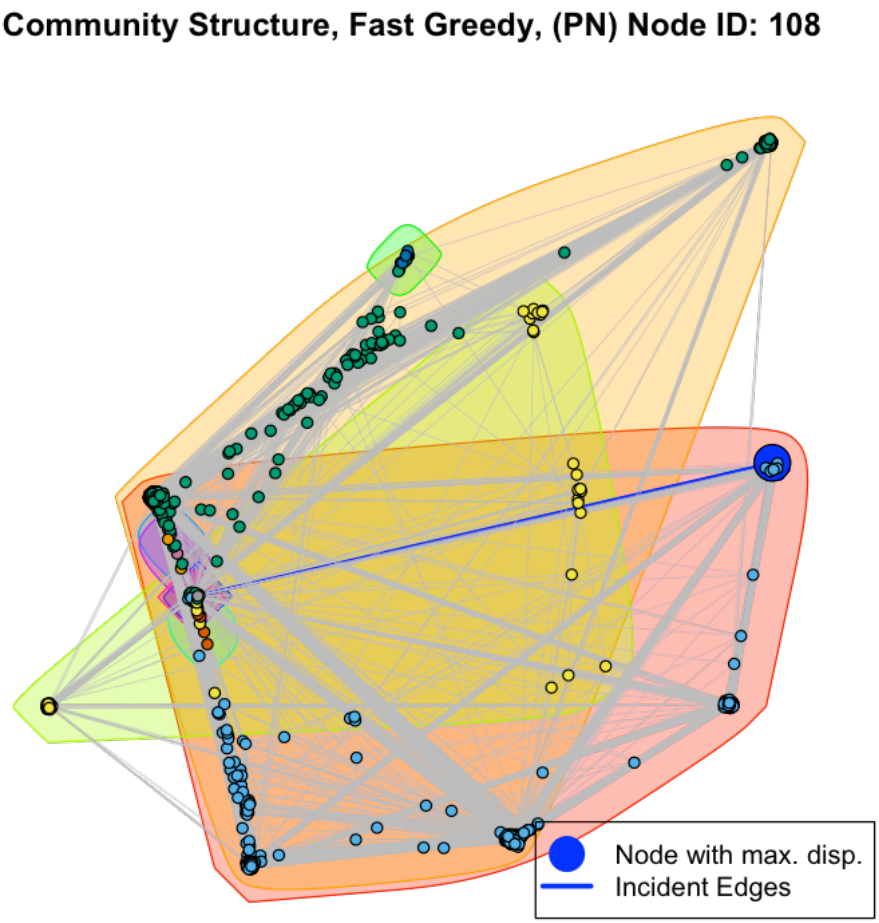
```
        vertex.label="",main = sprintf("Community Structure, Fast Greedy, (PN)␣
↪Node ID: %d", strtoi(node_list_str[j])+1))
    legend('bottomright', legend = c("Node with max. disp.", "Incident Edges"),
       lty = c(0, 1), lwd = c(5,3), pch=c(16,NA),
       col = c('blue','blue'), pt.cex=3)
    dev.copy2pdf(file=sprintf('Q13%d.pdf',strtoi(node_list_str[j])+1))

}
```




Community Structure, Fast Greedy, (PN) Node ID: 349
Node with max. disp.
Incident Edges


Community Structure, Fast Greedy, (PN) Node ID: 484
Node with max. disp.
Incident Edges


Community Structure, Fast Greedy, (PN) Node ID: 1087
Node with max. disp.
Incident Edges



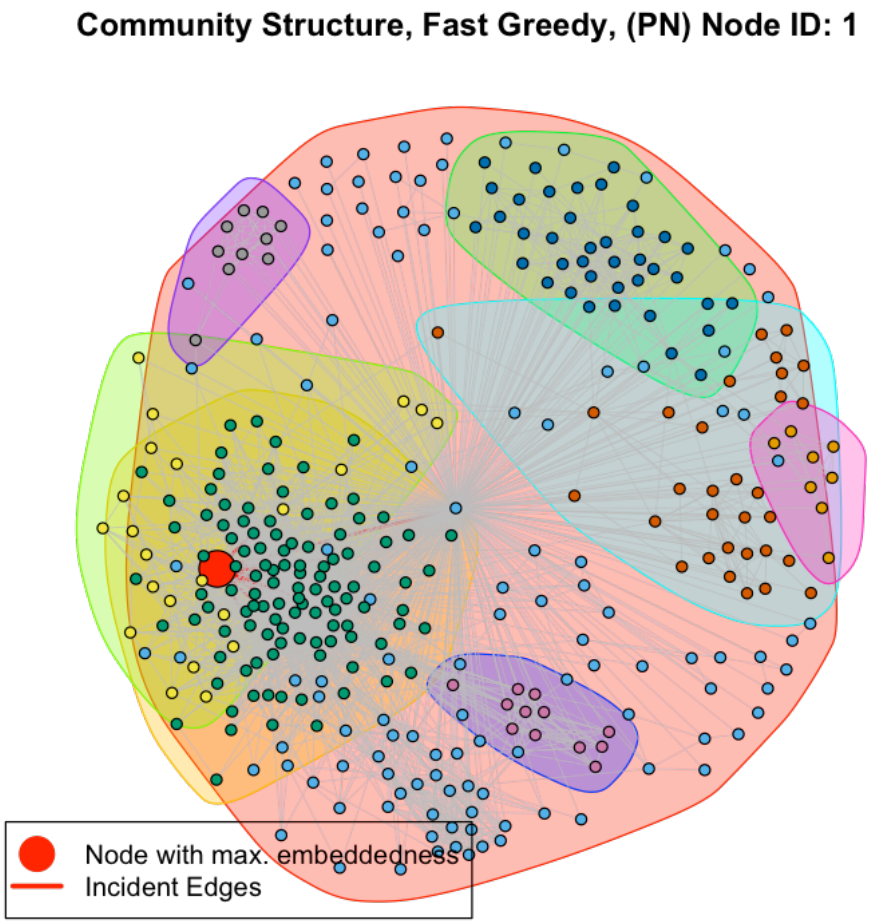
1.0.13 Question 14

[33]:
```
el <- read.table("facebook_combined.txt",header=FALSE)
fb <- graph.data.frame(el, directed=FALSE)
node_list_str = c("0","348","483","1086","107")
for(j in c(1:5)){

    nodelist = unlist(ego(fb, order=1, nodes=node_list_str[j]))
    pn = induced.subgraph(fb,nodelist)
    pn$name = sort(nodelist)
    embeddedness <- c()
    disp <- c()
    deg <- c()
```

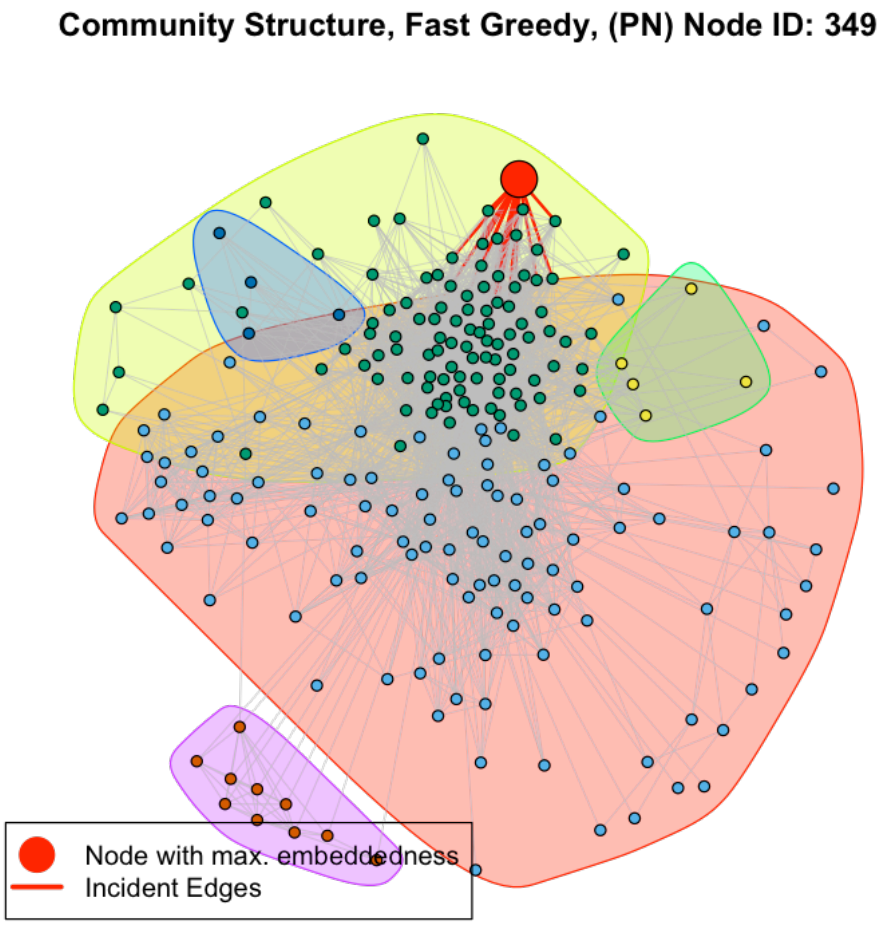
```
    i=1
    for(v in vertex_attr(pn)$name){
        if(v==node_list_str[j])
            next

        disp[i] = 0
        neh_ver = neighbors(pn, node_list_str[j])
        neh_core = neighbors(pn,v)
        inter = intersection(neh_ver, neh_core)
        embeddedness[i] = length(inter)

        deg[i] = degree(pn,v)
        eg2 = delete.vertices(pn, c(node_list_str[j], v))
        if(embeddedness[i]>1){
            ver = c()
            for(m in 1:length(inter)){
                ver =  c(ver,vertex_attr(pn)$name[inter[m]])
            }
            ver1=c()
            for(m in 1:length(ver)){
                ver1=c(ver1,which(vertex_attr(eg2)$name==ver[m]))
            }
            disp_mat = distances(eg2,v=ver1, to=ver1)
            disp_mat[disp_mat==Inf]<-diameter(eg2)+1
            disp[i] = sum(disp_mat)
        }
        i=i+1

    }

    maxemb = pn$name[which(embeddedness==max(embeddedness))]
    maxnode = which(pn$name==maxemb)
    print(sprintf("ID of node with max. emb. for core node with ID %d:␣
↪%d",strtoi(node_list_str[j])+1,maxnode))

    fc <- cluster_fast_greedy(pn)
    vert_col = fc$membership+1
    vert_col[maxnode] = "red"
    vert_size = rep(3, length(vert_col))
    vert_size[maxnode] = 10
    edge_col = rep("grey", length(E(pn)))
    edge_col[which(get.edgelist(pn, name = FALSE)[,1] == maxnode | get.
↪edgelist(pn, name = FALSE)[,2] == maxnode)] = "red"
    edge_wid = rep(0.5,length(E(pn)))
    edge_wid[which(get.edgelist(pn, name = FALSE)[,1] == maxnode |  get.
↪edgelist(pn, name = FALSE)[,2] == maxnode)] = 2;
```

```
    plot(pn,mark.groups = fc,vertex.size=vert_size,edge.arrow.size=.5,vertex.
↪color=vert_col,edge.color = edge_col,edge.width=edge_wid,
         vertex.label="",main = sprintf("Community Structure, Fast Greedy, (PN)␣
↪Node ID: %d", strtoi(node_list_str[j])+1))
    legend('bottomleft', legend = c("Node with max. embeddedness", "Incident␣
↪Edges"),
       lty = c(0, 1), lwd = c(5,3), pch=c(16,NA),
       col = c('red','red'), pt.cex=3)
    dev.copy2pdf(file=sprintf('Q14%d.pdf',strtoi(node_list_str[j])+1))

}
```

