39_Project3_Part2

Tilak Poudel

2025-04-20

Part 2: Use this link https://kateto.net/netscix2016.html to replicate all the SNA results and interpret them carefully.

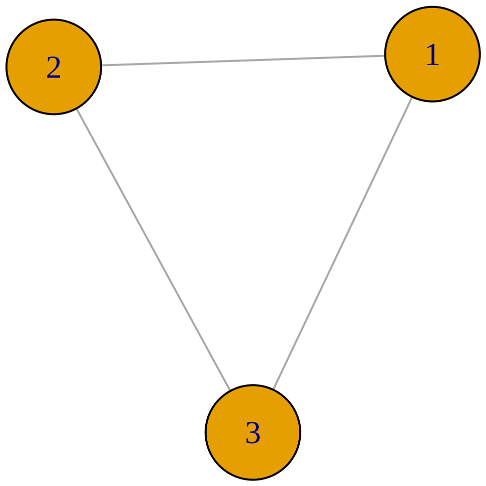
```r
library(igraph)
```

```
##
## Attaching package: 'igraph'

## The following objects are masked from 'package:stats':
##
##     decompose, spectrum

## The following object is masked from 'package:base':
##
##     union
```

```r
g1 <- graph(edges=c(1,2, 2,3, 3,1),n=3,directed=F)
```

```
## Warning: `graph()` was deprecated in igraph 2.1.0.
## i Please use `make_graph()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```r
plot(g1,vertex.size=50)
```



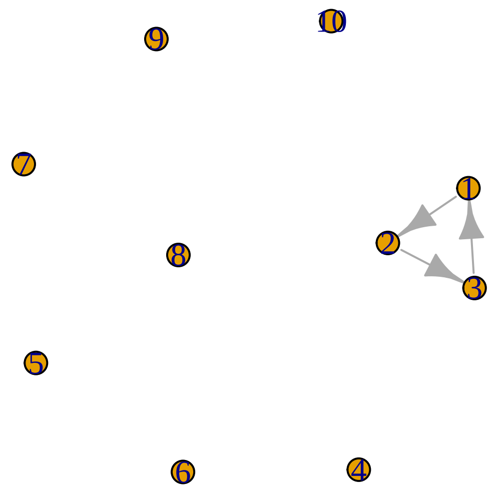
```r
class(g1)
```

```
## [1] "igraph"
```

```r
g1
```

```
## IGRAPH 351b49f U--- 3 3 --
## + edges from 351b49f:
## [1] 1--2 2--3 1--3
```

```r
g2 <- graph(edges=c(1,2, 2,3, 3,1),n=10)
plot(g2,vertex.size=10)
```



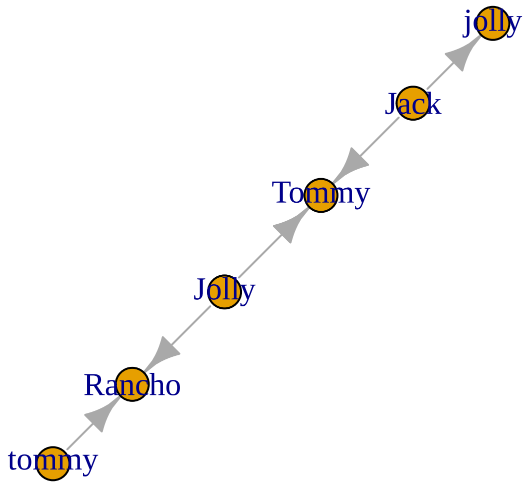
```
g2
```

```
## IGRAPH 165b093 D--- 10 3 --
## + edges from 165b093:
## [1] 1->2 2->3 3->1
```

```
g3 <- graph
```

