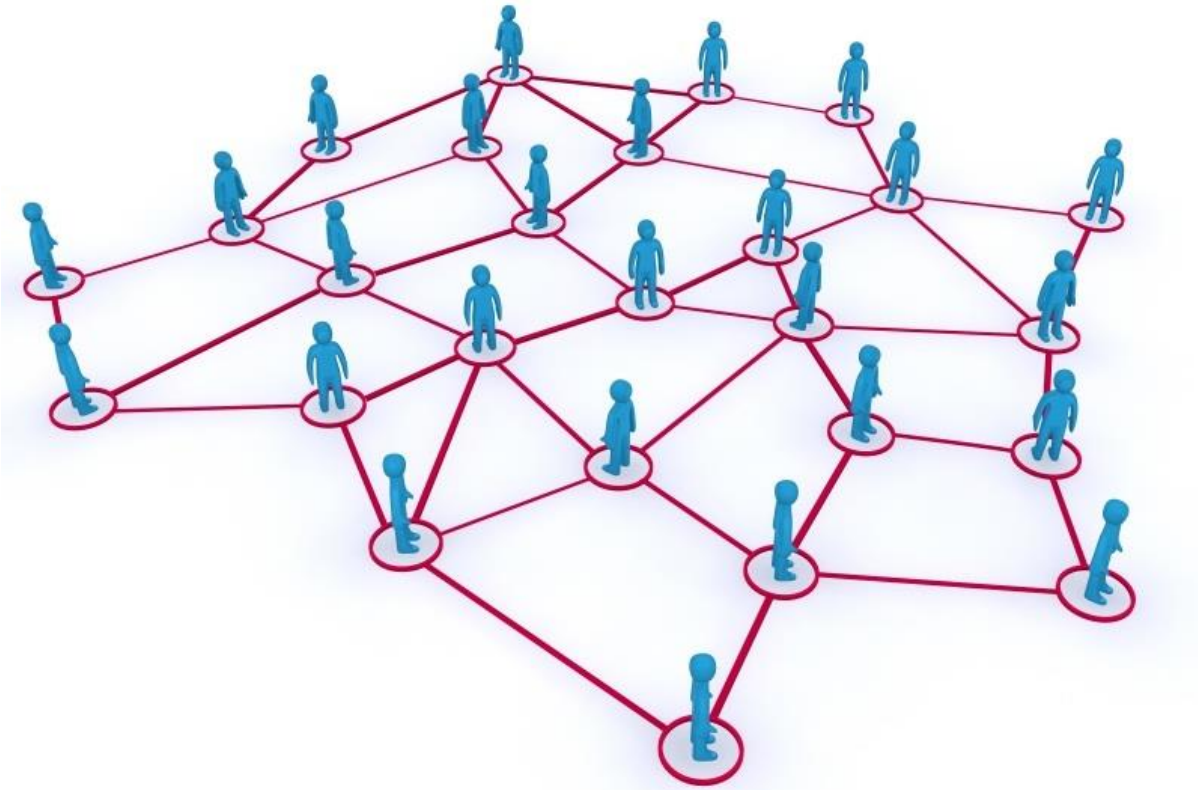


Network Analysis using igraph

Team OptimizeR

TABLE OF CONTENTS

Executive Summary:

The main objective of the project is to explore the concepts of Network Analysis using igraph package in R. Network analysis has become one of the pioneer concepts in exploring how people or institutes are connected with each other. Analyzing the strength of the relationships between the connections, and focusing on the interactions rather than individual behaviour alone. The team has worked to give a sneak peak about the concepts and build various networks using igraph package available in R.

Introduction to Network Analysis:

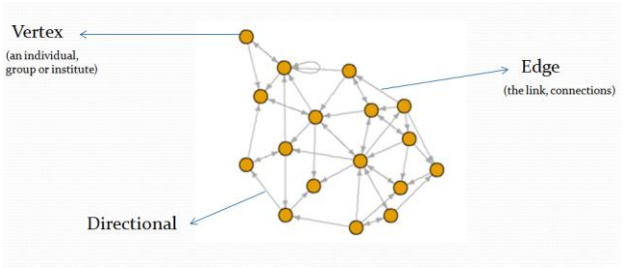
Network Analysis is focused on uncovering the patterning of people's interaction. Network analysis is based on the intuitive notion that these patterns are important features of the lives of the individuals who display them. Network analysts believe that how an individual lives depends in large part on how that individual is tied into the larger web of social connections. Many believe, moreover, that the success or failure of societies and organizations often depends on the patterning of their internal structure. That kind of intuition is probably as old as humankind.

There are many applications using Network Analysis. They serve as channels for informal insurance and risk sharing, and network structure influences patterns of decisions regarding education, career, hobbies, criminal activity, and even participation in micro-finance. Beyond the role of "social" networks in determining various economic behaviours, there are also many business and political interactions that are networked. Networks of relationships among various firms and political organizations affect research and development, patent activity, trade patterns, and political alliances. Given the many roles of networks in economic activity, they have become increasingly studied by economists.

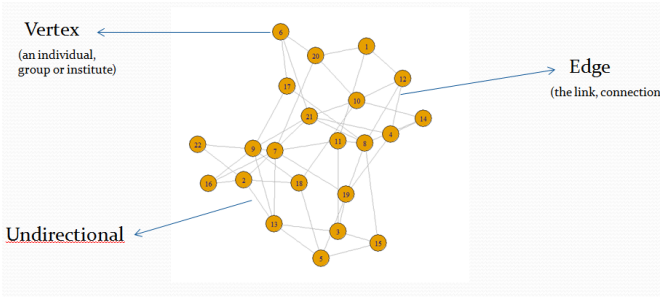
Methodology:

There are a lot of packages available in R such as network, statnet, igraph, sna, latentnet to explore the network analysis. We have decided to use igraph to continue working. There are some basic elements about the network that one has to know before jumping into the analysis.

