** **

**COMPUTER SCIENCE AND DATA ANALYTICS**

**Course: CSCI 6444 Intro to Big Data Analytics**

**Class project #1**

**R and Graph Analytics**

**Student(s): Group 6  
Anar Shikhaliyev  
Eljan Mammadli**

**Instructor: Dr. Abzatdin Adamov**

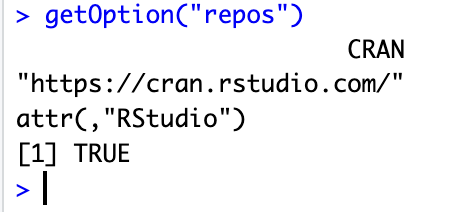
Baku 2023

1. **Data Set**

In this assignment, we have a network of all the incoming and outgoing email of the research institution communication between Europian countries. We have 3,038,531 emails in total, distributed among 287,755 different email accounts. We only have a complete email graph for 1,258 of the research institution's email addresses. Also, within the dataset's time frame, 34,203 email addresses sent and received email. All other email addresses are either invalid, typographically incorrect, or spam.

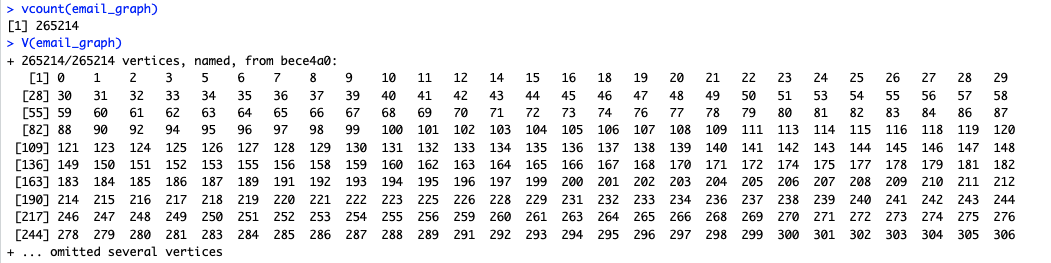
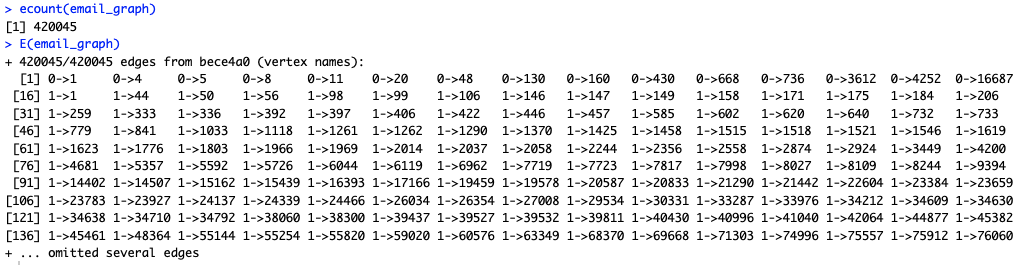
**2. Install the igraph package from one of the CRAN mirrors**

In order to install the igraph, we need to follow these steps:

1. Open RStudio
2. Type following command in the R console: install.packages("igraph")  
    Picture 1. Installing igraph
3. R did not prompt any CRAN mirror selection because, it installed packages from default CRAN mirror in R configuration which we can access it by:  
   

**3. Experiment with some of the functions that shown in the Introduction to Graph Analytics document on Blackboard on the graph generated from the data set. Present the results in your write-up.**

We have experimented this R functions on the given dataset:

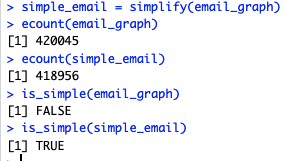
* Vcount(), V() and ecount(), E() - These functions help to gain information about the vectors and edges of the graph respectively.  
    
  
* Density - proportion between the number of edges and the number of potential edges is known as a graph's density.



* Degree - shows the degree of every node inside the graph. Since our data is too big we will not show the actual output.

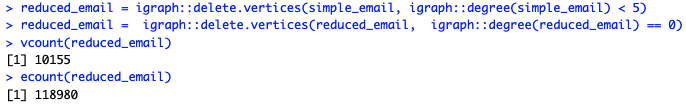
Since our data is too large some of the commands were taking too much time and resource to execute. For that reason, it might be helpful to simplify the graph when working with big networks like ours to minimize the number of nodes and edges, making it easier to manipulate. So, we need to simplify the graph in order to execute those commands.