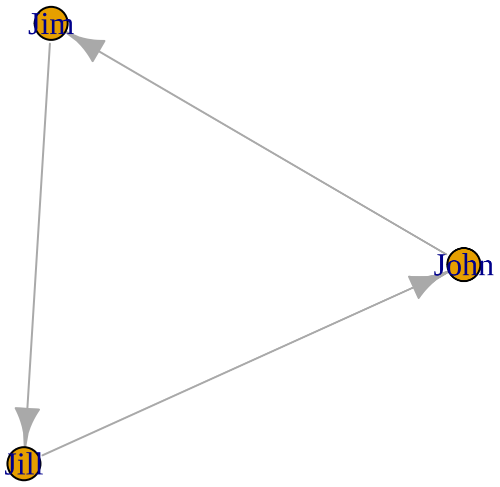
Here we loaded the igraph library and created a simple graph with 3 vertices and 3 edges. The `plot` function is used to visualize the graph, and we can see that the vertices are represented as circles with a size of 50. The second graph `g2` is created with 10 vertices and the same edges as `g1`. The `plot` function is used again to visualize the graph, but this time the vertices are represented as smaller circles with a size of 10 where 3 edges are created between the vertices 1, 2, and 3. The `class` function is used to check the class of the graph object, which is `igraph`.

```
g4 <- graph(c("Jack","jolly","Jack","Tommy","Jolly","Rancho","tommy","Rancho","Jolly","Tommy"))
plot(g4)
```

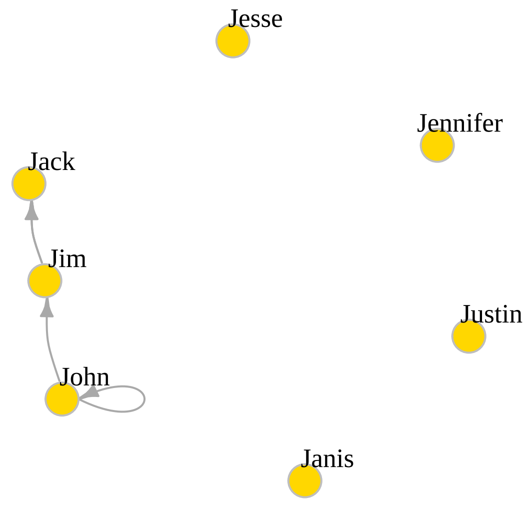


We created a graph directed graph `g4` with 6 vertices and 5 edges. The edges are represented as lines connecting the vertices.

```r
g5 <- graph( c("John", "Jim", "Jim", "Jill", "Jill", "John"))
plot(g5)
```

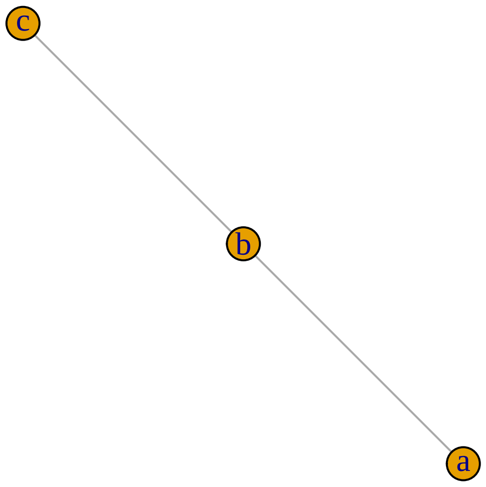


```
g4 <- graph( c("John", "Jim", "Jim", "Jack", "Jim", "Jack", "John", "John"), isolates=c("Jesse", "Janis
plot(g4, edge.arrow.size=.5, vertex.color="gold", vertex.size=15,vertex.frame.color="gray",vertex.label
```