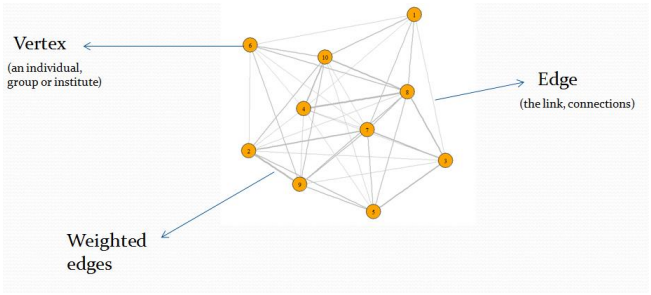
Vertex	An individual or a group that has some information or connections with the other individuals in any medium
Edge	The relationship between two people or entity is called edge. It tells us how they are connected and defines the relationships.
Directed graph	Whenever there is information propagating from one vertex to another, an arrow head on the edge would help to represent the direction in which the information is transmitting. These types of graphs are called directed graph
Undirected graph	The edge with no directions whatsoever and only represents the connections between vertices is called undirected graph.
Weighted Edge	When there is strong connection between two vertexes, then there is said to be a weighted edge between them.
Weighted Vertex	When a specific vertex has strong connection in terms of various factors, then it is said to have strong connector.



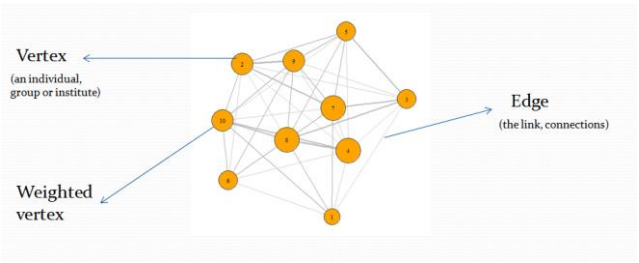
a) Directional graph



b) Undirectional Graph



c) Weighted Edges



d) Weighted Vertex

There are various kinds of data that you can work with:

1. Adjacency Matrix: It is a matrix in which the rows and columns represent different vertices. In an unweighted adjacency matrix, the edges are represented by 0 or 1, with indicating that

these two nodes are connected. If two nodes are connected, they are said to be adjacent (hence the name, adjacency matrix). In a weighted matrix, however, you can have different values, indicating different edge qualities (or tie strenghts).

Code:

```
Install.library("igraph")
Library("igraph")
#creating the adjacency matrix
a=c(0,1,0,1,0,1,0,1)
b=c(1,0,1,0,0,1,1,0)
c=c(0,1,0,1,1,0,0,1)
d=c(1,0,1,0,1,0,1,0)
e=c(0,0,1,1,0,1,0,0)
f=c(1,1,0,0,1,0,0,1)
g=c(0,1,0,1,0,0,0,0)
h=c(1,0,1,0,1,1,0,0)
am=matrix(c(a,b,c,d,e,f,g,h),nrow=8,byrow=TRUE)
#assigning the row and column labels
dimnames(am)=list(c("A","B","C","D","E","F","G","H"),
                  c("A","B","C","D","E","F","G","H"))
am
```

	A	B	C	D	E	F	G	H
A	0	1	0	1	0	1	0	1
B	1	0	1	0	0	1	1	0
C	0	1	0	1	1	0	0	1
D	1	0	1	0	1	0	1	0
E	0	0	1	1	0	1	0	1
F	1	1	0	0	1	0	0	1
G	0	1	0	1	0	0	0	0
H	1	0	1	0	1	1	0	0

#function to convert the adjacency matrix to a graph object. Creating an unidirectional network graph.

```
am_graph = graph.adjacency(am,mode = "undirected")
am_graph
```

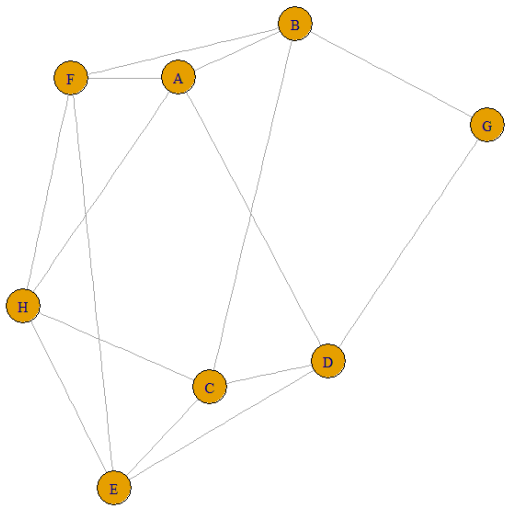
In igraph version 0.6, the way the igraph object is displayed has changed. The above code will return:

```
IGRAPH UN-- 8 15 --
+ attr: name (v/c)
+ edges (vertex names):
 [1] A--B A--D A--F A--H B--C B--F B--G C--D C--E C--H D--E D--G E--F E--H F--H
```

The letters on the first line (there can be up to 4) indicates some basic information about the graph. The first letter indicates whether this is a directed ('D') or undirected ('U') graph. The 2nd letter tells you if this is a named ('N') graph--i.e., whether or not the vertex set has a

'name' attribute. The 3rd letter tells you if this graph is weighted ('W'). The fourth letter is 'B' for bipartite graphs. These letter codes are followed by two numbers: the first is the number of vertices and the second is the number of edges.
The second line gives you information about the 'attributes' associated with the graph. In this case, there is only one attribute, called 'name', which is associated with the vertex set.

```
plot(am_graph)
```



2. Incidency Matrix: It is a matrix in which the rows and columns represent different type of vertices. The matrix doesn't imply any interaction among row vertices and the column vertices. For example, picture a network of pollination interactions between plants and their pollinators. Pollinators only pollinate plants, so there are no edges between pollinators. Similarly, there are no edges between plants. The second scenario is when you have individuals belong to particular groups. Edges represent membership of individuals in groups. These are sometimes called affiliation networks. Here, you can imagine 'individuals' being one set of nodes, and 'groups' being another set of nodes. Individuals can only be connected to groups, and vice versa.