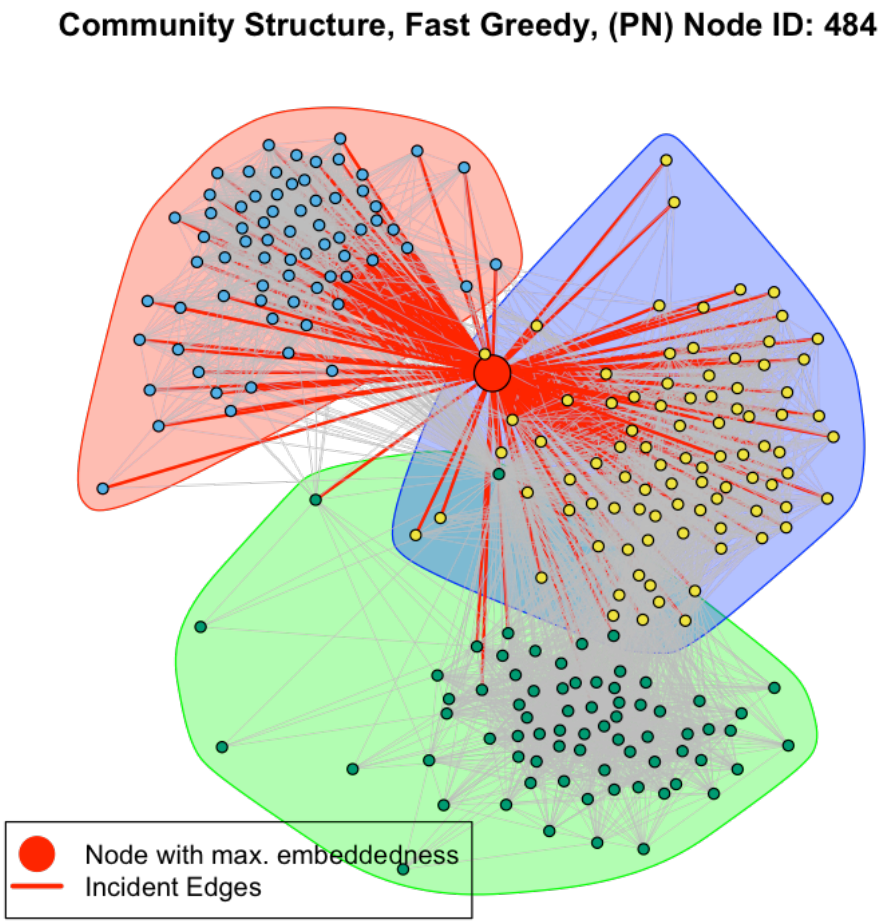
```
[1] "ID of node with max. emb. for core node with ID 1: 49"
[1] "ID of node with max. emb. for core node with ID 349: 31"
```



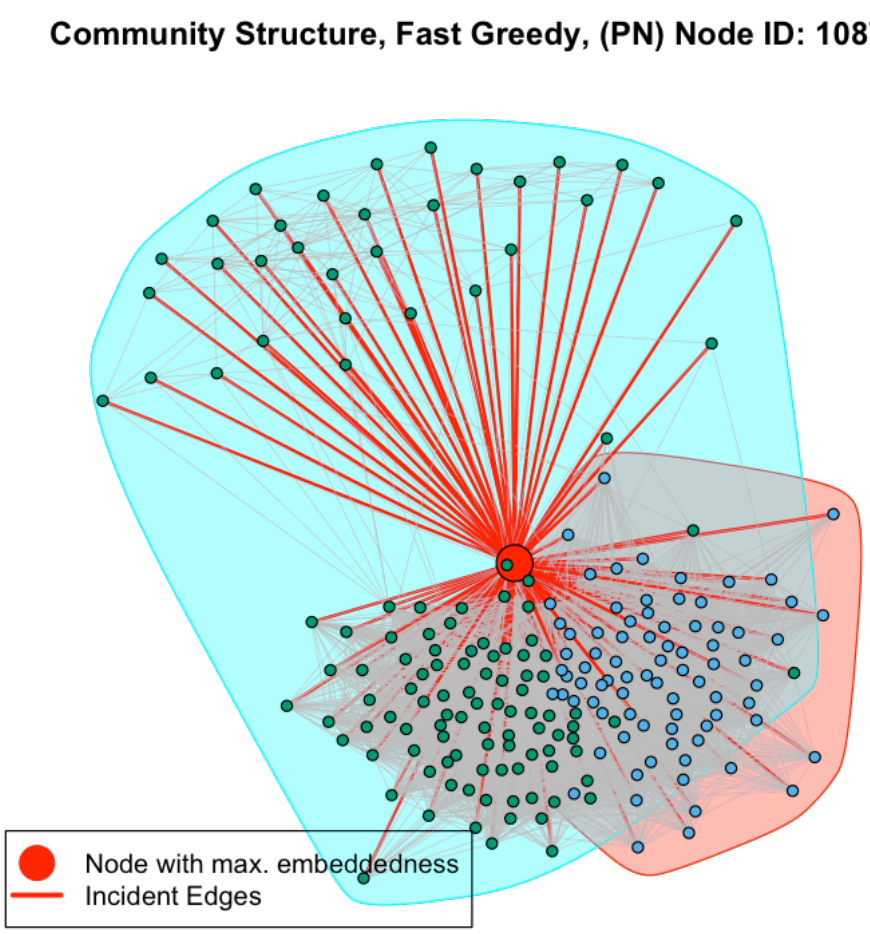
```
[1] "ID of node with max. emb. for core node with ID 484: 1"
```

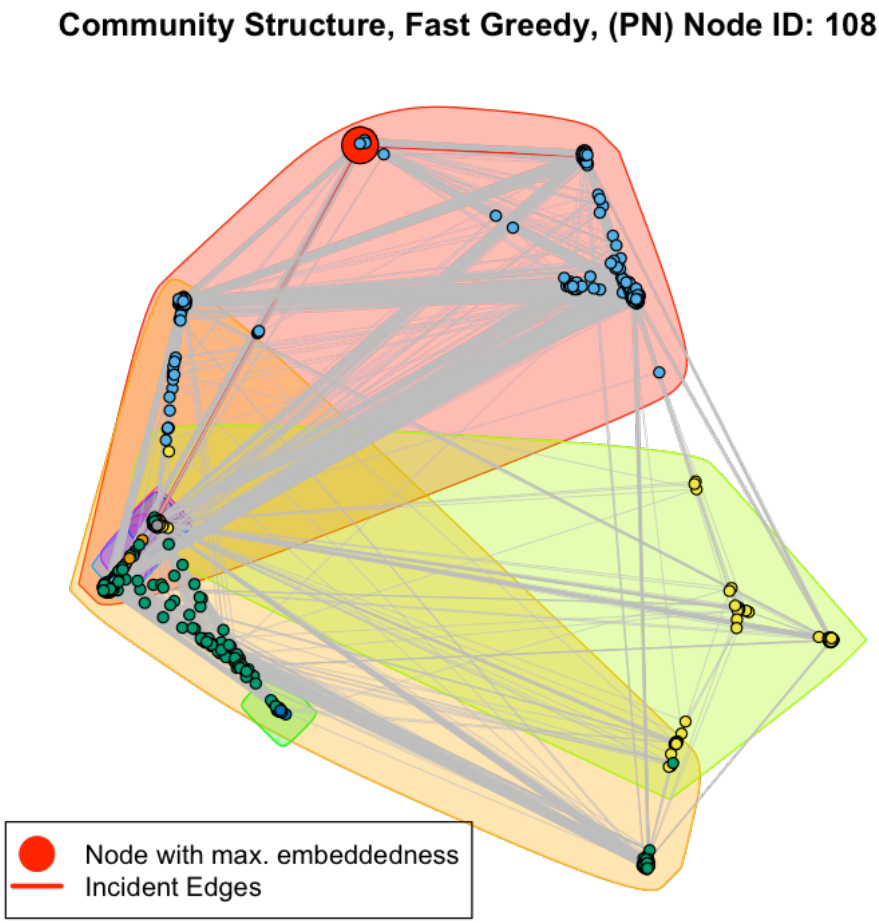



```
[1] "ID of node with max. emb. for core node with ID 1087: 1"
```




```
[1] "ID of node with max. emb. for core node with ID 108: 977"
```


Community Structure, Fast Greedy, (PN) Node ID: 1087
Node with max. embeddedness
Incident Edges

