

CS4907/CS6444 Big Data and Analytics  
Class project #1 Report  
Spring 2025

Group 4  
Tianfang Fang, Rohith Vijayakumar

## 1. Data Set: email-EU (on Blackboard)

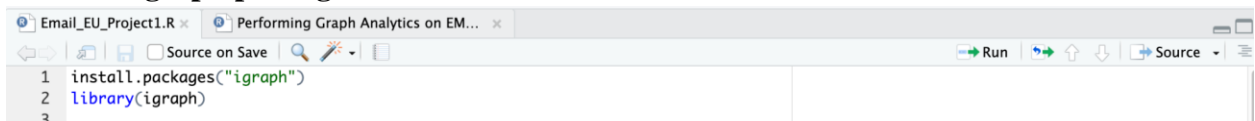
**Dataset information:** Email-EU is a network representing emails transmitted between EU entities. The data is represented as pairs consisting of FromNode ToNode. All nodes are represented by the nodeID- an integer.

Dataset statistics	
Nodes	32.4K
Edges	54.4K
Density	0.000103449
Maximum degree	623
Minimum degree	1
Average degree	3
<a href="#">Assortativity</a>	-0.381627
Number of triangles	147K
Average number of triangles	4
Maximum number of triangles	1.6K
Average clustering coefficient	0.112681

### Source (citation)

J. Leskovec, J. Kleinberg and C. Faloutsos. [Graph Evolution: Densification and Shrinking Diameters](#). ACM Transactions on Knowledge Discovery from Data (ACM TKDD), 1(1), 2007.des.

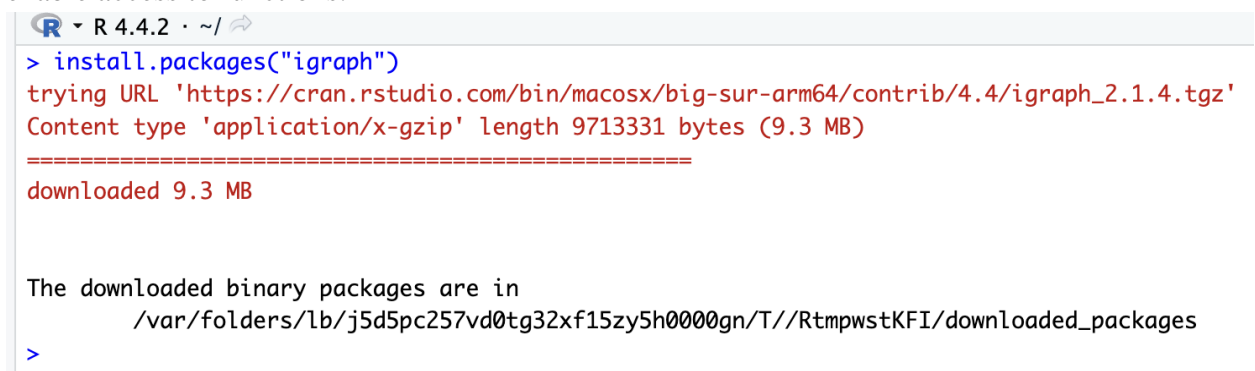
## 2. Install the igraph package from one of the CRAN mirrors.



```
1 install.packages("igraph")
2 library(igraph)
3
```

Above is the code used to install the “igraph” package from one of the CRAN mirrors.

Following are console outputs obtained after installing and declaring “igraph”, as a library to enable access to functions.



```
R 4.4.2 · ~/
> install.packages("igraph")
trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.4/igraph_2.1.4.tgz'
Content type 'application/x-gzip' length 9713331 bytes (9.3 MB)
=====
downloaded 9.3 MB

The downloaded binary packages are in
  /var/folders/lb/j5d5pc257vd0tg32xf15zy5h0000gn/T//RtmpwstKFI/downloaded_packages
>
```

```
> library(igraph)
```

Attaching package: 'igraph'