Code:

```
# creating a incidency matrix
a=c(1,1,0,0,0,0,0,0,0,1)
b=c(0,0,1,0,0,1,0,1,0,0)
c=c(1,0,1,0,1,0,0,0,1,0)
d=c(0,1,0,1,0,0,0,0,1,0)
e=c(0,0,1,0,1,0,0,1,1,0)
f=c(0,0,0,0,0,0,1,0,0,1)
g=c(1,0,1,0,0,1,0,0,1,1)
h=c(1,1,0,1,1,0,1,0,1,0)
im=matrix(c(a,b,c,d,e,f,g,h),nrow=8,byrow=TRUE)
#assigning row and column labels to the matrix
dimnames(im)=list(c("A","B","C","D","E","F","G","H"),
                  c("G1","G2","G3","G4","G5","G6","G7","G8","G9","G10"))
im
```

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
A	1	1	0	0	0	0	0	0	0	1
B	0	0	1	0	0	1	0	1	0	0
C	1	0	1	0	1	0	0	0	1	0
D	0	1	0	1	0	0	0	0	1	0
E	0	0	1	0	1	0	0	1	1	0
F	0	0	0	0	0	0	1	0	0	1
G	1	0	1	0	0	1	0	0	1	1
H	1	1	0	1	1	0	1	0	1	0

#graph.incidence is a function that is used to convert the incidence matrix to a graph object

```
im_graph = graph.incidence(im)
```

If we examine the incidency graph object, we observe the following:

```
im_graph
```

In igraph version 0.6, the way the igraph object is displayed has changed. The above code will return:

```
IGRAPH UN-B 18 30 --
+ attr: type (v/l), name (v/c)
+ edges (vertex names):
 [1] A--G1  A--G2  A--G10 B--G3  B--G6  B--G8  C--G1  C--G3  C--G5  C--G9  D--G2  D--G4  D--G9
E--G3  E--G5
[16] E--G8  E--G9  F--G7  F--G10 G--G1  G--G3  G--G6  G--G9  G--G10 H--G1  H--G2  H--G4  H--
G5  H--G7  H--G9
```

From the first line, we see that we have created an Undirected, Named graph that is Bipartite and has 18 nodes and 30 edges. The second line tells us that there are two vertex attributes: type and name. Let's check these out:

```
 V(im_graph)$type
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE
```

```
 V(im_graph)$name
[1] "A"   "B"   "C"   "D"   "E"   "F"   "G"   "H"   "G1"  "G2"  "G3"  "G4"  "G5"  "G6"  "G7"  "G8"
"G9"  "G10"
```

You see that igraph has arranged both individuals and groups as nodes, and then created the attribute 'type' to indicate that these are two distinct classes of nodes.

#assigning different shapes to rows and columns

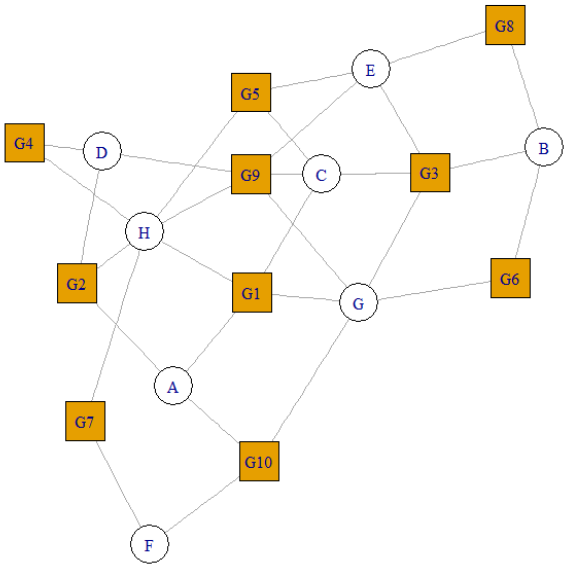
```
shapes=c(rep("circle",nrow(im)),rep("square",ncol(im)))
```

#to make the graph look more readable, we are placing the labels of rows and columns inside the entity.

```
labeldistances=c(rep(0,nrow(im)+ncol(im)))
```

#plotting the graph, by their respective shapes, in a certain angle to look better, and assigning color based on the type of the vertex

```
plot(im_graph,vertex.shape=shapes,vertex.label.degree=-pi/2,
   vertex.label.dist=labeldistances,vertex.color=V(im_graph)$type)
```



This plot tells that various row vertices are linked with each other through the column vertices. For instance, A and C are linked using the G1, which defines the relationship between A and C as well.

As we have seen the incidency matrix, in this case, you might actually be interested in generating a social network of individuals based on co-membership. We can do this using what is called a one-mode projection, or bipartite projection. Let's say you are interested in the co-membership relations between the individuals. What you need to do is to create a one-mode projection of the bipartite network.

#Creating an incidency matrix

```
a=c(1,1,0,0,0)
b=c(0,0,1,0,0)
c=c(1,0,1,0,1)
d=c(0,1,0,1,0)
e=c(0,0,1,0,1)
f=c(0,1,0,1,0)
bi=matrix(c(a,b,c,d,e,f),nrow = 6,byrow = TRUE)
dimnames(bi)=list(c("A","B","C","D","E","F"),
            c("G1","G2","G3","G4","G5"))
bi_graph=graph.incidence(bi)
```

	G1	G2	G3	G4	G5
A	1	1	0	0	0
B	0	0	1	0	0
C	1	0	1	0	1
D	0	1	0	1	0
E	0	0	1	0	1
F	0	1	0	1	0

#to bipartite the network we use “bipartite.projection” function

```
bp=bipartite.projection(bi_graph)
bp
```

which will give:

```
$proj1
IGRAPH UNW- 6 7 --
+ attr: name (v/c), weight (e/n)
+ edges (vertex names):
[1] A--C A--D A--F B--C B--E C--E D--F

$proj2
IGRAPH UNW- 5 5 --
+ attr: name (v/c), weight (e/n)
+ edges (vertex names):
[1] G1--G2 G1--G3 G1--G5 G2--G4 G3--G5
```

You can see here that the bipartite projection has given us a list object with two graphs: pr$proj1 (6 vertices and 7 edges) and pr$proj2 (5 vertices and 5 edges).