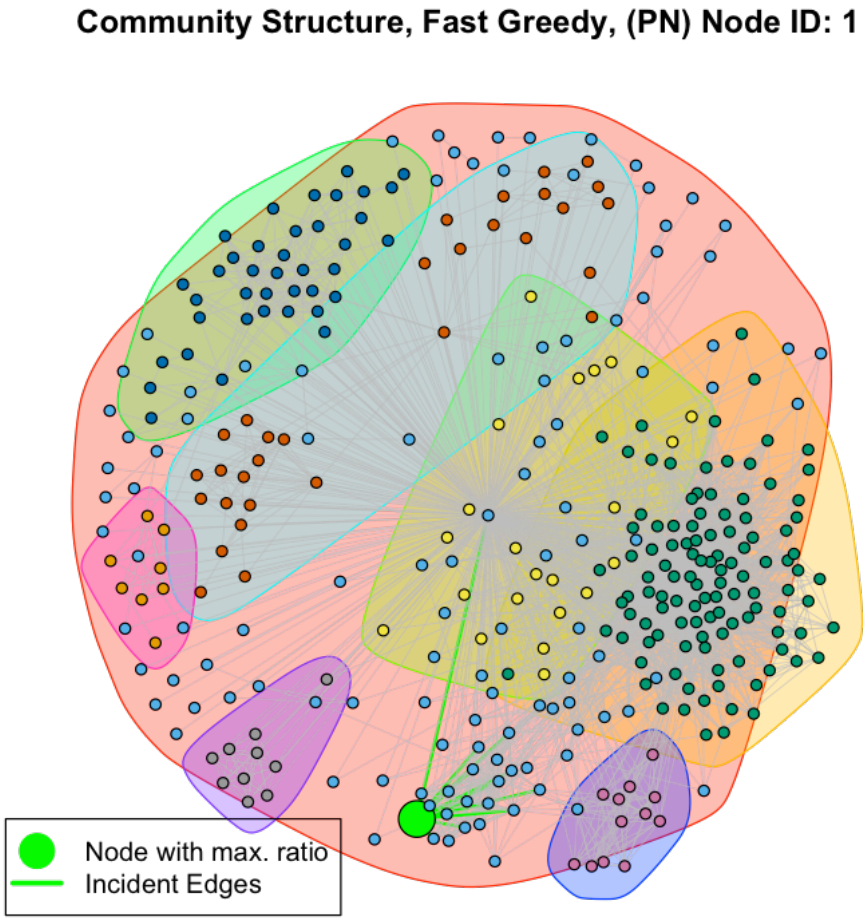


[34]:
```
el <- read.table("facebook_combined.txt",header=FALSE)
fb <- graph.data.frame(el, directed=FALSE)
node_list_str = c("0","348","483","1086","107")
for(j in c(1:5)){

    nodelist = unlist(ego(fb, order=1, nodes=node_list_str[j]))
    pn = induced.subgraph(fb,nodelist)
    pn$name = sort(nodelist)
    embeddedness <- c()
    disp <- c()
    deg <- c()
    i=1
    for(v in vertex_attr(pn)$name){
```

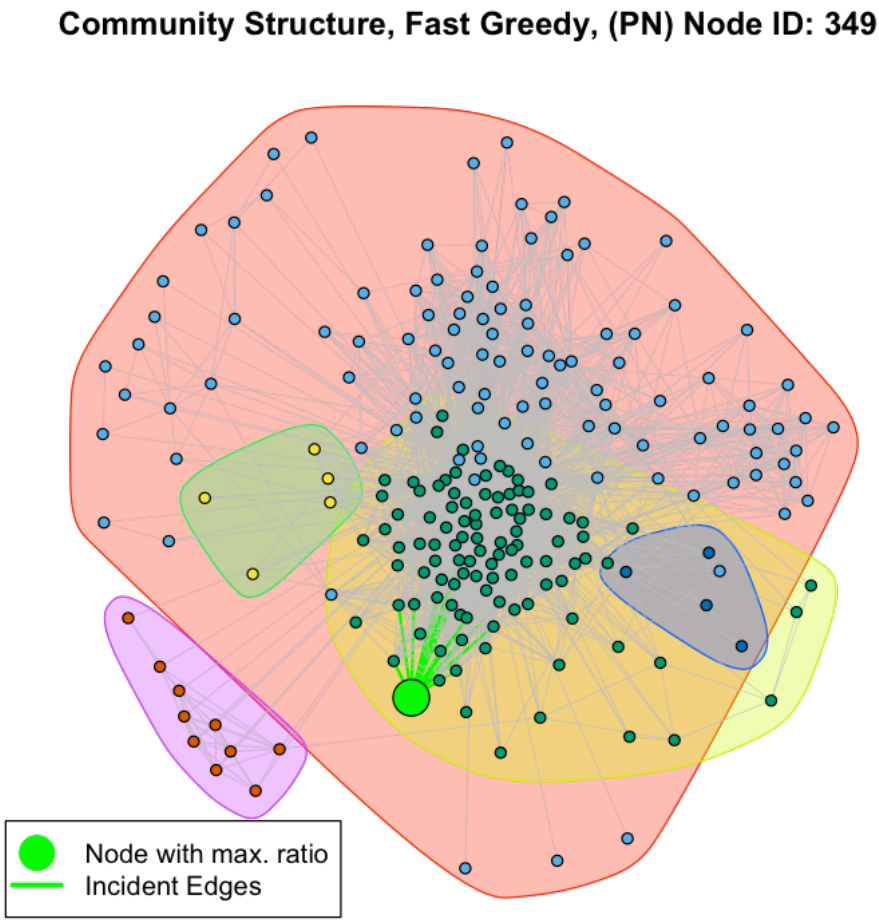
```
        if(v==node_list_str[j])
            next

        disp[i] = 0
        neh_ver = neighbors(pn, node_list_str[j])
        neh_core = neighbors(pn,v)
        inter = intersection(neh_ver, neh_core)
        embeddedness[i] = length(inter)

        deg[i] = degree(pn,v)
        eg2 = delete.vertices(pn, c(node_list_str[j], v))
        if(embeddedness[i]>1){
            ver = c()
            for(m in 1:length(inter)){
                ver =  c(ver,vertex_attr(pn)$name[inter[m]])
            }
            ver1=c()
            for(m in 1:length(ver)){
                ver1=c(ver1,which(vertex_attr(eg2)$name==ver[m]))
            }
            disp_mat = distances(eg2,v=ver1, to=ver1)
            disp_mat[disp_mat==Inf]<-diameter(eg2)+1
            disp[i] = sum(disp_mat)
        }
        i=i+1

    }

    ratio = disp/embeddedness
    ratio[mapply(is.nan,ratio)]=0
    maxratio=pn$name[which(ratio==max(ratio))]
    maxnode = which(pn$name==maxratio)
    print(sprintf("ID of node with max. ratio for core node with ID %d:␣
↪%d",strtoi(node_list_str[j])+1,maxnode))

    fc <- cluster_fast_greedy(pn)
    vert_col = fc$membership+1
    vert_col[maxnode] = "green"
    vert_size = rep(3, length(vert_col))
    vert_size[maxnode] = 10
    edge_col = rep("grey", length(E(pn)))
    edge_col[which(get.edgelist(pn, name = FALSE)[,1] == maxnode | get.
↪edgelist(pn, name = FALSE)[,2] == maxnode)] = "green"
    edge_wid = rep(0.5,length(E(pn)))
    edge_wid[which(get.edgelist(pn, name = FALSE)[,1] == maxnode |  get.
↪edgelist(pn, name = FALSE)[,2] == maxnode)] = 2;
```

```
    plot(pn,mark.groups = fc,vertex.size=vert_size,edge.arrow.size=.5,vertex.
↪color=vert_col,edge.color = edge_col,edge.width=edge_wid,
         vertex.label="",main = sprintf("Community Structure, Fast Greedy, (PN)␣
↪Node ID: %d", strtoi(node_list_str[j])+1))
    legend('bottomleft', legend = c("Node with max. ratio", "Incident Edges"),
       lty = c(0, 1), lwd = c(5,3), pch=c(16,NA),
       col = c('green','green'), pt.cex=3)
    dev.copy2pdf(file=sprintf('Q14b%d.pdf',strtoi(node_list_str[j])+1))

}
```

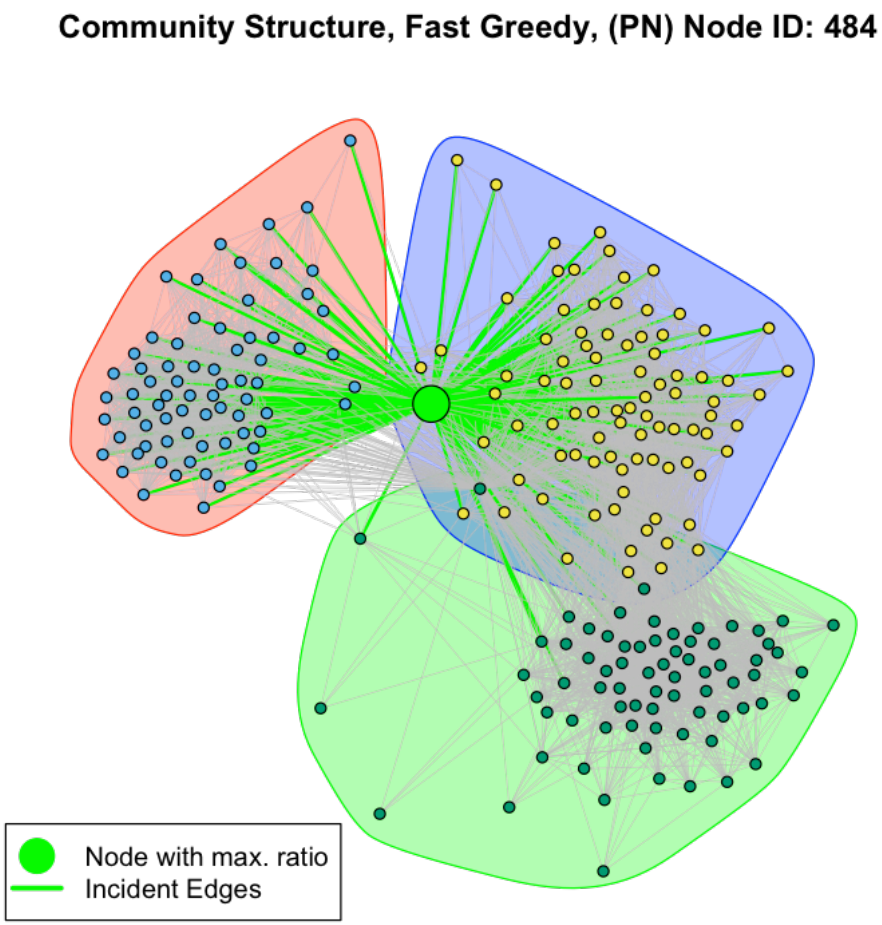
```
[1] "ID of node with max. ratio for core node with ID 1: 21"
[1] "ID of node with max. ratio for core node with ID 349: 31"
```