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

decompose, spectrum

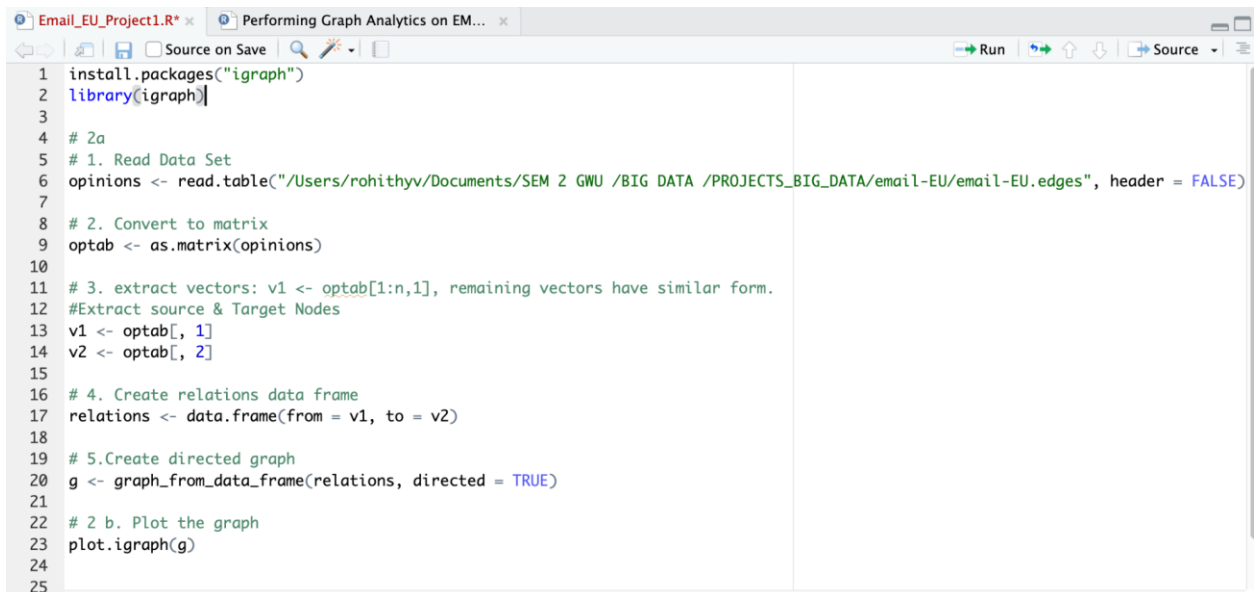
The following object is masked from 'package:base':

union

### a. Import the specified data set

The dataset is loaded after ensuring that the file exists in the current working directory. The data is loaded via the function **read.table** and stored in the variable **opinions**. The data is converted to a matrix and vectors are extracted as v1, v2, and v3, here v3 contains additional information for each edge and is treated as weights.

A data frame is procured from the vectors by use of the function **graph\_from\_data\_frame()**. The following code snippet and the corresponding output are shown below:  
The graph obtained from using the **plot** function is as below, and can be described as a blob. Iterating through the entire graph proves to be costly, resulting in high execution time and an indiscernable graph, despite tweaking parameters to reduce vertex size and edge arrow size.



```
1 install.packages("igraph")
2 library(igraph)
3
4 # 2a
5 # 1. Read Data Set
6 opinions <- read.table("/Users/rohithyv/Documents/SEM 2 GWU /BIG DATA /PROJECTS_BIG_DATA/email-EU/email-EU.edges", header = FALSE)
7
8 # 2. Convert to matrix
9 optab <- as.matrix(opinions)
10
11 # 3. extract vectors: v1 <- optab[1:n,1], remaining vectors have similar form.
12 #Extract source & Target Nodes
13 v1 <- optab[, 1]
14 v2 <- optab[, 2]
15
16 # 4. Create relations data frame
17 relations <- data.frame(from = v1, to = v2)
18
19 # 5.Create directed graph
20 g <- graph_from_data_frame(relations, directed = TRUE)
21
22 # 2 b. Plot the graph
23 plot.igraph(g)
24
25
```

### Procedure:

**1. opinions<-read.table(<<data set>>) assuming you have set the working directory**

# Read Data Set

```
opinions <- read.table("/Users/rohithyv/Documents/SEM 2 GWU /BIG DATA /PROJECTS_BIG_DATA/email-EU/email-EU.edges", header = FALSE)
```

```
4 # 2a
5 # 1. Read Data Set
6 opinions <- read.table("/Users/rohithyv/Documents/SEM 2 GWU /BIG DATA /PROJECTS_BIG_DATA/email-EU/email-EU.edges", header = FALSE)
7
```

**2. convert to matrix: optab <-as.matrix(opinions)**

```
optab <- as.matrix(opinions)
```

```
8 # 2. Convert to matrix
9 optab <- as.matrix(opinions)
```

**3. extract vectors: v1 <- optab[1:n,1], remaining vectors have similar form. This gets the vertices as separate vectors**

#Extract source & Target Nodes

```
v1 <- optab[, 1]
v2 <- optab[, 2]
11 # 3. extract vectors: v1 <- optab[1:n,1], remaining vectors have similar form.
12 #Extract source & Target Nodes
13 v1 <- optab[, 1]
14 v2 <- optab[, 2]
```

**4. relations<- data.frame(from=v1,to=v2)**

#Create relations data frame

```
relations <- data.frame(from = v1, to = v2)
16 # 4. Create relations data frame
17 relations <- data.frame(from = v1, to = v2)
```