#1st graph object adjacency matrix with weights on the edges

```
get.adjacency(bp$proj1,sparse = FALSE, attr = "weight")
```

	A	B	C	D	E	F
A	0	0	1	1	0	1
B	0	0	1	0	1	0
C	1	1	0	0	2	0
D	1	0	0	0	0	2
E	0	1	2	0	0	0
F	1	0	0	2	0	0

This matrix is the adjacency matrix for the rows only with the weights

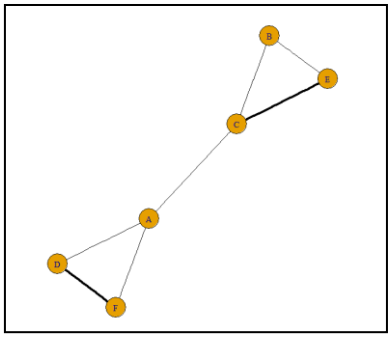
#2nd graph object

```
get.adjacency(bp$proj2,sparse = FALSE, attr = "weight")
```

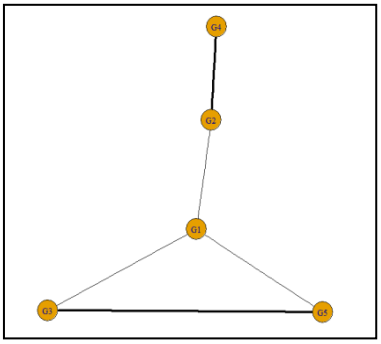
	G1	G2	G3	G4	G5
G1	0	1	1	0	1
G2	1	0	0	2	0
G3	1	0	0	0	2
G4	0	2	0	0	0
G5	1	0	2	0	0

This matrix is the adjacency matrix for the columns only with the weights.

```
plot(bp$proj1,edge.width=E(bp$proj1)$weight^2,edge.color="black",vertex.label=V(bp$proj1)$name)
```




```
plot(bp$proj2,edge.width=E(bp$proj2)$weight^2,edge.color="black",vertex.label=V(bp$proj2)$name)
```



Now, you have an affiliation network of individuals based on co-membership in groups. The dark links represents stronger relationships between the two vertices.

Vertex Attributes:

Reading data to analysis the vertex metrics

```
zach <- read.table("C:/Users/VISWANATH/Desktop/Aditya/Classwork/R/Project 2/zach.txt", quote="\"")
#for simplicity we have selected 10 rows and 10 columns
zach1 <- zach[0:10,0:10]
#assigning the names to the rows and the columns
dimnames(zach1)=  list(  c("A","B","C","D","E","F","G","H","I", "J"),
                         c("A","B","C","D","E","F","G","H","I", "J")    )
#converting the data table to matrix
zach1 = as.matrix(zach1)
#converting to igraph objects
g<-  graph_from_adjacency_matrix(zach1 , mode = "undirected")
```

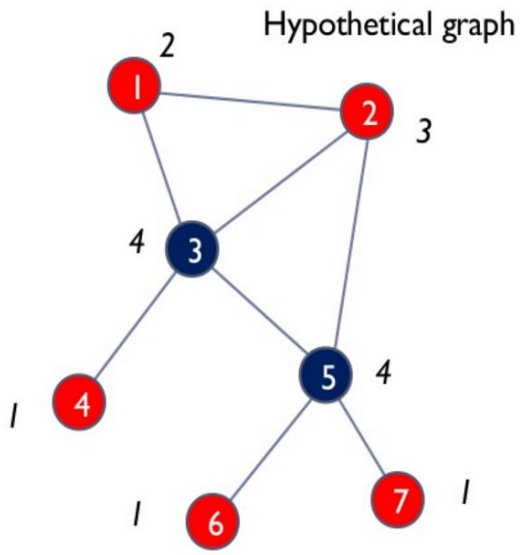
Centrality :

Centrality is the measure of which nodes are most important nodes in the network . Centrality can be measured by using various measures such as

- Degree Centrality
- Closeness Centrality
- Betweenness Centrality

Degree Centrality:

Degree is a measure of number of edges connected to a node or number of neighbour a node is connected to for an undirected network.



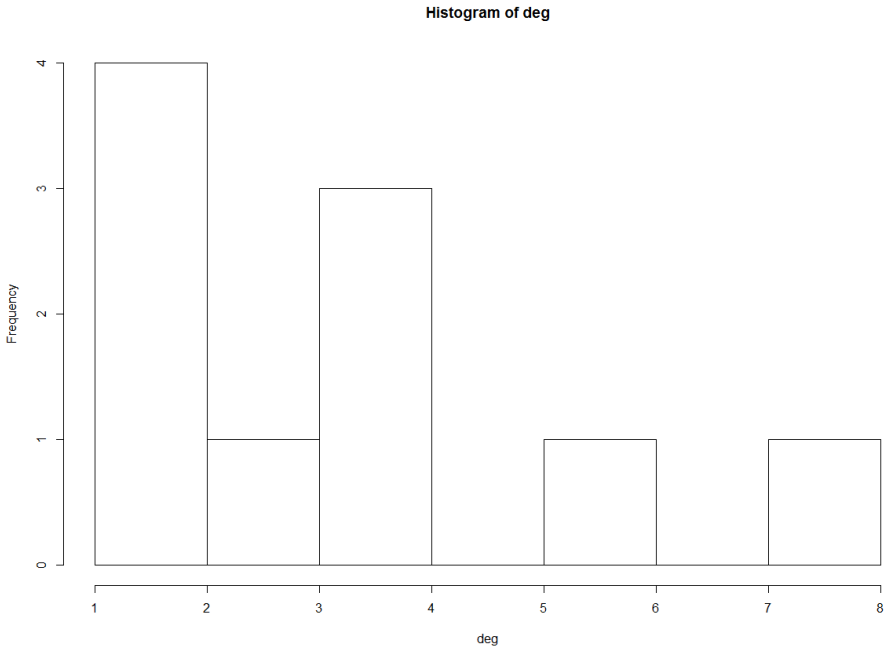
From the graph which is undirected network node 3 ,5 has highest degree.