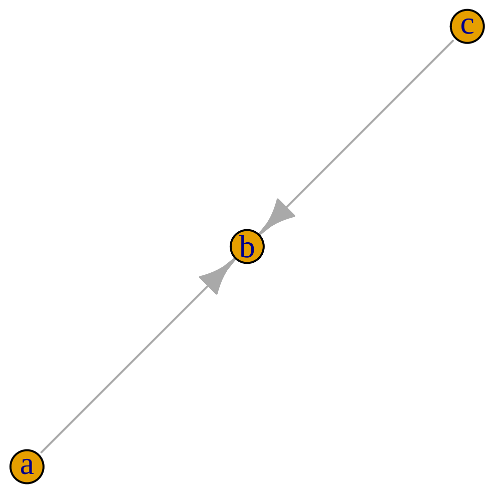


The graph `g5` is created with 3 vertices and 2 edges. The `plot` function is used to visualize the graph, and we can see that the vertices are represented as circles with a size of 15. The `isolates` argument is used to specify the vertices that are not connected to any other vertices in the graph. The `edge.arrow.size`, `vertex.color`, `vertex.frame.color`, `vertex.label.color`, `vertex.label.cex`, `vertex.label.dist`, and `edge.curved` arguments are used to customize the appearance of the graph.

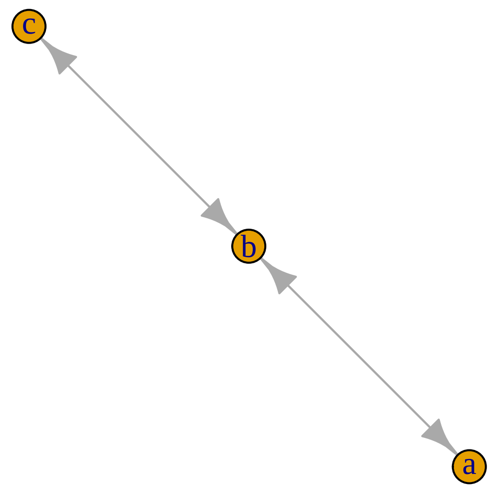
```
plot(graph_from_literal(a---b,b---c))
```



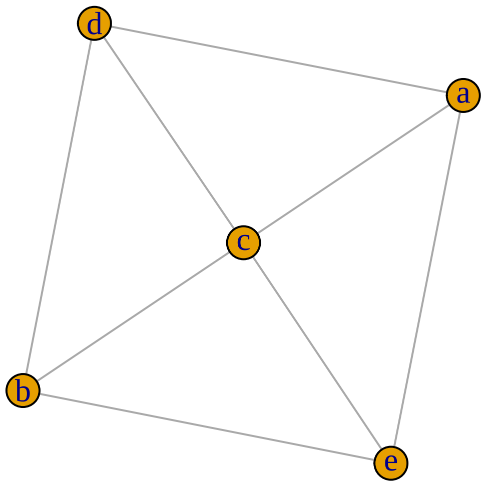
```
plot(graph_from_literal(a--+b,b+--c))
```



```r
plot(graph_from_literal(a+-+b,b+-+c))
```

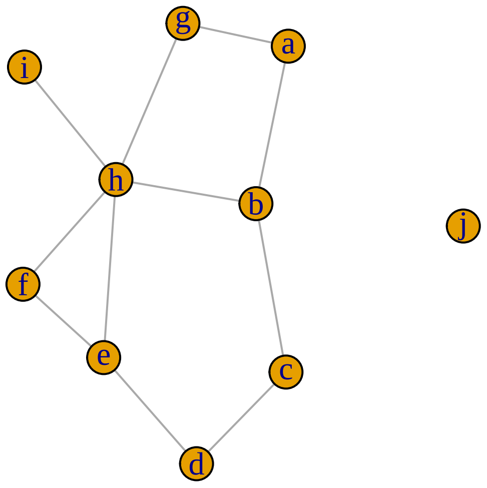


```
plot(graph_from_literal(a:b:c---c:d:e))
```



The `graph_from_literal` function is used to create graphs from a string representation of the edges. The `plot` function is used to visualize the graphs, and we can see that the vertices are represented as circles with different styles of edges connecting them. The `a`, `b`, `c`, `d`, and `e` vertices are connected in different ways, such as directed edges (`--+`), undirected edges (`---`), and multiple edges (`+-+`). The : `a:b:c` notation is used to create a bipartite graph with two sets of vertices, `a`, `b`, and `c`, and `c`, `d`, and `e`. The `plot` function is used to visualize the graphs, and we can see that the vertices are represented as circles with different styles of edges connecting them.