**5. g<-graph.data.frame(relations,directed=TRUE) need to have installed igraph**

#Create directed graph

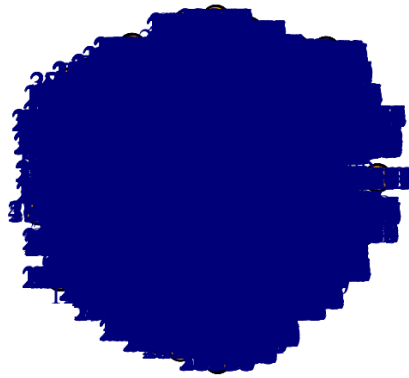
```
g <- graph_from_data_frame(relations, directed = TRUE)
19 # 5.Create directed graph
20 g <- graph_from_data_frame(relations, directed = TRUE)
21
```

**b. Determine how to create a graph and plot. Show the plot in your report.**

#Plot the graph

```
plot.igraph(g)
```

```
23 plot.igraph(g)
24
```



**Simplified Email Network**

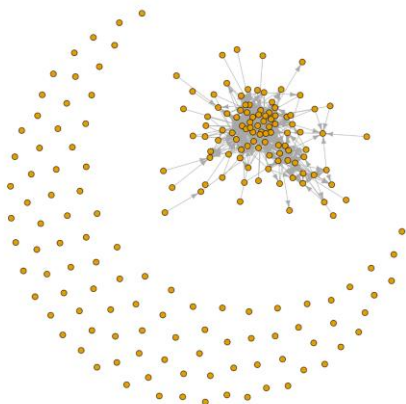


This data is incomprehensible, the vertex numbers and the vertex size take up majority of the space thus resulting in a larger blob. We can attempt to understand the plot by looking at sections of the node list. Below are the plots of the first 200, 500, 1000, 2500 and last 200, 500, 1000, and 2500 nodes.