For a directed network Indegree and Outdegree are calculated. Indegree is number of connection incoming towards the node. Outdegree is number of connection going out from a corresponding node.

In R degree centrality is calculated by :

```
Deg  <- degree(g ,mode = "all")
```

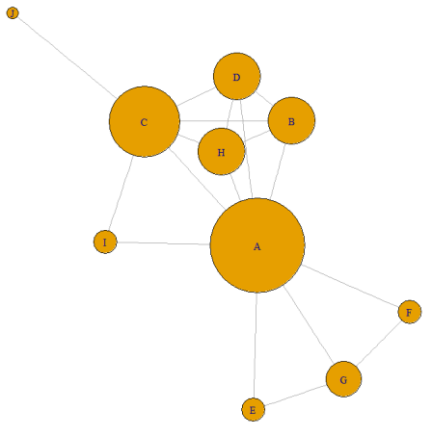
A histogram can be plotted with the degree to analyze the degree range across the network.



In the above network the node has a highest degree of 8, but the most frequent occurred degree is 1.The Degree centrality can be normalized by dividing the degree with (n-1) where n is total nodes of the network.

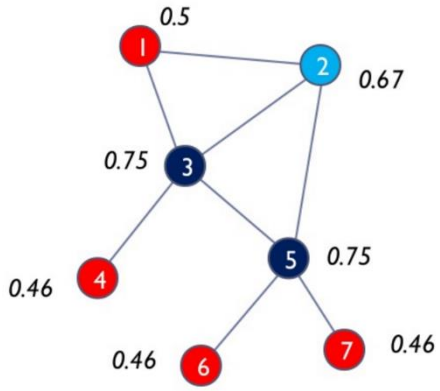
Degree centrality indicates the most connected nodes and through which nodes the information can spread in the closest neighbour.

```
plot(g,vertex.size = deg*6)
```



Closeness Centrality:

Closeness is calculated by reciprocal mean length of shortest path from the node to the other nodes.



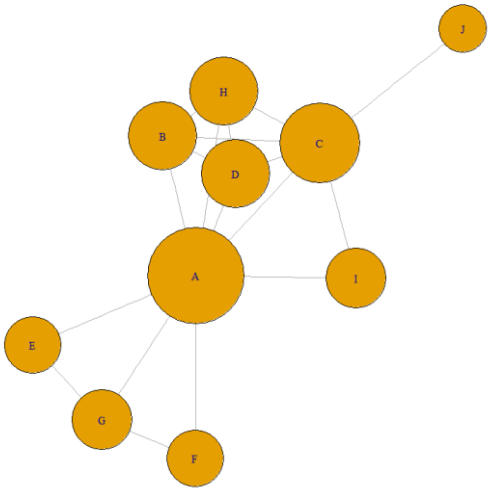
$$C_c(i) = \left[\sum_{j=1}^{N} d(i,j)\right]^{-1}$$

Considering the node 3, the number of hops it should make to reach all the nodes are 8 (i.e. 1 hop to reach 1,2,4,5 nodes 2 hops for 7,6) . Taking the inverse of the number and normalizing it (To normalize divide by n-1).

Closeness is equal to 0.75. In similar way closeness is calculated for all the nodes in the network.

It can be used to calculate the speed at which the information can be spread across the network.

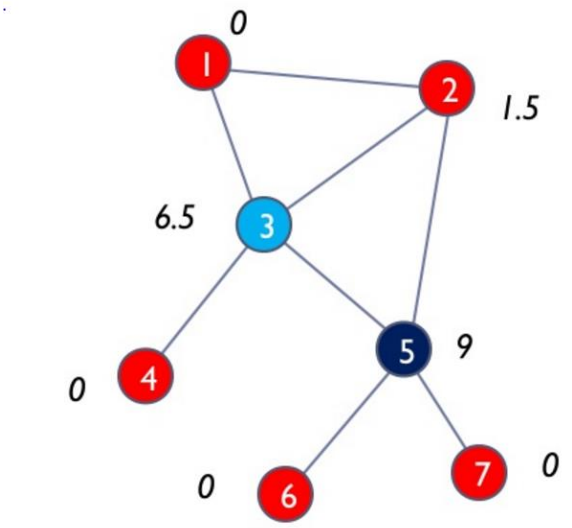
```
closenes <- closeness(g,weights = NULL, normalized = T)

plot(g, vertex.size = closenes * 50)
```



Betweenness Centrality :

Betweenness Centrality is the defined as number of the shortest path between i and j on which K resides

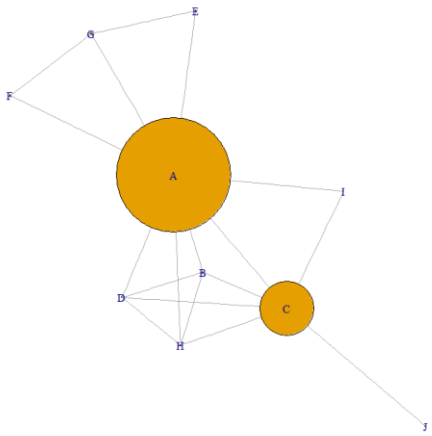
$$C_B(i) = \sum_{j<k} g_{jk}(i) / g_{jk}$$



In the above graph the 4,6,7,1 are not between any nodes hence the betweennes is zero. The betweeness for 5 is 9.