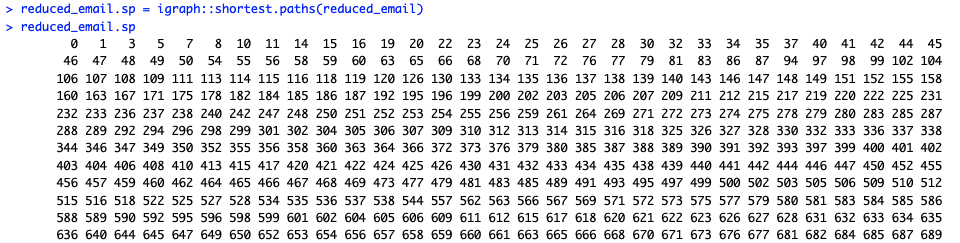
* Simplify and is\_simple - eliminates the loop as well as several edges from a graph. As we can see from the output below, there is a slight decrease in number of edges after removing loops and it is verified by the is\_simple function.



But doing only this was not enough to simplify the graph, we needed to implement other simplification tactics.

First technic we used is *Node degree thresholding*. With this method we eliminated nodes with a low degree, considering they have less connections and therefore not crucial to the network's overall structure. The minimal number of edges a node must possess in order to be included in the condensed graph can be specified as a threshold (which is **5** in our case). After deleting nodes with less than five it is likely to have some nodes with zero degree, thus we eliminated them as well.

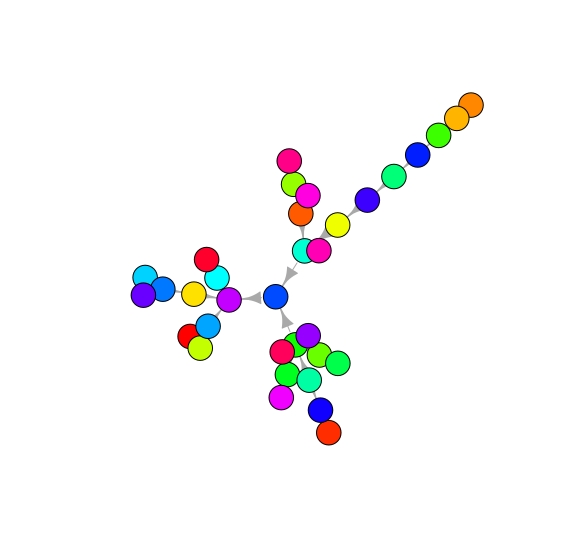


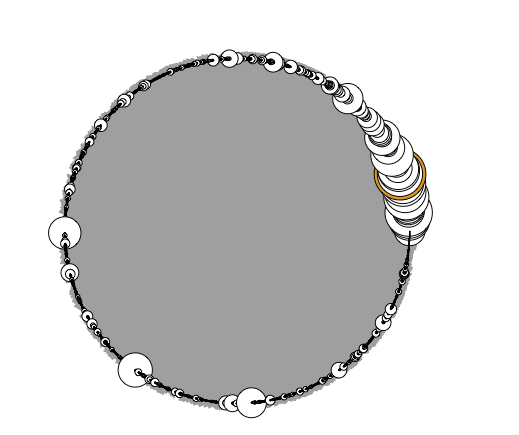
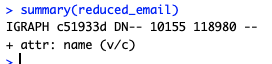
* shortest.paths()  
  The shortest pathways between each pair of vertices in a graph are determined using the igraph::shortest.paths() method. The shortest path between a source vertex and every other vertex in the graph is determined using Dijkstra's method.  
  

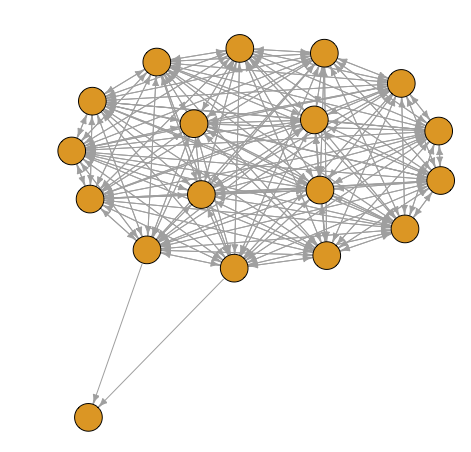
**4. Explore other functions in the igraph package**