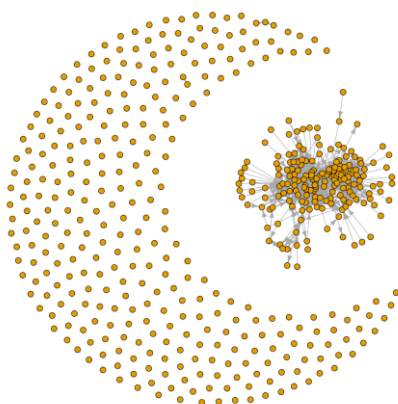
**First 200 nodes**

**First 500 nodes**

First 200 Nodes

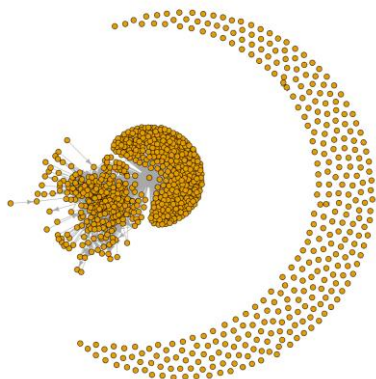


First 500 Nodes



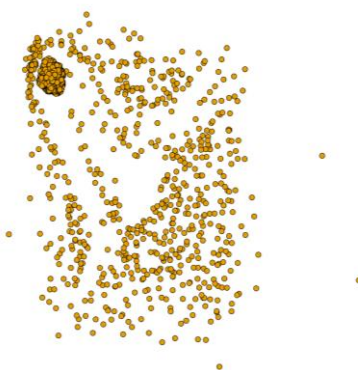
First 1000 nodes

First 1000 Nodes



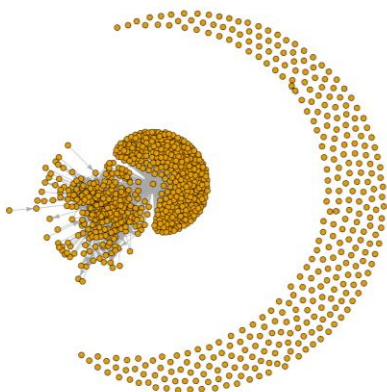
First 2500 nodes

First 2500 Nodes



First 10000 nodes

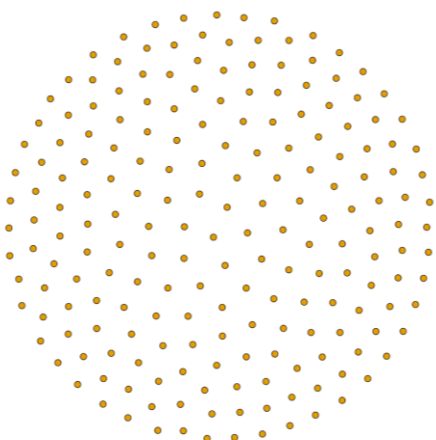
First 1000 Nodes



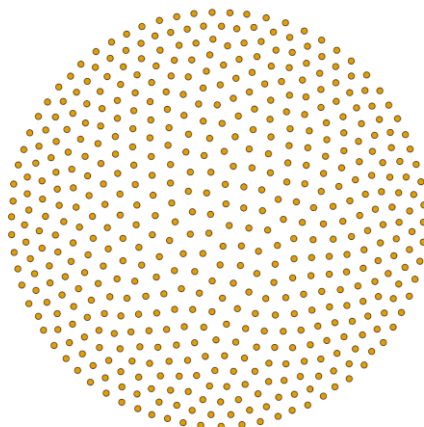
Last 200 nodes

Last 500 nodes

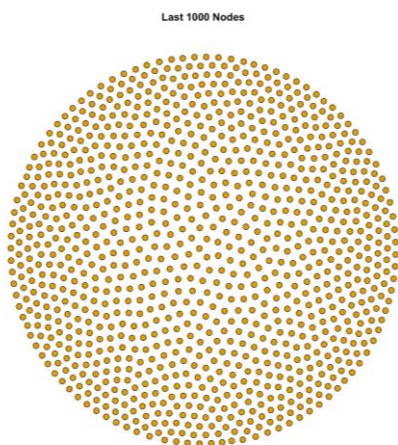
Last 200 Nodes



Last 500 Nodes

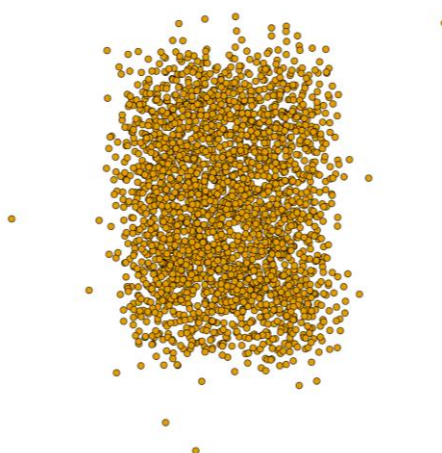


Last 1000 nodes



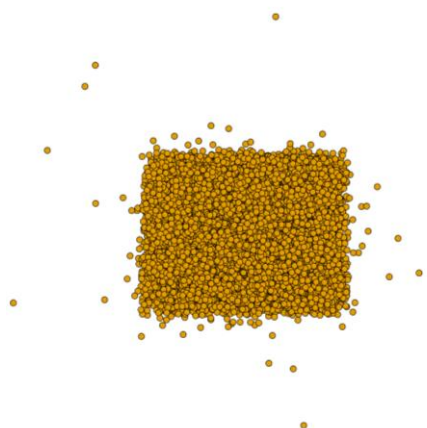
Last 2500 nodes

Last 2500 Nodes



Last 10000 nodes

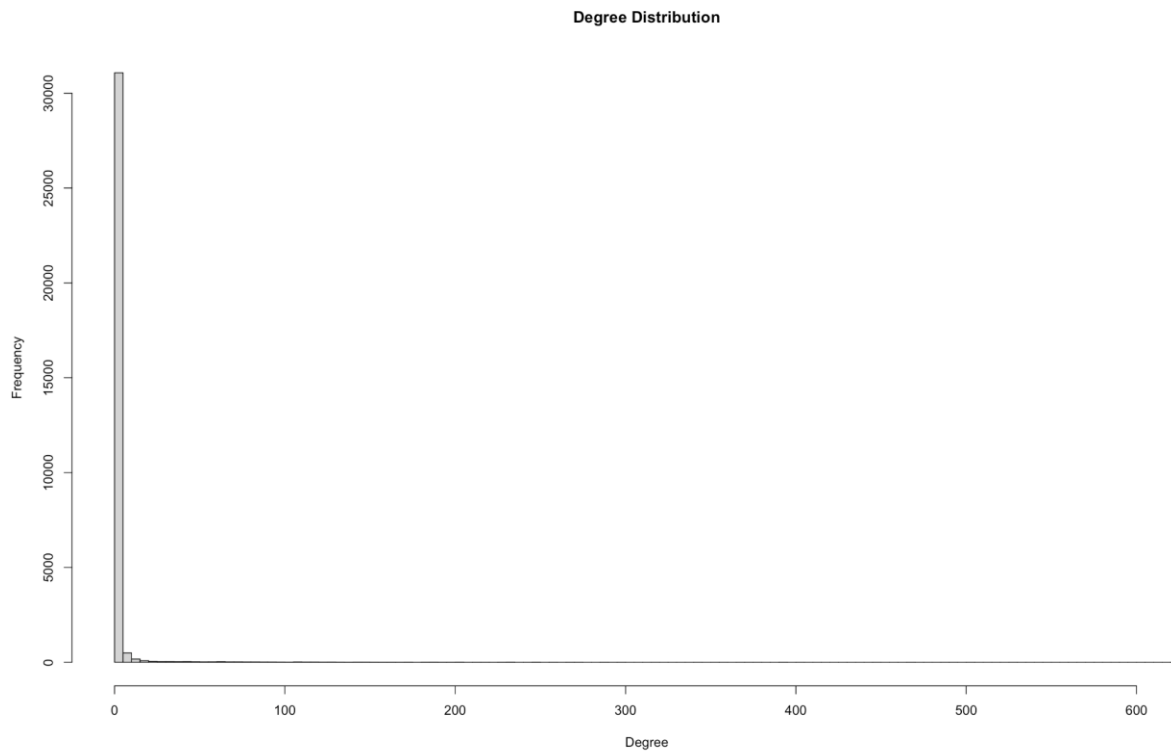
Last 10000 Nodes



**3. Apply the functions that I have shown in the *Introduction to Graph Analytics* document on Blackboard on the graph generated from the data set.**

**a. Present the results in your write-up as screen shots.**

### 3a. Degree Distribution



### 3b. Average Path Length

```
> # 3b. Average Path Length
> avg_path_length <- average.path.length(g_simplified)
> print(paste("Average Path Length:", avg_path_length))
[1] "Average Path Length: 3.76927675013719"
> # Interpretation: Average distance between any two nodes.
> |
```

### 3c. Clustering Coefficient

```
> # 3c. Clustering Coefficient
> clustering_coeff <- transitivity(g_simplified, type = "average")
> print(paste("Average Clustering Coefficient:", clustering_coeff))
[1] "Average Clustering Coefficient: 0.52091748475565"
> # Interpretation: Measures how interconnected a node's neighbors are.
```

### 3d. Diameter



```

> # 3d. Diameter
> graph_diameter <- diameter(g_simplified)
> print(paste("Diameter:", graph_diameter))
[1] "Diameter: 13"
> # Interpretation: Longest shortest path in the network.

```

### 3e. Connected Components

```

> # 3e. Connected Components
> components_result <- components(g_simplified)
> print(paste("Number of Connected Components:", components_result$no))
[1] "Number of Connected Components: 1"
> print(paste("Size of Largest Component:", max(components_result$size)))
[1] "Size of Largest Component: 32430"
> # Interpretation: Shows if the graph is fully connected or has separate parts.

```

### 3f. Density

```

> # 3f. Density
> graph_density <- edge_density(g_simplified)