Betweenness indicates node that lies in the communication path can control the communication flow hence the most important person .

```
bet <- betweenness(g,directed = TRUE ,weights = NULL, normalized = T )

plot(g,vertex.size = bet* 100)
```



Edge Attributes:

Edge_Density :

Edge_density is used to compare two networks or a two regions in a single network

The edge_density is defined as ratio of number of the actual connections to the total potential connections. Let the actual connection of the network be l. The total number of the potential connections that can be formed with n edges is n(n-1)/2 . So the edge density of the network would be l/ n(n-1)/2 for a undirected network. If it is a directed network the potential connection would be double and the Edge_density would half of the undirected network for directed .

In R :

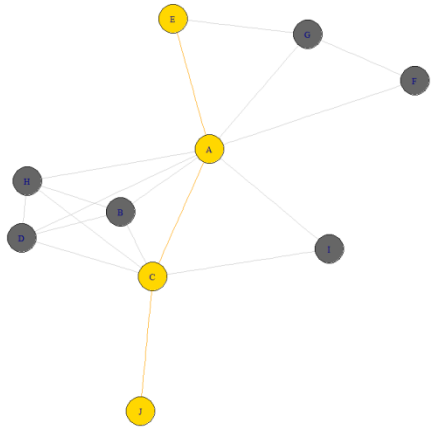
```
edge_density(graph, loops = FALSE)
```

Diameter :

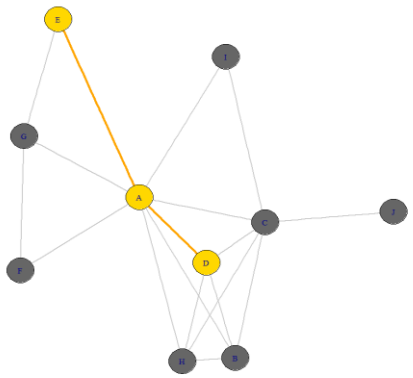
Diameter gives the length of the longest geodeisc . Diameter indicates how long it would take to reach any node in the network .

In R :

```
diameter(g , directed = TRUE , unconnected = FALSE , weights = NULL)
#diameter for this network is 3
# To get the path which diameter is passing through
d = get_diameter(g, directed = TRUE , unconnected = FALSE , weights = NULL)
#assigning different colors to diameter path and others.
vcol <- rep("gray40", vcount(g))
vcol[d] <- "gold"
ecol <- rep("gray80", ecount(g))
ecol[E(g, path=d)] <- "orange"
#plotting the network with the diameter path
plot(g, vertex.color=vcol, edge.color=ecol, edge.arrow.mode=0)
```




```
#to find the shortest path in the network
news.path <- shortest_paths(g, from = V(g)[V(g)$name == "E"],   to  = V(g)[V(g)$name == "D"],
                            output = "both")
# both path nodes and edges
# Generate edge color variable to plot the path:
ecol <- rep("gray80", ecount(g))
ecol[unlist(news.path$epath)] <- "orange"
# Generate edge width variable to plot the path:
ew <- rep(2, ecount(g))
ew[unlist(news.path$epath)] <- 4
# Generate node color variable to plot the path:
vcol <- rep("gray40", vcount(g))
vcol[unlist(news.path$vpath)] <- "gold"
#plotting the shortest distance in the network
plot(g, vertex.color=vcol, edge.color=ecol,  edge.width=ew, edge.arrow.mode=0)
```



The end vertices of the diameter

farthest_vertices(g,directed = TRUE , unconnected = FALSE , weights = NULL)

Reciprocity :

Reciprocity of the network is the likelihood of the nodes to be mutually linked . The networks can be linked in Dyads and Traids. For Example : When two nodes (A, B) are linked by A→ B . The likelihood of B being linking back to A (B→ A) is called Reciprocity . In Traids it takes all interactions between the 3 nodes into account.

reciprocity(graph, ignore. Loops = TRUE, mode = c("default", "ratio"))

Reciprocity indicates degree of the mutuality and reciprocal exchange in the network

Network Layouts:

The primary concern of drawing a graph drawing is the spatial arrangement of nodes and edges. More often or not the goal is to effectively depict:

- Connectivity
- Network Distance
- Clustering
- Ordering

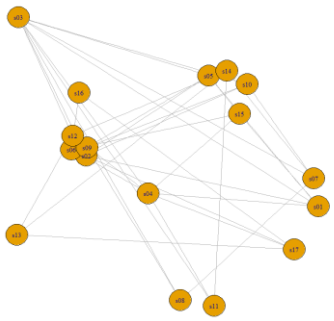
The package “igraph” has 18 different layouts we can use to plot the nodes and edges. Network layouts are simply algorithms that return coordinates for each node in the network. We can also define our own layout. For the purpose of this project we will explore a few igraph layouts and one layout defined by us.

1. Random Layout:

   In this layout the nodes are not arranged on the basis of any property of the edges or nodes. It has a lot of cross overs of edges and rarely used for any type of depiction. This can serve as a reference for our other layouts.

   plot(net, layout=layout_randomly)

   Net is the graph object and layout_randomly is a type of layout defined in the igraph package.

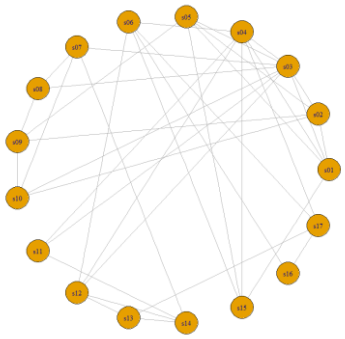


2. Circle Layout:

This is very simple layout. It arranges all the nodes on the periphery of a circle. We can define the size of the circle.

```
l <- layout_in_circle(net)
plot(net, layout=l)
```

Here we are calculating the x and y coordinates using the circle layout algorithm and putting it in l. Then we are using l to plot the layout.



#tkplot helps you to customize the layout in the run time.