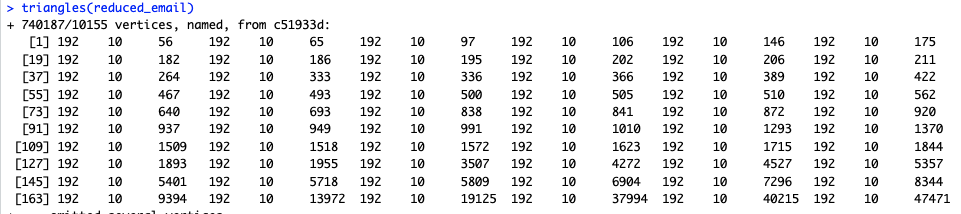
* mst()

Minimal Spanning Tree connects all the vertices of the graph with the minimum possible total edge weight. Interpretation can vary depending on the context such as identifying most important (central nodes), efficient paths between pairs of vertices, and even identfying communities.

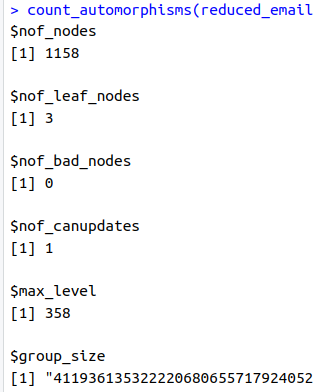


* eigenvector\_centrality()  
  A measure of a vertex's relevance in a network called eigenvector centrality and considers both the vertex's connectivity to other nodes and the significance of the nodes to which those connections are made.  
    
  
* summary()   
  This function offers a summary of the graph, including its type, size, number of vertices and edges, and any properties.  
  
* Cluster\_louvain (explained in more detail at 5.f)
* Layout\_with\_kk  
  This function from the igraph package uses the Kamada-Kawai technique to determine a graph's layout. By placing vertices in places that minimize the sum of the spring and electrical forces connecting them, the Kamada-Kawai algorithm, a force-directed graph layout technique, seeks to reduce the overall energy of the network.  
  

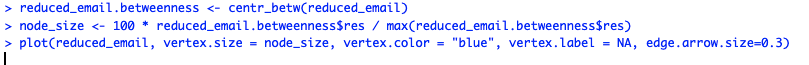
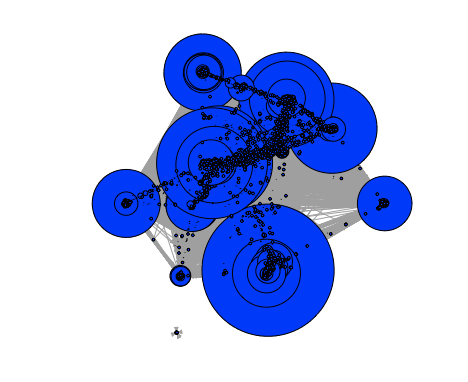
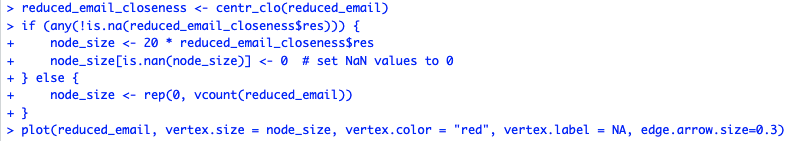
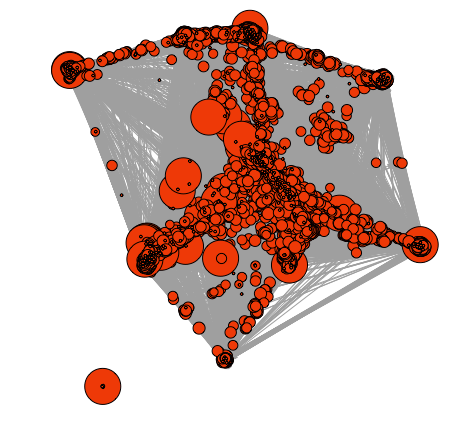
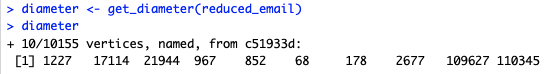
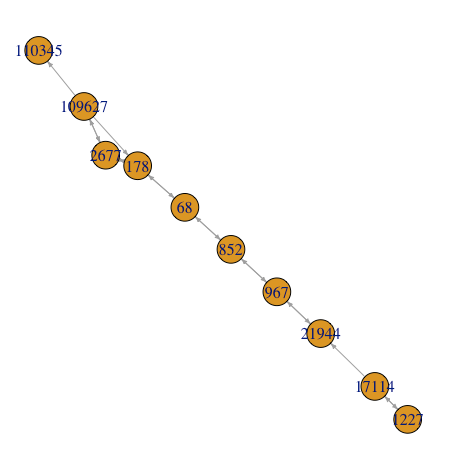
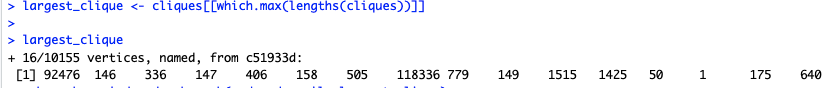
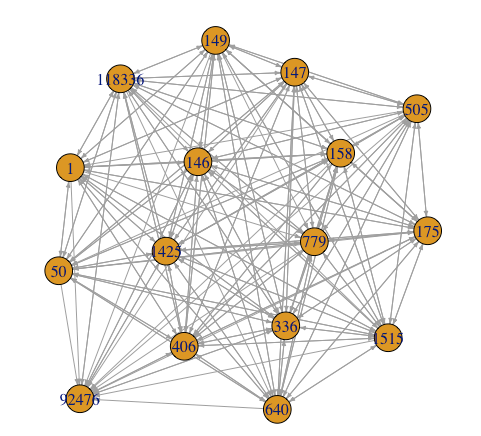