> print(paste("Graph Density:", graph_density))
[1] "Graph Density: 5.1724278757625e-05"
> # Interpretation: The proportion of potential edges that are actual edges. Low density indicates a sparse network.

```

### 3g. Edge Density

```

> # 3g. Edge Density
> edge_density_simplified <- edge_density(g_simplified)
> print(paste("Edge Density:", edge_density_simplified))
[1] "Edge Density: 5.1724278757625e-05"
> # Interpretation: Same as graph density.

```

### 3h. Node Density (often refers to graph density)

```

> # 3h. Node Density (often refers to graph density)
> node_density <- vcount(g_simplified) / (vcount(g_simplified) * (vcount(g_simplified)-1) / 2) #for directed graph
> print(paste("Node Density:", node_density))
[1] "Node Density: 6.16731937463382e-05"
> # Interpretation: The same as graph density.

```

## **b. Explain what the function could tell you about the problem domain.**

### 3a. Degree Distribution

Degree shows the number of connections a node has(email sent and received in this case). This function tells us the distribution of degrees, and can help identify highly connected nodes(users).

### 3b. Average Path Length

The average shortest path is around 3.7 which suggests that most users can reach each other in 4 steps, and this is an efficient communication structure.

### 3c. Clustering Coefficient

Clustering coefficient measures how interconnected a node's neighbors are. 0.52 means there is a 52% chance that B and C email each other if A emails both B and C.

### 3d. Diameter

The diameter indicated the longest shortest path between any two nodes. In this case meaning it takes 13 steps for an email to reach from one user to another.

### 3e. Connected Components

There is only one connected component which means all users are part of the same communication network. A single component means that eventually information can reach everyone.

### 3f. Density

Density represents the proportion of potential edges that are actual edges. In this case a very low density indicates that the network is sparse.

### 3g. Edge Density

Edge density is identical to graph density and it confirms the sparsity of the email network.

### 3h. Node Density (often refers to graph density)

Node density is another way to measure connectivity, the very low value confirms that a tiny fraction of all possible connections exist.