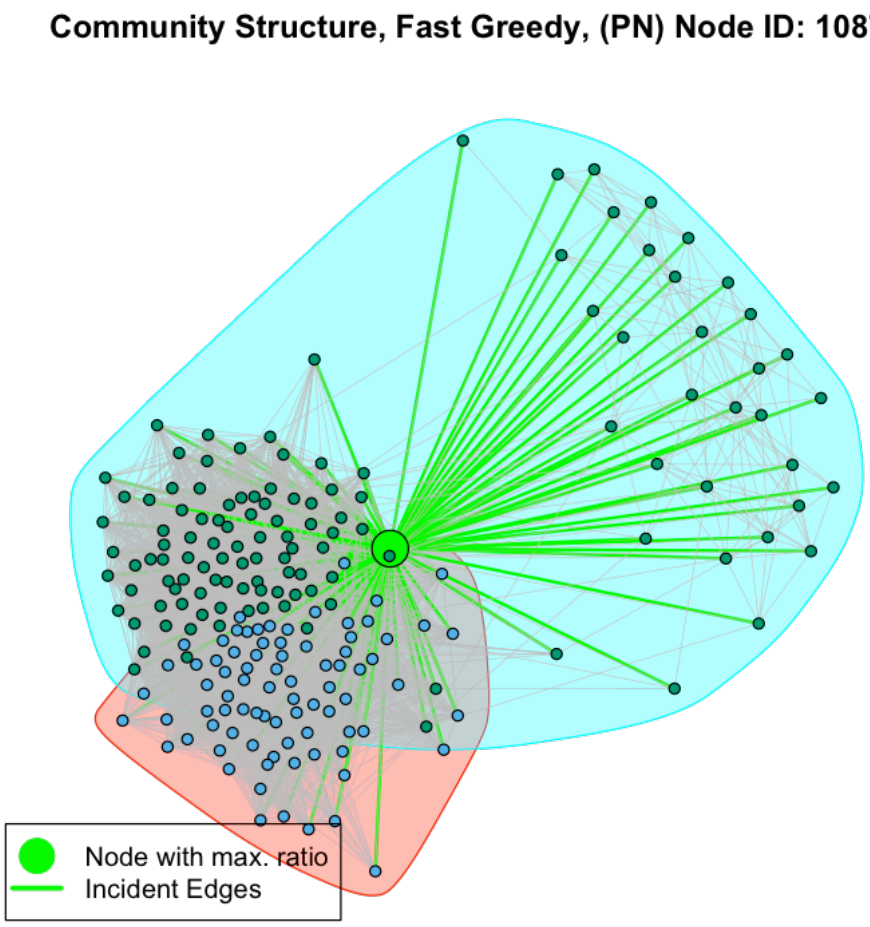
```
[1] "ID of node with max. ratio for core node with ID 484: 1"
```

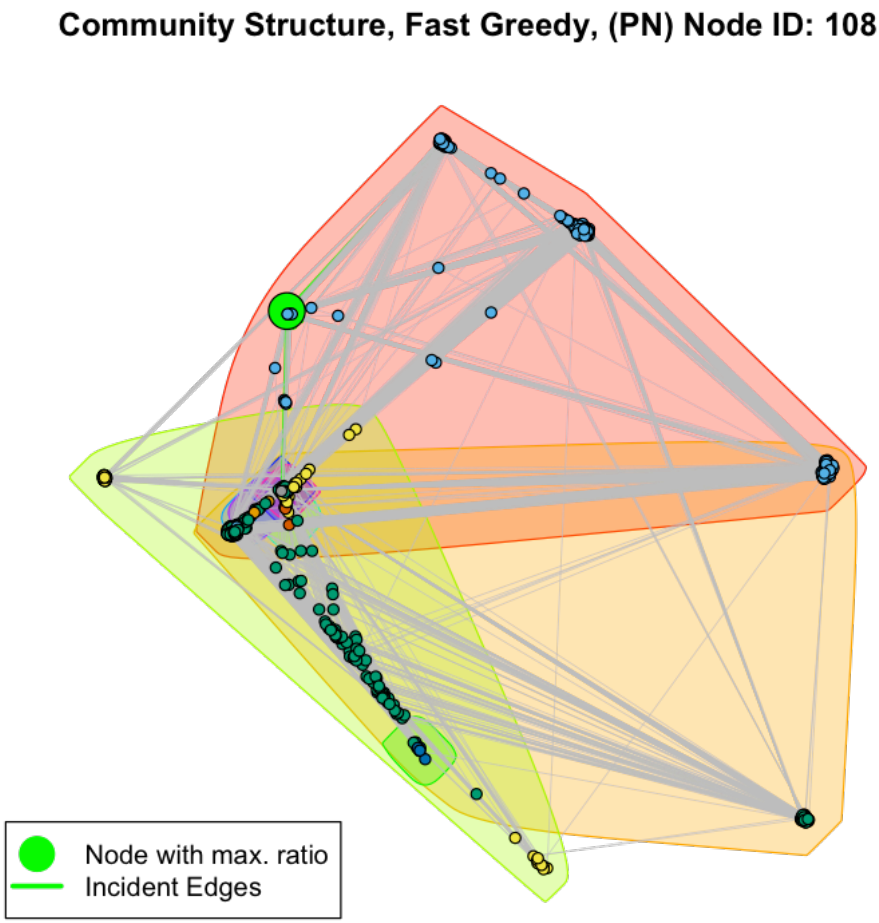



```
[1] "ID of node with max. ratio for core node with ID 1087: 1"
```




```
[1] "ID of node with max. ratio for core node with ID 108: 977"
```


Community Structure, Fast Greedy, (PN) Node ID: 1087
Node with max. ratio
Incident Edges



1.0.14 Question 16

[8]:
```
el <- read.table("facebook_combined.txt")
g <- graph.data.frame(el, directed=FALSE)
g_pn = make_ego_graph(g, nodes=c('414'))
Nr = which(degree(g_pn[[1]]) == 24)
print(length(Nr))
```

```
[1] 11
```

1.0.15 Question 17

[21]:
```
common.neighbors <- function(graph, i, j) {
    s_i <- neighbors(graph, i, mode = c("all"))
    s_j <- neighbors(graph, j, mode = c("all"))
    inter <- intersection(s_i, s_j)
    length(inter)
}

jaccard <- function(graph, i, j) {
    s_i <- neighbors(graph, i, mode = c("all"))
    s_j <- neighbors(graph, j, mode = c("all"))
    inter <- intersection(s_i, s_j)
    length(inter)/(length(s_i)+length(s_j))
}

adamic.adar <- function(graph, i, j) {
    s_i <- neighbors(graph, i, mode = c("all"))
    s_j <- neighbors(graph, j, mode = c("all"))
    inter <- intersection(s_i, s_j)
    sum <- 0
    for(k in inter) {
        s_k <- neighbors(graph, k, mode = c("all"))
        sum <- sum + 1/(log(length(s_k)))
    }
    sum
}

not.friends <- function(graph, i) {
    ver <- V(graph)
    s_i <- neighbors(graph, i, mode = c("all"))
    not_friend <- ver[!ver %in% s_i]
    not_friend
}

friend_recs <- function(graph, i, neighborhood_measure, num_of_recs) {
    measure <- c()
    not_friends <- not.friends(graph, i)
    for(j in not_friends){
        if(neighborhood_measure == "common"){
            measure <- c(measure, common.neighbors(graph, i, j))
        } else if(neighborhood_measure == "jaccard") {
            measure <- c(measure, jaccard(graph, i, j))
        } else if(neighborhood_measure == "adamic") {
            measure <- c(measure, adamic.adar(graph, i, j))
        }
    }
```

```
    measure_sorted <- sort(measure, decreasing = TRUE, index.return=TRUE)
    recd_friends <- not_friends[measure_sorted$ix]
    recd_friends[c(1:num_of_recs)]
}

select.edges <- function(graph, i, p) {
    incident_edges <-incident(graph, v=i, mode = c("all"))
    sample(incident_edges, p*degree(graph,v=i))
}

delete.edges <- function(graph, edges) {
    delete_edges(graph, edges)
}

add.edges <- function(graph, i, edges) {
    for(j in 1:length(edges)) {
        graph <- graph + edge(i, edges[j])
    }
    graph
}

accuracy <- function(graph, graph3, i) {
    s_i <- neighbors(graph, i, mode = c("all"))
    s_j <- neighbors(graph3, i, mode = c("all"))
    inter <- intersection(s_i, s_j)
    length(inter)/length(s_i)
}

rec.accuracy <- function(graph, i, p, user_list, neighborhood_measure) {
    edges_to_delete <- select.edges(graph, i, p)
    graph2 <- delete.edges(graph, edges_to_delete)
    recommended <- friend_recs(graph2, i, neighborhood_measure,␣
 ↪length(edges_to_delete))
    graph3 <- add.edges(graph2, i, recommended)
    accuracy(graph, graph3, i)
}
```