* Transitivity  
  This function determines a graph's transitivity, which is the proportion of triangles to linked triples of vertices in the graph.  
  
* Triangles  
  This function determines a graph's triangle count, which is a gauge of the graph's clustering coefficient.  
  
* Isomorphic  
  This function determines if two graphs are isomorphic, which implies they share the same structure but may have distinct vertex and edge names. Output below is clearly false, because we eliminated some connections, resulting changing the structure.  
  
* Mean\_distance  
  The term "mean distance" describes the shortest path's average length between every pair of vertices in a network. The smallest number of edges that must be crossed to get from one vertex to another is known as the shortest path length.  
  
* count\_automorphisms()

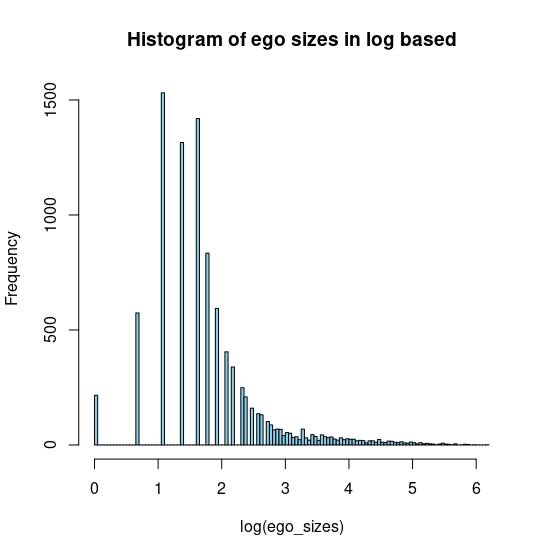
This is a method to find the number of automorphisms in a graph, which are isomorphisms from the graph to itself. An automorphism of a graph is a permutation of its vertices that preserves its edges. Isomorphisms means, two graphs being structurally equivalent. It seems that there is a high degree of symmetry in the network.



**5. Determine the (a) central nodes(s) in the graph, (b) longest path(s), (c) largest clique(s), (d) ego(s), (e) power centrality, (f) find communities.**

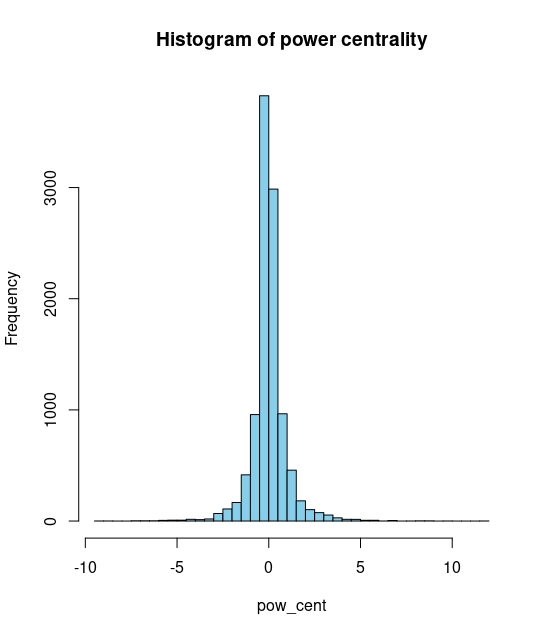
* central nodes(s) in the graph  
  There are several methods for calculating centrality in a network, and various centrality metrics can draw attention to various kinds of significant nodes.
  1. Betweenness centrality  
     This calculates how far a node is located along the network's shortest pathways to other nodes. Bridges connecting various regions of the network are built by nodes with high betweenness centrality.  
     In order to visualize betweenness, we can filter nodes where size is proportional to the corresponding centrality score:  
     