## **c. As an example what do node density and edge density indicate about the problem space?**

As the datasets represent an email communication network, the node density and edge density help describe how well-connected the network is and whether it exhibits social network properties in real world such as sparsity, hubs, and clustering.

Node density is a measure of how well the nodes in a network are connected compared to the maximum possible number of connections. A low node density which close to 0 indicates that the network is sparse with a small fraction of possible connections existing, and it is expected in email and social networks. A high node density which close to 1 indicates that almost all nodes are connected to each other, and it is expected in small and close-knit communities such as family groups.

Edge density is a measure of how many actual connections(edges) exist in a network compared to the maximum possible number of connections. A low edge density which close to 0 indicates that the network is sparse, and expected in email and social networks. A high edge density which close to 1 indicates that the network is dense with most nodes are interconnected, and it happens in small and close-knit communities such as small groups.

#### **4. Determine the (a) central nodes(s) in the graph, (b) longest path(s), (c) largest clique(s), (d) ego(s), and (e) power centrality.**

##### 4a. Central Nodes

```
> # 4a. Central Nodes (Several options - choose one or more)
>
> # Degree Centrality (number of connections)
> degree_centrality <- degree(g_simplified)
> central_node_degree <- which.max(degree_centrality) # Node with highest degree
> print(paste("Central Node (Highest Degree):", central_node_degree))
[1] "Central Node (Highest Degree): 467"
```

##### 4b. Longest Path (Diameter)

```
> # 4b. Longest Path (Diameter)
> graph_diameter <- diameter(g_simplified) # This returns the *length* of the diameter
> print(paste("Diameter (Length):", graph_diameter))
[1] "Diameter (Length): 13"
>
> # To get the actual path (nodes involved):
> diameter_path <- get_diameter(g_simplified) # Returns an edge sequence
> print("Diameter Path (Edges):")
[1] "Diameter Path (Edges):"
> print(diameter_path)
+ 14/32430 vertices, named, from fff84a8:
[1] 17763 16731 8855 3759 2136 1867 1501 1500 1380 1374 1229 1217 999 938
>
> # To get the nodes involved in the diameter path:
> diameter_nodes <- V(g_simplified)[get_diameter(g_simplified, directed=FALSE)]
> print("Diameter Path (Nodes):")
[1] "Diameter Path (Nodes):"
> print(diameter_nodes)
+ 10/32430 vertices, named, from fff84a8:
[1] 1557 113 17829 102 95 2136 3759 8855 16731 17763
```