[24]:
```
node_ind <- c()
eg_tmp <- g_pn[[1]]
ver_tmp <- V(eg_tmp)
for(i in 1:length(degree(eg_tmp))) {
    if(degree(eg_tmp, v=i) == 24) {
        node_ind <- c(node_ind,i)
    }
}
user_list <- ver_tmp[node_ind]
graph <- g_pn[[1]]
```

```
p <- 0.75
neighborhood_measure <- c("common", "jaccard", "adamic")

for(measure in neighborhood_measure) {
    avg <- c()
    for(i in user_list) {
        sum <- 0
        for(m in 1:10) {
            recs <- rec.accuracy(graph, i, p, user_list, measure)
            sum <- sum + recs
        }
        avg <- c(avg, sum/10)
    }
    print(measure)
    print(mean(avg))
}
```

```
[1] "common"
[1] 0.8878788
[1] "jaccard"
[1] 0.8530303
[1] "adamic"
[1] 0.8806818
```

2 Google Plus Network

2.0.1 Question 18

[25]:
```
circle_list = list.files(path="gplus", pattern="*.circles")
count = 0
for (file in circle_list){
  file = paste("gplus", file, sep="/")
  if (length(readLines(file)) > 2){
    count = count + 1
  }
}
print(count)
```

```
[1] 57
```

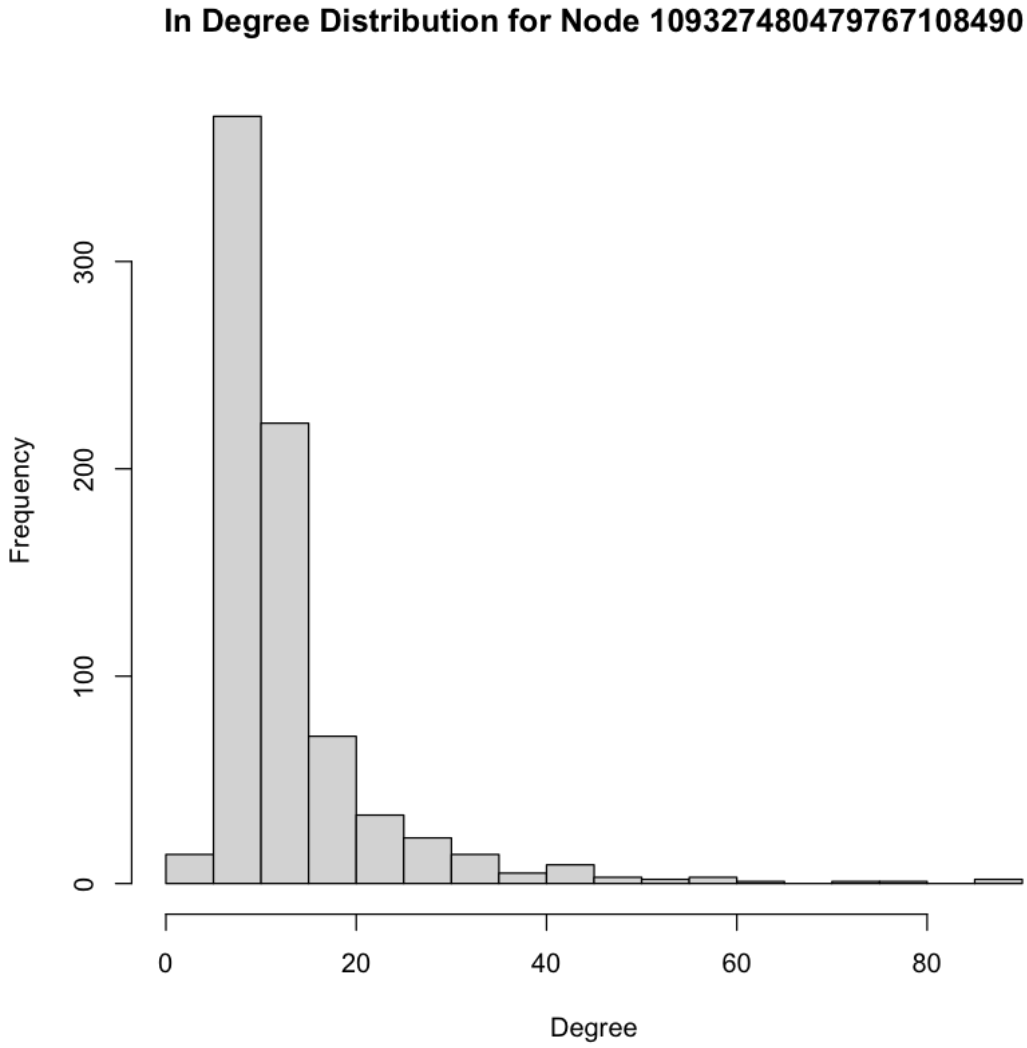
2.0.2 Question 19

[26]:
```
gp1_e <- read.table("gplus/109327480479767108490.edges")
gp2_e <- read.table("gplus/115625564993990145546.edges")
gp3_e <- read.table("gplus/101373961279443806744.edges")
gp1 <- graph.data.frame(gp1_e, directed=TRUE)
gp2 <- graph.data.frame(gp2_e, directed=TRUE)
```

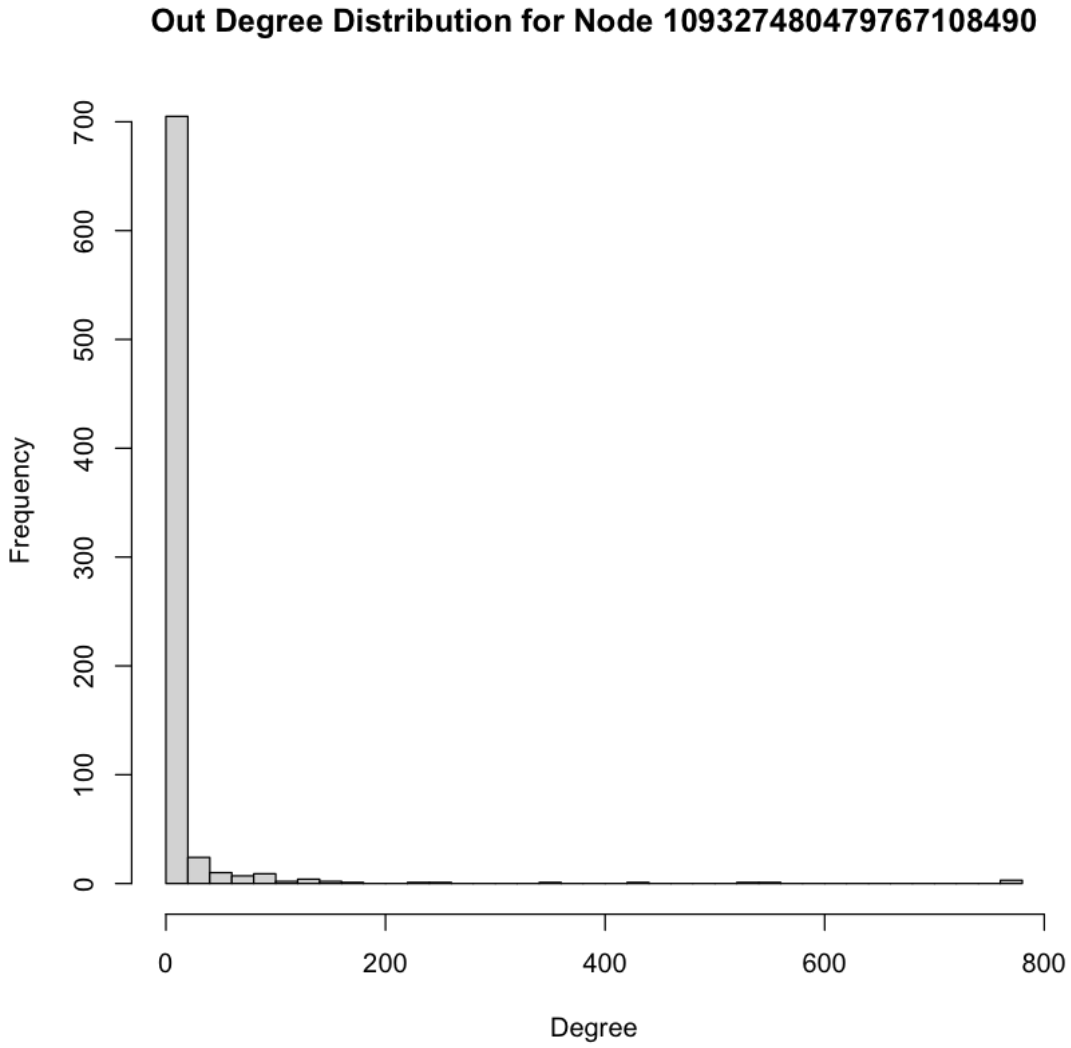
```
gp3 <- graph.data.frame(gp3_e, directed=TRUE)
```

[4]:
```
hist(degree(gp1,mode = c("in")),breaks=30,main= "In Degree Distribution for␣
 ↪Node 109327480479767108490", xlab="Degree",ylab="Frequency")
dev.copy2eps(file="Q19a.eps")
hist(degree(gp1,mode = c("out")),breaks=30,main= "Out Degree Distribution for␣
 ↪Node 109327480479767108490", xlab="Degree",ylab="Frequency")
dev.copy2eps(file="Q19b.eps")
hist(degree(gp2,mode = c("in")),breaks=30,main= "In Degree Distribution for␣
 ↪Node 115625564993990145546", xlab="Degree",ylab="Frequency")
dev.copy2eps(file="Q19c.eps")
hist(degree(gp2,mode = c("out")),breaks=30,main= "Out Degree Distribution for␣
 ↪Node 115625564993990145546", xlab="Degree",ylab="Frequency")
dev.copy2eps(file="Q19d.eps")
hist(degree(gp3,mode = c("in")),breaks=30,main= "In Degree Distribution for␣
 ↪Node 101373961279443806744", xlab="Degree",ylab="Frequency")
dev.copy2eps(file="Q19e.eps")
hist(degree(gp3,mode = c("out")),breaks=30,main= "Out Degree Distribution for␣
 ↪Node 101373961279443806744", xlab="Degree",ylab="Frequency")
dev.copy2eps(file="Q19f.eps")
```