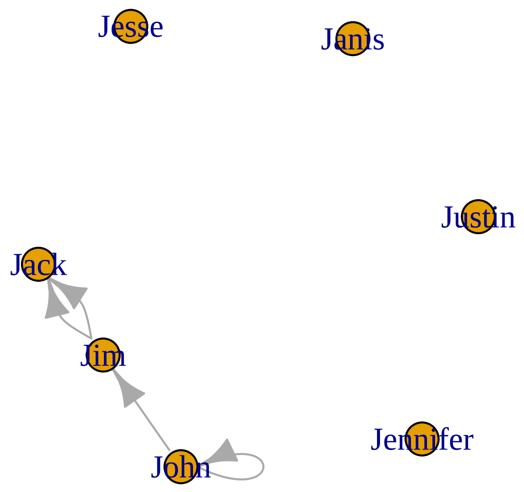
```
g6 <- graph_from_literal(a-b-c-d-e-f,a-g-h-b,h-e:f:i,j)
plot(g6)
```



The `graph_from_literal` function is used to create a graph with 10 vertices and 9 edges. The `plot` function is used to visualize the graph, and we can see that the vertices are represented as circles with different styles of edges connecting them. The `a`, `b`, `c`, `d`, `e`, `f`, `g`, `h`, `i`, and `j` vertices are connected in different ways, such as directed edges (`-`) and undirected edges (`:`).

Edge, vertex, and network attributes

```
plot(g4)
```



```r
E(g4)
```

```
## + 4/4 edges from d099d5d (vertex names):
## [1] John->Jim  Jim ->Jack Jim ->Jack John->John
```

```r
# shows the edges existing between different nodes.

V(g4)
```

```
## + 7/7 vertices, named, from d099d5d:
## [1] John     Jim      Jack     Jesse    Janis    Jennifer Justin
```

```r
#gives the vertices of g4

g4[]
```

```
## 7 x 7 sparse Matrix of class "dgCMatrix"
##            John Jim Jack Jesse Janis Jennifer Justin
## John          1   1    .     .     .        .      .
## Jim           .   .    2     .     .        .      .
## Jack          .   .    .     .     .        .      .
## Jesse         .   .    .     .     .        .      .
## Janis         .   .    .     .     .        .      .
## Jennifer      .   .    .     .     .        .      .
## Justin        .   .    .     .     .        .      .
```

```r
#gives the matrix of edges with respect to different vertices

g4[1,]
```

```
##     John      Jim     Jack    Jesse    Janis Jennifer   Justin
##        1        1        0        0        0        0        0
```

```r
#gives the number of vertices associated with the first vertex

V(g4)$name
```

```
## [1] "John"     "Jim"      "Jack"     "Jesse"    "Janis"    "Jennifer" "Justin"
```

```r
#Names of vertices in g4
```

The E function is used to get the edges of the graph g4, and the V function is used to get the vertices of the graph. The g4[] notation is used to get the matrix of edges with respect to different vertices, and the g4[1,] notation is used to get the number of vertices associated with the first vertex. The V(g4)$name notation is used to get the names of the vertices in g4.

```r
#Different attributes can be set on vertex and edge of g4
V(g4)$gender <-c("male","male","male","male","female","female","male")
E(g4)$type <- "email"
#Assigns "email" to all the attributes of all edges

E(g4)$weight <- 10
edge_attr(g4)
```

```
## $type
## [1] "email" "email" "email" "email"
##
## $weight
## [1] 10 10 10 10
```

```r
vertex_attr(g4)
```

```
## $name
## [1] "John"     "Jim"      "Jack"     "Jesse"    "Janis"    "Jennifer" "Justin"
##
## $gender
## [1] "male"   "male"   "male"   "male"   "female" "female" "male"
```

```r
graph_attr(g4)
```

```
## named list()
```