  2. Closeness centrality determined by taking the reciprocal of the lengths of the shortest routes that connect the node to every other node in the graph. As in betweenness centrality, we filter nodes by size as a measure of the importance of each node based on its closeness centrality, where larger nodes indicate greater importance.  
       
     
* longest path  
  In order to find the longest path of the graph, we can calculates the diameter of the graph, which is the length of the longest shortest path in the graph.  
    
  
* largest clique   
  This function returns all the maximal cliques in the graph, and we can select the largest one using the which.max function. After finding the largest clique, we can visualize it as subgraph.  
    
  
* (d) ego(s),

From the histogram below it is visible that it is right skewed which means generality of the nodes in the network have ego sizes and only among 10536 nodes 150 of them has ego sizes more than 100. The mean, median, maximum values are 11.2, 5, 479 respectively.



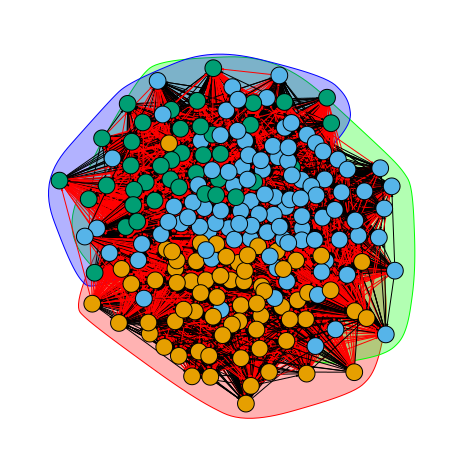
* (e) power centrality,

It is measure of the influence of nodes in the network using the connections of them. In this network we can clearly see from distribution that majority of the the values in the power centrality lies around zero which are very small number. This can indicate that few of the nodes in the network play crucial influence in the organization while lots of them do not have reasonable effect. The maximum value is 11.9 while median is 0.



* (f) find communities

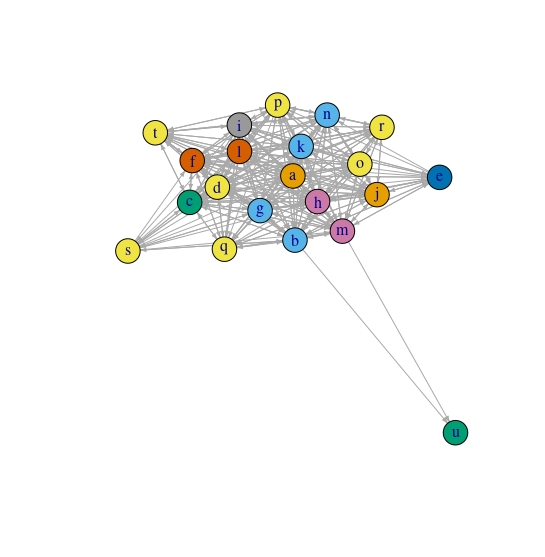
We performed both clustering and community detection on the network. Community discovery is process of clustering nodes of the network which are densely connected but sparsely to the rest of the network.



In order to understand the shared network characteristics among the nodes we used *Clustering and Node aggregation technic.* With this method, we find closely linked groups of nodes and consider them as a single node in the condensed graph. By doing so, it enables us to decrease the quantity of nodes and edges while maintaining the network's general structure.

We used Louvian method which is a well-known technique for community discovery in graphs. It seeks to divide a graph's nodes into distinct communities or clusters depending on the graph's topology. It is a two-stage procedure: the first phase involves assigning each node to a distinct community, and the second step involves iteratively merging neighboring communities to maximize a quality function that gauges the modularity of the final partition. The modularity gauges how closely related nodes within a community are to those connecting nodes between communities.