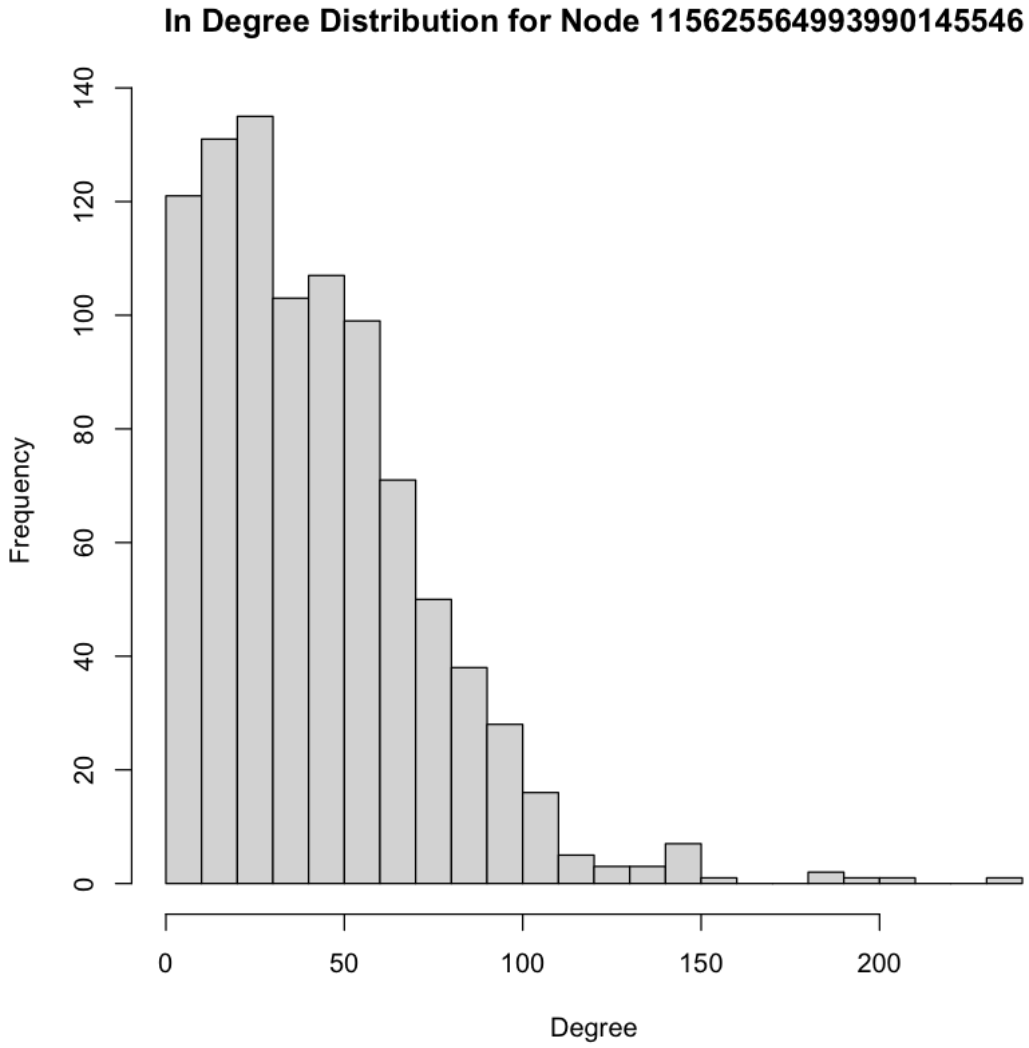
pdf: 2


In Degree Distribution for Node 109327480479767108490
Frequency
0
100
200
300
0
20
40
60
80
Degree

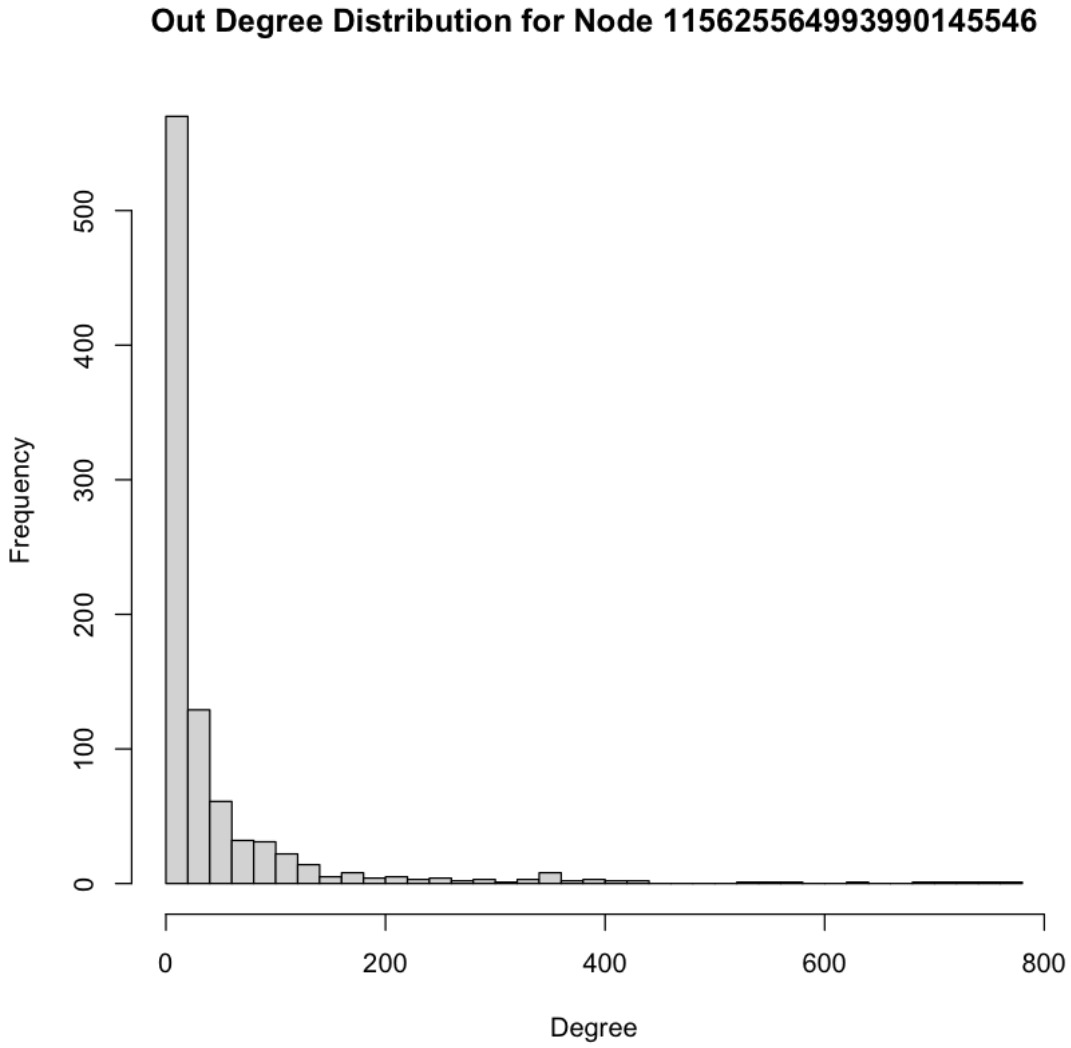
pdf: 2


Out Degree Distribution for Node 109327480479767108490
Frequency
0
100
200
300
400
500
600
700
0
200
400
600
800
Degree

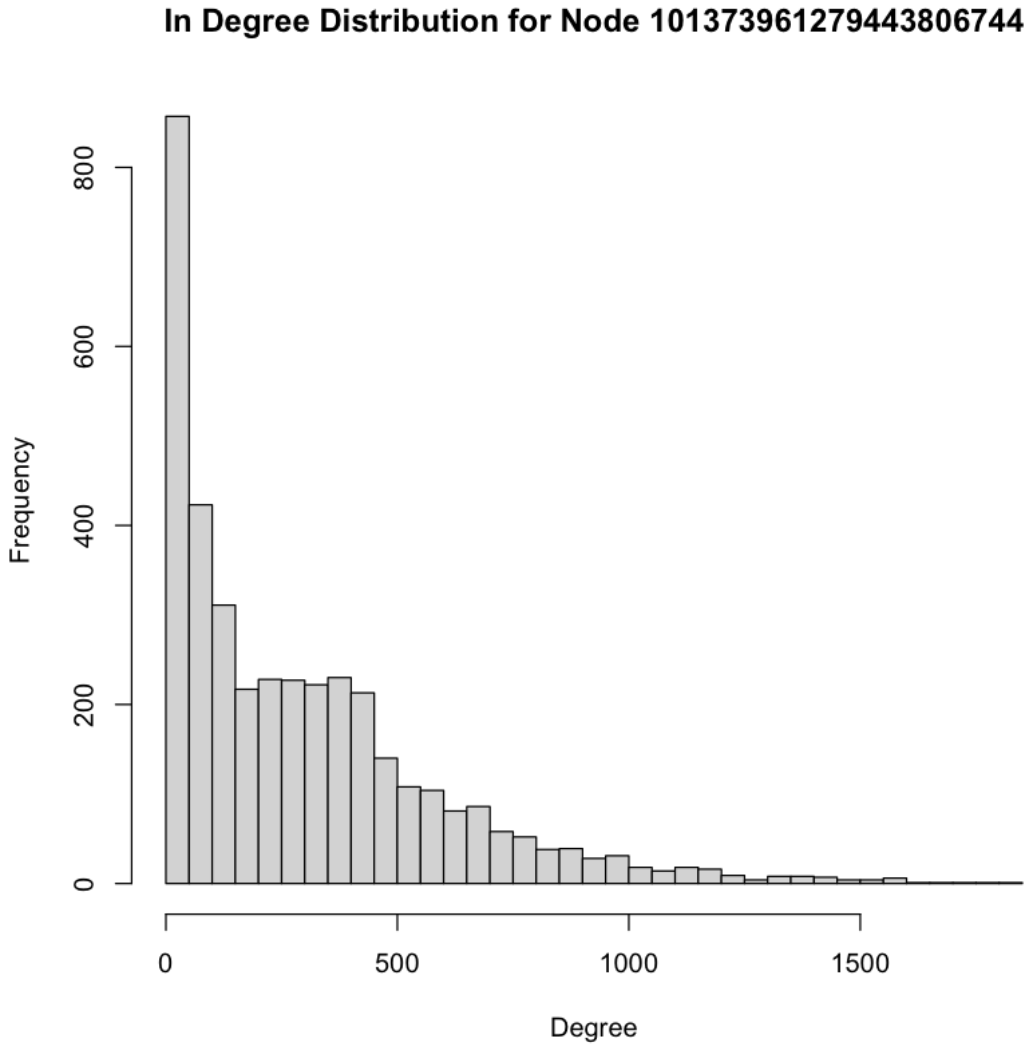
pdf: 2


In Degree Distribution for Node 11562556499399014554 6
Frequency
0
20
40
60
80
100
120
140
0
50
100
150
200
Degree