##### 4c. Largest Clique(s)

```

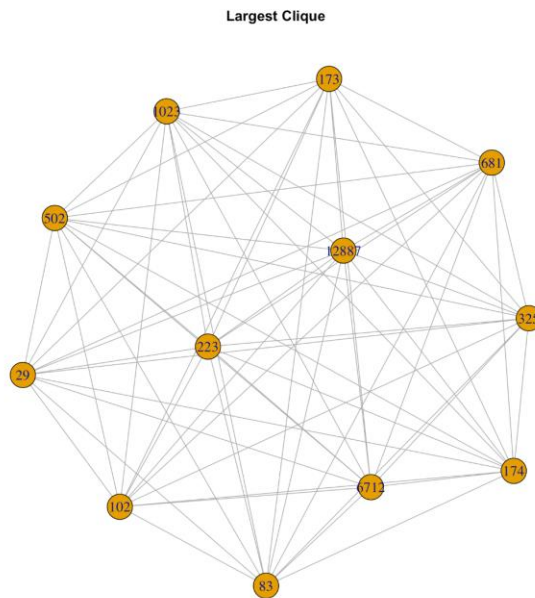
> # 4c. Largest Clique(s)
> # Convert the directed graph to an undirected graph
> g_undirected <- as.undirected(g_simplified, mode = "collapse")
> # Find the largest cliques
> largest_cliques_result <- largest_cliques(g_undirected)
>
> print(paste("Largest Clique(s):", largest_cliques_result))
[1] "Largest Clique(s): c(`6712` = 74, `29` = 2, `12887` = 104, `83` = 7, `102` = 467, `1023` = 40, `174` = 1
6, `502` = 28, `325` = 23, `223` = 20, `681` = 2228, `173` = 15)"
[2] "Largest Clique(s): c(`223` = 20, `174` = 16, `325` = 23, `502` = 28, `83` = 7, `82` = 6, `102` = 467, `1
2887` = 104, `1023` = 40, `1022` = 39, `29` = 2, `8013` = 83)"
[3] "Largest Clique(s): c(`223` = 20, `174` = 16, `325` = 23, `502` = 28, `83` = 7, `82` = 6, `102` = 467, `1
2887` = 104, `1023` = 40, `1022` = 39, `29` = 2, `173` = 15)"
[4] "Largest Clique(s): c(`223` = 20, `174` = 16, `325` = 23, `502` = 28, `83` = 7, `82` = 6, `1` = 32112, `1
2887` = 104, `29` = 2, `1022` = 39, `1023` = 40, `8013` = 83)"
[5] "Largest Clique(s): c(`223` = 20, `174` = 16, `325` = 23, `502` = 28, `83` = 7, `82` = 6, `1` = 32112, `1
2887` = 104, `29` = 2, `1022` = 39, `1023` = 40, `173` = 15)"
[6] "Largest Clique(s): c(`223` = 20, `174` = 16, `325` = 23, `502` = 28, `83` = 7, `681` = 2228, `102` = 46
7, `12887` = 104, `1023` = 40, `29` = 2, `1022` = 39, `173` = 15)"
[7] "Largest Clique(s): c(`52` = 4, `174` = 16, `214` = 19, `83` = 7, `82` = 6, `502` = 28, `173` = 15, `102`
= 467, `87` = 9, `355` = 25, `453` = 2001, `26` = 1)"
[8] "Largest Clique(s): c(`52` = 4, `174` = 16, `214` = 19, `83` = 7, `82` = 6, `502` = 28, `173` = 15, `102`
= 467, `87` = 9, `355` = 25, `453` = 2001, `279` = 3520)"
[9] "Largest Clique(s): c(`8013` = 83, `82` = 6, `502` = 28, `102` = 467, `1023` = 40, `214` = 19, `29` = 2,
`174` = 16, `83` = 7, `12887` = 104, `1022` = 39, `87` = 9)"
[10] "Largest Clique(s): c(`8013` = 83, `82` = 6, `502` = 28, `102` = 467, `1023` = 40, `214` = 19, `29` = 2,
`174` = 16, `83` = 7, `12887` = 104, `1022` = 39, `325` = 23)"
[11] "Largest Clique(s): c(`8013` = 83, `82` = 6, `502` = 28, `1` = 32112, `214` = 19, `174` = 16, `83` = 7, `
29` = 2, `1023` = 40, `12887` = 104, `1022` = 39, `87` = 9)"
[12] "Largest Clique(s): c(`8013` = 83, `82` = 6, `502` = 28, `1` = 32112, `214` = 19, `174` = 16, `83` = 7, `
29` = 2, `1023` = 40, `12887` = 104, `1022` = 39, `325` = 23)"
[13] "Largest Clique(s): c(`559` = 506, `174` = 16, `102` = 467, `214` = 19, `681` = 2228, `12887` = 104, `102
3` = 40, `29` = 2, `325` = 23, `1022` = 39, `502` = 28, `173` = 15)"
[14] "Largest Clique(s): c(`103` = 468, `214` = 19, `279` = 3520, `174` = 16, `87` = 9, `82` = 6, `83` = 7, `3
55` = 25, `95` = 11, `502` = 28, `1023` = 40, `102` = 467)"
[15] "Largest Clique(s): c(`103` = 468, `214` = 19, `279` = 3520, `174` = 16, `87` = 9, `82` = 6, `83` = 7, `3
55` = 25, `95` = 11, `502` = 28, `453` = 2001, `102` = 467)"
[16] "Largest Clique(s): c(`103` = 468, `214` = 19, `279` = 3520, `174` = 16, `87` = 9, `82` = 6, `83` = 7, `3
55` = 25, `95` = 11, `502` = 28, `453` = 2001, `950` = 37)"
[17] "Largest Clique(s): c(`103` = 468, `214` = 19, `279` = 3520, `174` = 16, `87` = 9, `82` = 6, `83` = 7, `3
55` = 25, `1022` = 39, `1023` = 40, `502` = 28, `102` = 467)"
[18] "Largest Clique(s): c(`102` = 467, `214` = 19, `174` = 16, `83` = 7, `1023` = 40, `502` = 28, `173` = 15,
`82` = 6, `1022` = 39, `325` = 23, `12887` = 104, `29` = 2)"
[19] "Largest Clique(s): c(`102` = 467, `214` = 19, `174` = 16, `83` = 7, `1023` = 40, `502` = 28, `173` = 15,
`82` = 6, `1022` = 39, `87` = 9, `355` = 25, `279` = 3520)"
[20] "Largest Clique(s): c(`102` = 467, `214` = 19, `174` = 16, `83` = 7, `1023` = 40, `502` = 28, `173` = 15,
`82` = 6, `1022` = 39, `87` = 9, `12887` = 104, `29` = 2)"
[21] "Largest Clique(s): c(`102` = 467, `214` = 19, `174` = 16, `83` = 7, `1023` = 40, `502` = 28, `173` = 15,
`681` = 2228, `325` = 23, `1022` = 39, `29` = 2, `12887` = 104)"
[22] "Largest Clique(s): c(`95` = 11, `82` = 6, `174` = 16, `83` = 7, `888` = 35, `214` = 19, `87` = 9, `453`
= 2001, `279` = 3520, `950` = 37, `502` = 28, `355` = 25)"
[23] "Largest Clique(s): c(`173` = 15, `174` = 16, `214` = 19, `83` = 7, `82` = 6, `87` = 9, `502` = 28, `1`
= 32112, `1023` = 40, `1022` = 39, `12887` = 104, `29` = 2)"
[24] "Largest Clique(s): c(`173` = 15, `174` = 16, `214` = 19, `83` = 7, `82` = 6, `325` = 23, `502` = 28, `1`
= 32112, `1022` = 39, `1023` = 40, `29` = 2, `12887` = 104)"