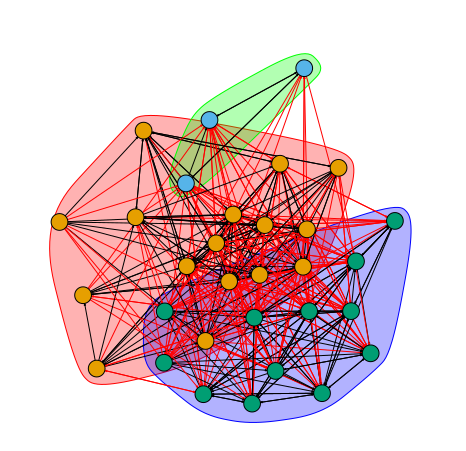
After aggregating the vertices which belong to the same community into a vertex, we removed the isolated vertices from the graph.



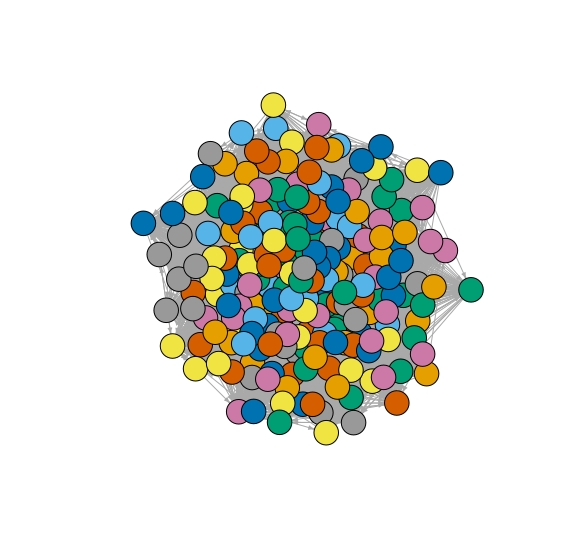
In the picture above we can clearly see that there are departments or teams in this network which have mostly close relationship with each other. One of the insight we can get from graph is that clusters “a”, “k”, “l”, “o” seems to be important having dense relationship with almost other groups. Another striking pattern is that cluster “u” never emailed to any of the groups which may indicate that this group does not actively play role in the organization but only gets some updates from “b” and “m” teams. Next outstanding point can be that there are some groups such as “t”, “s”, “e”, “q” which are located at the edges of the network, seems to have close relationship with only specific departments.

**6. Resulting graph with too many vertices and edges will look very messy in the plot. Try to filter vertices and their edges in some way having in resulting plot (visualization) 30 – 100 vertexes. Differentiate vertices (by color, size, shape) and edges (color, type) of graph. Think about opportunity to assign weights to edges differentiating them accordingly.**

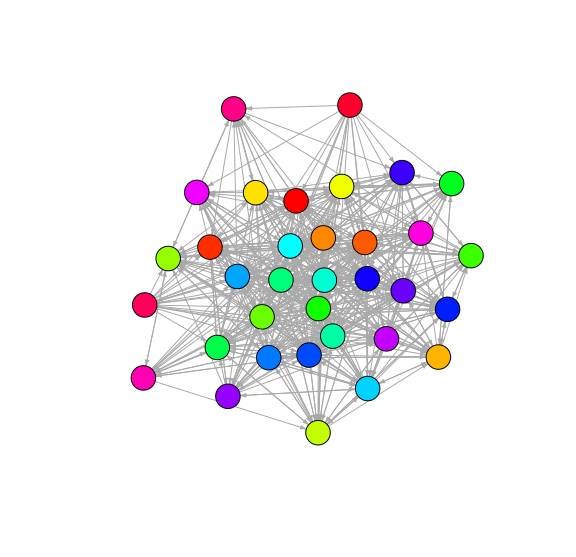
After assigning weight and eliminating some of the nodes we get more visual plot as below. We also changed some parameters such as arrow size, vertex color and vertex size.



We also cluster the reduced graph using louvain algorithm using resolution parameter 10 which produces 221 clusters.

****

We then assign weights to the edges and nodes to prune some of them. After selecting nodes with weight more than two using induced subgraph we end up with 31 nodes.

****

**7. Source code:**[**https://github.com/anar-sixeliyev/GW\_BIG\_DATA**](https://github.com/anar-sixeliyev/GW_BIG_DATA)