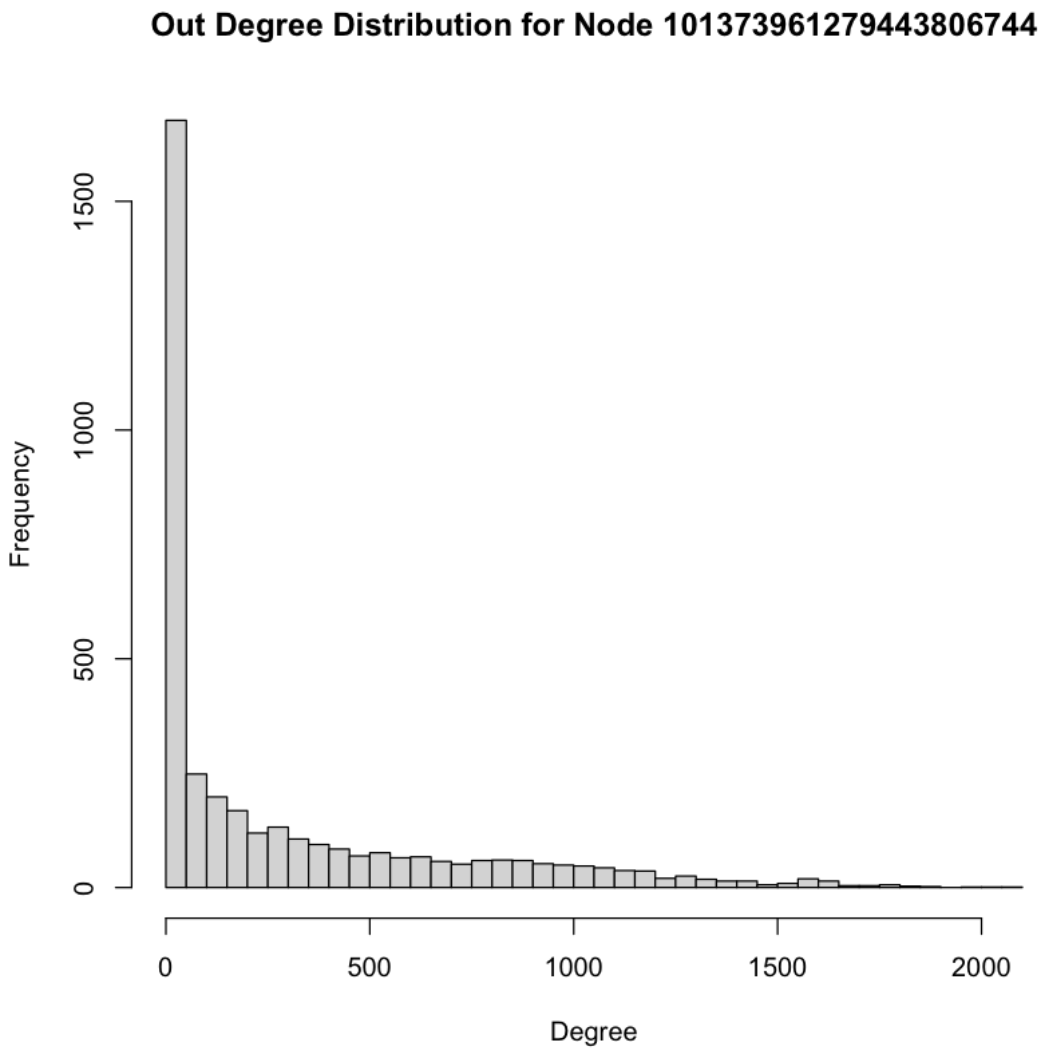
pdf: 2


Out Degree Distribution for Node 11562556499399014554 6
Frequency
0
100
200
300
400
500
0
200
400
600
800
Degree

pdf: 2


In Degree Distribution for Node 101373961279443806744
Frequency
0
200
400
600
800
0
500
1000
1500
Degree

pdf: 2



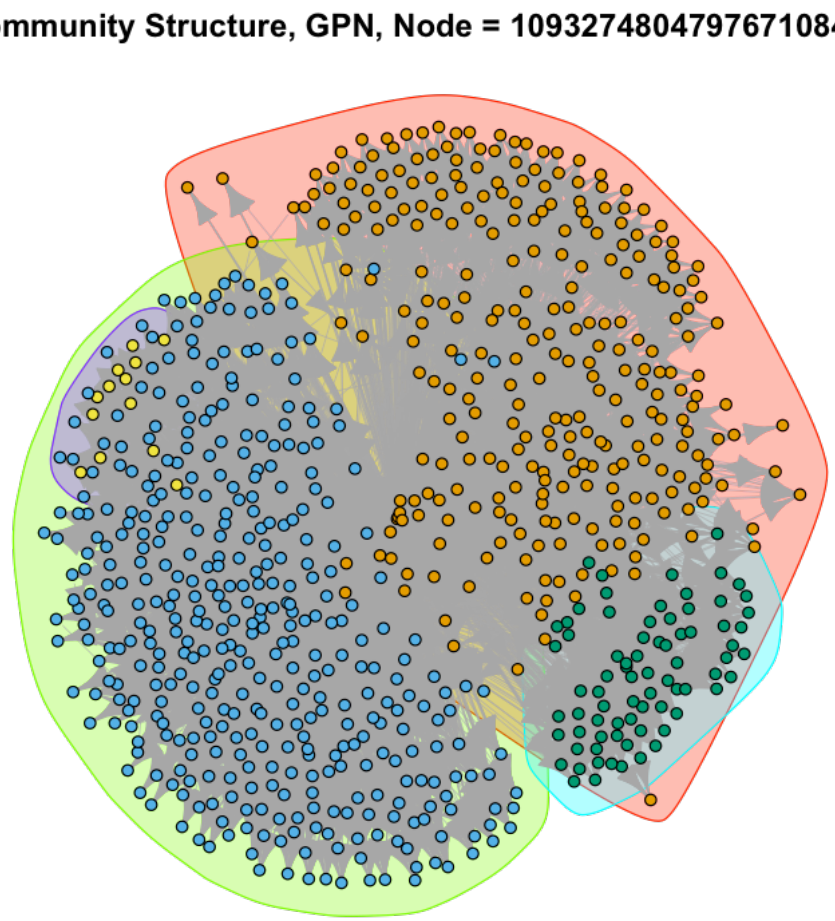
2.0.3 Question 20

[5]:
```
gp1 <- graph.data.frame(gp1_e, directed=TRUE)
gp1 = gp1 + vertex("10932748047976710849O")
for (i in seq(1, vcount(gp1)-1,1)){
  gp1 = gp1 + edge(vcount(gp1), i)
}
wt1 <- cluster_walktrap(gp1)
modularity(wt1)
plot(gp1,mark.groups = wt1,edge.width=0.5,vertex.size=3,vertex.
↪color=wt1$membership,vertex.label="",main = "Community Structure, GPN, Node␣
↪= 10932748047976710849O")
```

```
dev.copy2pdf(file="Q20a.pdf")
```

0.252765387296677

pdf: 2

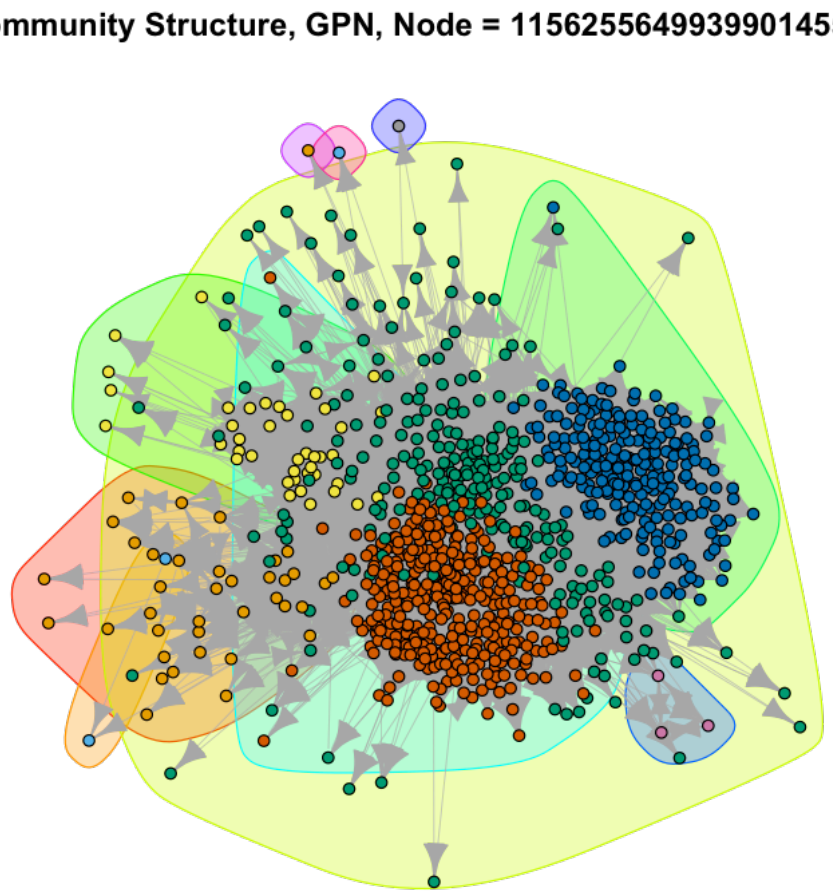


[6]:
```
gp2 <- graph.data.frame(gp2_e, directed=TRUE)
gp2 = gp2 + vertex("115625564993990145546")
for (i in seq(1, vcount(gp2)-1,1)){
  gp2 = gp2 + edge(vcount(gp2), i)
}
wt2 <- cluster_walktrap(gp2)
modularity(wt2)
```

```
plot(gp2,mark.groups = wt2,edge.width=0.5,vertex.size=3,vertex.
 ↪color=wt2$membership,vertex.label="",main = "Community Structure, GPN, Node␣
 ↪= 115625564993990145546")
dev.copy2pdf(file="Q20b.pdf")
```