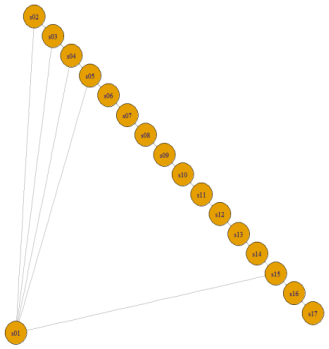
```
tkplot(net)
```

3. User defined Layout:

Layout is nothing but x,y coordinates (N*2) for the N nodes in the graph. So in this user defined layout we will place all nodes in a line and see how only 1 node how it is connected to the others.

```
l <- cbind(1:vcount(net), c(1, vcount(net):2))
plot(net, layout=l)
```

Here we are manually defining the x, y coordinates for each node. The first node will have (1,1). All other nodes will be increasing in x and reducing in y.



4. Fruchterman-Reingold Layout

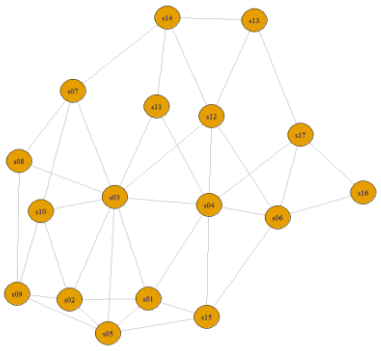
This is one of the most used layout. It works on a force directed algorithm. It tries to get a nice looking graph where edges are equal in length and cross overs are minimal. They simulate the graph as a physical system. Nodes are electrically charged particles that repulse each other when they get too close. The edges act as springs that attract connected nodes closer together. As a result, nodes are evenly distributed through the chart area, and the layout is intuitive in that nodes which share more connections are closer to each other. Some parameters you can set for this layout include:

Area: (the default is the square of # nodes)

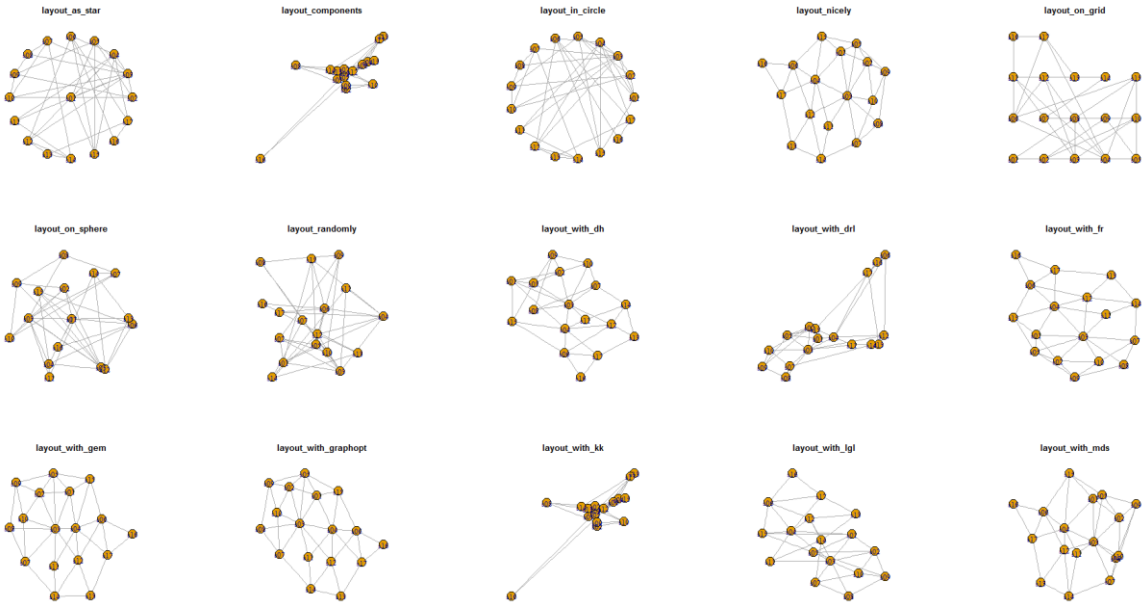
Repulserad: cancellation radius for the repulsion - the area multiplied by # nodes.

Both parameters affect the spacing of the plot.You can also set the “weight” parameter which increases the attraction forces among nodes connected by heavier edges.

```
l <- layout_with_fr(net)
plot(net, layout=l)
```

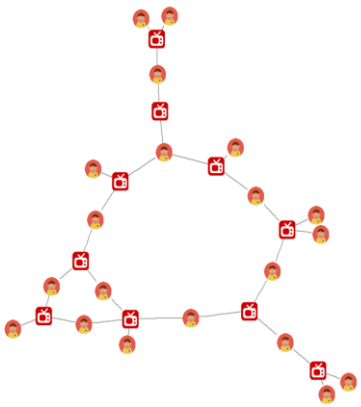


Other Layout:



Two Mode Network Layouts:

Another type of network data is where you have two types of nodes. For example, a network of people and the different news channel they follow. This type of data forms a new matrix called the incidency matrix. When we plot the data the function identifies two mode networks. It plots people with different shape and groups with a different shape so that they can be distinguished. The above layouts can all be applied to this type of matrix as well. Below is a network with the default layout but we have inserted images in place of shapes. This can be very useful to depict business application.



Application of Network Analysis.

Community Detection

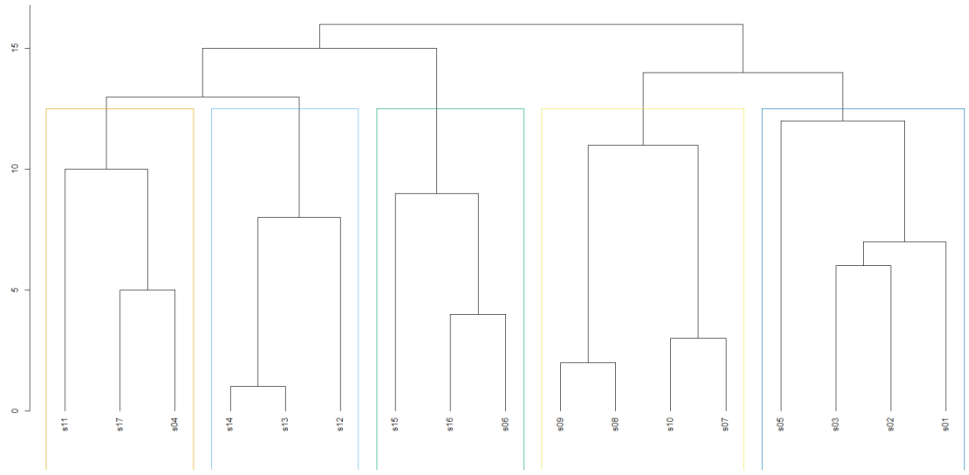
One of the most important tasks when studying networks is to identify communities in the network. For example, when we analyze social networks we would want to find friends who attended the same school or people from the same hometown. They help us identify the organization principles behind the network. There are two possible sources of information while detecting communities in a network graph, namely the network structure and the attributes of nodes.