```

```

> largest_clique_nodes <- V(g_simplified)[largest_cliques_result[[1]]] # [[1]] gets the first (largest) clique
> print("Nodes in Largest Clique:")
[1] "Nodes in Largest Clique:"
> print(largest_clique_nodes)
+ 12/32430 vertices, named, from fff84a8:
[1] 6712 29 12887 83 102 1023 174 502 325 223 681 173
> largest_clique_size <- max(sapply(largest_cliques_result, length))
> print(largest_clique_size)
[1] 12
> # Extract the first largest clique
> clique_nodes <- largest_cliques_result[[1]]
>
> # Create a subgraph of just this clique
> clique_subgraph <- induced_subgraph(g_undirected, clique_nodes)
>
> # Plot the largest clique
> plot(clique_subgraph, vertex.size = 10, vertex.label.cex = 1.2, edge.arrow.size = 0.5, main = "Largest Clique")
>

```

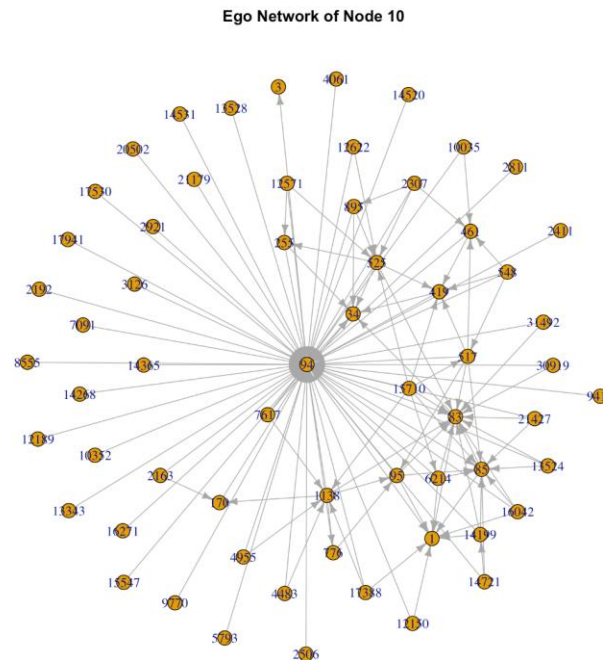


#### 4d. Ego Network (example: ego network of node 10 - change as needed)

```

> # 4d. Ego Network (example: ego network of node 10 - change as needed)
> # Get the ego network for node 10 (ego returns a list of vertex sets)
> ego_nodes <- ego(g_simplified, order = 1, nodes = V(g_simplified)[10])[[1]]
>
> # Create an induced subgraph from the ego network
> ego_net <- induced_subgraph(g_simplified, ego_nodes)
>
> # Plot the ego network
> plot(ego_net, vertex.size = 5, vertex.label.cex = 1, edge.arrow.size = 0.3, main = "Ego Network of Node 10")
>

```



#### 4e. Power Centrality (Eigenvector Centrality)

```
> # 4e. Power Centrality (Eigenvector Centrality)
> power_cent <- eigen_centrality(g_simplified)$vector
> most_powerful_node <- which.max(power_cent)
> print(paste("Most Powerful Node (Eigenvector Centrality):", most_powerful_node))
[1] "Most Powerful Node (Eigenvector Centrality): 500"
```

## Discussion

Throughout this project, we learnt how to use R studio to plot data and process data analysis with the email communication network datasets. Firstly, we loaded and preprocessed the data and we found out the dataset is consist of email exchanges between users with each node represents an email address, each edge represents an email sent from one user to another, and the graph is directed meaning email direction matters( A to B is different from B to A). We simplified the graph by removing multiple edges and loops so self-loop and duplicate emails are removed.