0.319472551345825

pdf: 2

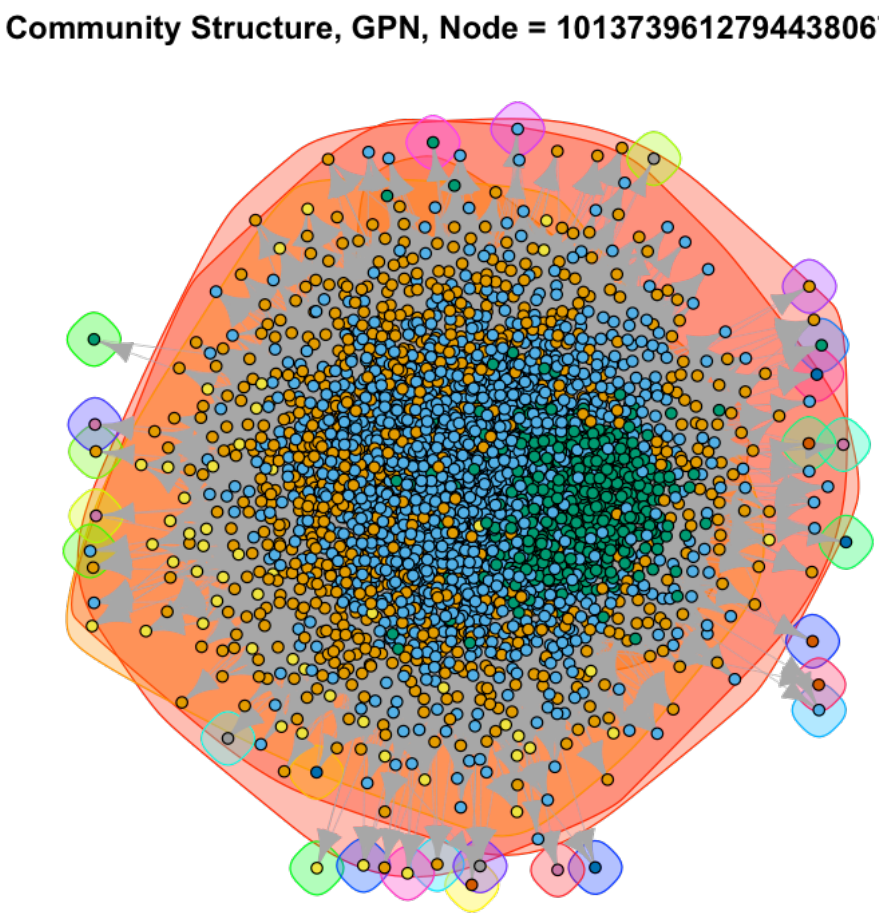


[27]:
```
gp3 <- graph.data.frame(gp3_e, directed=TRUE)
gp3 = gp3 + vertex("101373961279443806744")
for (i in seq(1, vcount(gp3)-1,1)){
  gp3 = gp3 + edge(vcount(gp3), i)
```

```
}
wt3 <- cluster_walktrap(gp3)
modularity(wt3)
plot(gp3,mark.groups = wt3,edge.width=0.5,vertex.size=3,vertex.
 ↪color=wt3$membership,vertex.label="",main = "Community Structure, GPN, Node␣
 ↪= 101373961279443806744")
dev.copy2pdf(file="Q20c.pdf")
```

0.191090270876884

pdf: 2



2.0.4 Question 22

[2]:
```
circlefile <- readLines("gplus/109327480479767108490.circles")
circles = list()

#Obtain circles
for (j in 1:length(circlefile)) {
    circle_nodes = strsplit(circlefile[j],"\t")
    circles = c(circles, list(circle_nodes[[1]][-1]))
}

#Calculate N
all_circle <- c()
for (l in circles){
    all_circle <- c(all_circle,l)
}
all_circle <- unique(all_circle)
N <- length(all_circle)

#Calculate H(C)
h_c <- 0
for (l in circles){
    h_c<- h_c+ ((length(l)/N)* log10(length(l)/N))
}
h_c<-h_c*-1

#Extract Community Structures
g1<-read_graph("gplus/109327480479767108490.edges",format="ncol",directed=TRUE)
g1 = add.vertices(g1, nv = 1, name = "109327480479767108490")
index = which(V(g1)$name=="109327480479767108490")
el = c()
for (vertex in 1:(vcount(g1) - 1)) {
    el = c(el, c(index, vertex))
}
g1 = add_edges(g1, el)
fg <- walktrap.community(g1)

#Find H(K),H(C|K) and H(K|C).
percentage = vector()
percentage_circle = vector()
hck<-0
hkc<-0
hk<-0
for(m in 1:max(fg$membership)){
    community_nodes = V(g1)$name[which(fg$membership == m)]
    bi<-length(intersect(community_nodes,all_circle))
    if(bi!=0) {
```

```
    hk<- hk + ((bi/N) * log10(bi/N))
    for (n in 1:length(circles)) {
      common_nodes = intersect(community_nodes, circles[[n]])
        cij<-length(unique(common_nodes))
        ai<-length(circles[[n]])
        if(cij!=0){
            hck<-hck+ (cij/N)*log10(cij/bi)
            hkc <- hkc + (cij/N)*log10(cij/ai)
            }
        }
    }
}
hck<-hck*-1
hkc<-hkc*-1
hk<-hk*-1
# Calculate Homogenity and Completeness
h<- 1-hck/h_c
c<- 1- hkc/hk
print(h)
print(c)
```

```
[1] 0.8518851
[1] 0.3298739
```

[3]:
```
circlefile <- readLines("gplus/115625564993990145546.circles")
circles = list()

#Obtain circles
for (j in 1:length(circlefile)) {
    circle_nodes = strsplit(circlefile[j],"\t")
    circles = c(circles, list(circle_nodes[[1]][-1]))
}

#Calculate N
all_circle <- c()
for (l in circles){
    all_circle <- c(all_circle,l)
}
all_circle <- unique(all_circle)
N <- length(all_circle)

#Calculate H(C)
h_c <- 0
for (l in circles){
    h_c<- h_c+ ((length(l)/N)* log10(length(l)/N))
}
h_c<-h_c*-1
```

```
#Extract Community Structures
g1<-read_graph("gplus/115625564993990145546.edges",format="ncol",directed=TRUE)
g1 = add.vertices(g1, nv = 1, name = "115625564993990145546")
index = which(V(g1)$name=="115625564993990145546")
el = c()
for (vertex in 1:(vcount(g1) - 1)) {
    el = c(el, c(index, vertex))
}
g1 = add_edges(g1, el)
fg <- walktrap.community(g1)

#Find H(K),H(C|K) and H(K|C).
percentage = vector()
percentage_circle = vector()
hck<-0
hkc<-0
hk<-0
for(m in 1:max(fg$membership)){
    community_nodes = V(g1)$name[which(fg$membership == m)]
    bi<-length(intersect(community_nodes,all_circle))
    if(bi!=0) {
    hk<- hk + ((bi/N) * log10(bi/N))
    for (n in 1:length(circles)) {
      common_nodes = intersect(community_nodes, circles[[n]])
        cij<-length(unique(common_nodes))
        ai<-length(circles[[n]])
        if(cij!=0){
            hck<-hck+ (cij/N)*log10(cij/bi)
            hkc <- hkc + (cij/N)*log10(cij/ai)
            }
        }
    }
}
hck<-hck*-1
hkc<-hkc*-1
hk<-hk*-1
# Calculate Homogenity and Completeness
h<- 1-hck/h_c
c<- 1- hkc/hk
print(h)
print(c)
```

```
[1] 0.4518903
[1] -3.423962
```

[4]:
```
circlefile <- readLines("gplus/101373961279443806744.circles")
circles = list()

#Obtain circles
for (j in 1:length(circlefile)) {
    circle_nodes = strsplit(circlefile[j],"\t")
    circles = c(circles, list(circle_nodes[[1]][-1]))
}

#Calculate N
all_circle <- c()
for (l in circles){
    all_circle <- c(all_circle,l)
}
all_circle <- unique(all_circle)
N <- length(all_circle)

#Calculate H(C)
h_c <- 0
for (l in circles){
    h_c<- h_c+ ((length(l)/N)* log10(length(l)/N))
}
h_c<-h_c*-1

#Extract Community Structures
g1<-read_graph("gplus/101373961279443806744.edges",format="ncol",directed=TRUE)
g1 = add.vertices(g1, nv = 1, name = "101373961279443806744")
index = which(V(g1)$name=="101373961279443806744")
el = c()
for (vertex in 1:(vcount(g1) - 1)) {
    el = c(el, c(index, vertex))
}
g1 = add_edges(g1, el)
fg <- walktrap.community(g1)

#Find H(K),H(C|K) and H(K|C).
percentage = vector()
percentage_circle = vector()
hck<-0
hkc<-0
hk<-0
for(m in 1:max(fg$membership)){
    community_nodes = V(g1)$name[which(fg$membership == m)]
    bi<-length(intersect(community_nodes,all_circle))
    if(bi!=0) {
    hk<- hk + ((bi/N) * log10(bi/N))
    for (n in 1:length(circles)) {
```

```
      common_nodes = intersect(community_nodes, circles[[n]])
        cij<-length(unique(common_nodes))
        ai<-length(circles[[n]])
        if(cij!=0){
            hck<-hck+ (cij/N)*log10(cij/bi)
            hkc <- hkc + (cij/N)*log10(cij/ai)
            }
        }
    }
}
hck<-hck*-1
hkc<-hkc*-1
hk<-hk*-1
# Calculate Homogenity and Completeness
h<- 1-hck/h_c
c<- 1- hkc/hk
print(h)
print(c)
```

```
[1] 0.003866707
[1] -1.504238
```