There are a number of algorithms for community detection. We will be demonstrating two such algorithms which can be easily used in igraph.

1. Community Detection based on edge betweenness:
   It is a hierarchical decomposition process where edges are removed in the decreasing order of their edge betweenness scores (i.e. the number of shortest paths that pass through a given edge). This is motivated by the fact that edges connecting different groups are more likely to be contained in multiple shortest paths simply because in many cases they are the only option to go from one group to another. This method yields good results but is very slow because of the computational complexity of edge betweenness calculations and because the betweenness scores have to be re-calculated after every edge removal.

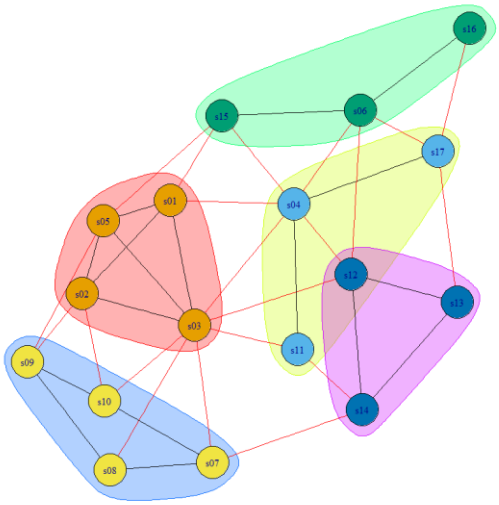
   We used hierarchical clustering to form clusters on the basis of betweenness of edges. We can define the number of clusters we would like to divide the data. Below is the dendrogram of which takes the between as the distance parameter to form the dendrogram.

   ```
   ceb <- cluster_edge_betweenness(net)

   dendPlot(ceb, mode="hclust")
   ```



   We can very easily plot the cluster in the graph and distinguish them into different communities as seen in the diagram below.

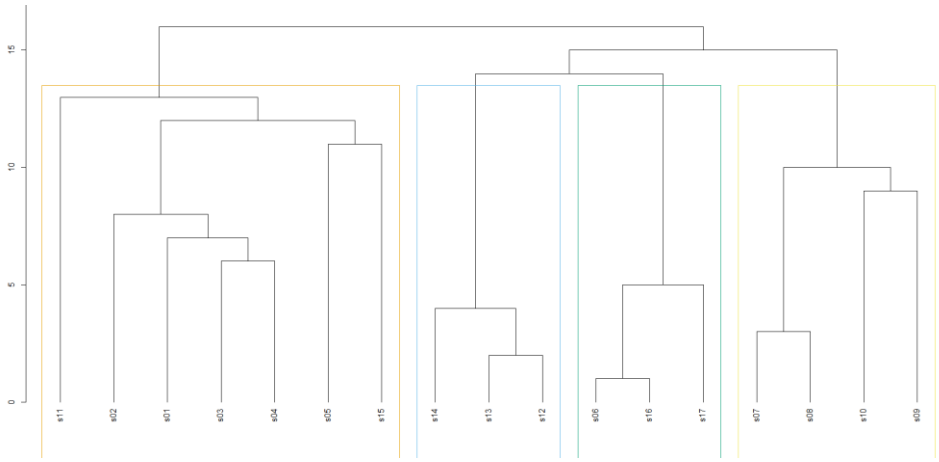
   ```
   plot(ceb, net)
   ```



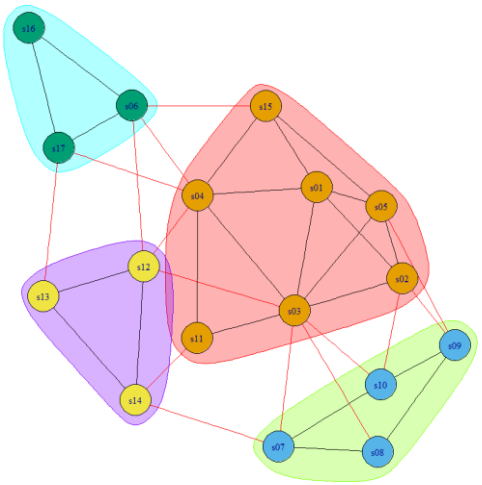
2. Community Detection based on fast greedy

   Fastgreedy.community is another hierarchical approach, but it is bottom-up instead of top-down. It tries to optimize a quality function called modularity in a greedy manner. Initially, every vertex belongs to a separate community, and communities are merged iteratively such that each merge is locally optimal (i.e. yields the largest increase in the current value of modularity). The algorithm stops when it is not possible to increase the modularity anymore, so it gives you a grouping as well as a dendrogram. The method is fast and it is the method that is usually tried as a first approximation because it has no parameters to tune. However, it is known to suffer from a resolution limit, i.e. communities below a given size threshold (depending on the number of nodes and edges if I remember correctly) will always be merged with neighboring communities. This can be easily implemented using igraph with the help of the below code:

```
cfg <- cluster_fast_greedy(as.undirected(net))
dendPlot(cfg, mode="hclust")
```




```
plot(cfg, as.undirected(net))
```


s16
s06
s17
s15
s04
s01
s05
s12
s13
s02
s03
s11
s09
s10
s14
s07
s08