Attributes can also be set a s following ways:

```r
g4 <- set_graph_attr(g4,"name","Email Network")
graph_attr_names(g4)
```

```
## [1] "name"
```

```r
g4 <- set_graph_attr(g4, "something", "A thing")
graph_attr_names(g4)
```

```
## [1] "name"      "something"
```

```r
graph_attr(g4,"name")
```

```
## [1] "Email Network"
```

```r
graph_attr(g4)
```

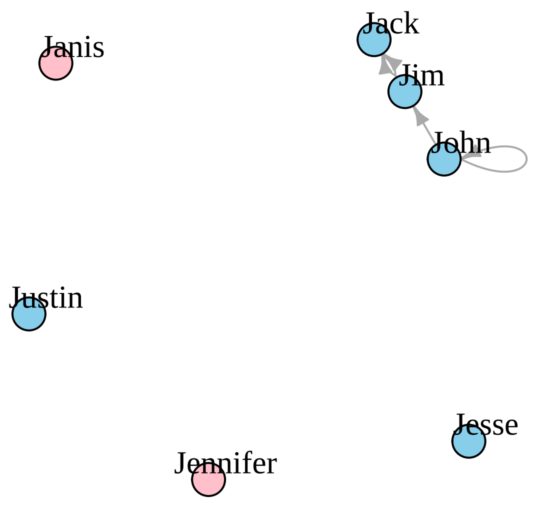
```
## $name
## [1] "Email Network"
##
## $something
## [1] "A thing"
```

```r
g4 <- delete_graph_attr(g4,"something")
graph_attr(g4)
```

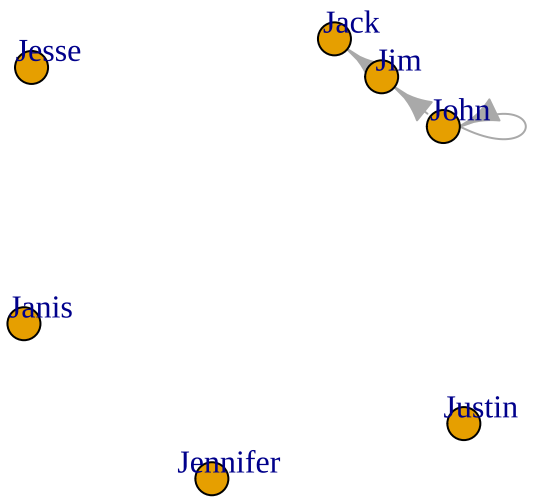
```
## $name
## [1] "Email Network"
```

We deleted the attribute "something" from the graph `g4` using the `delete_graph_attr` function. The `graph_attr` function is used to get the attributes of the graph after deleting the attribute.

```
plot(g4,edge.arrow.size=.5,vertex.label.color="black",vertex.label.dist=1.5,vertex.color=c("pink","skybl
```



```
g4s <- simplify(g4,remove.multiple=T,remove.loop=F,edge.attr.comb=c(weight="sum",type="ignore"))
plot(g4s,vertex.label.dist=1.5)
```

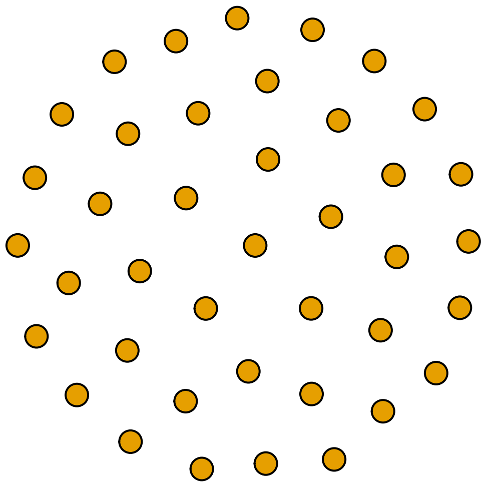


```
g4s
```

```
## IGRAPH 814c835 DNW- 7 3 -- Email Network
## + attr: name (g/c), name (v/c), gender (v/c), weight (e/n)
## + edges from 814c835 (vertex names):
## [1] John->John John->Jim  Jim ->Jack
```

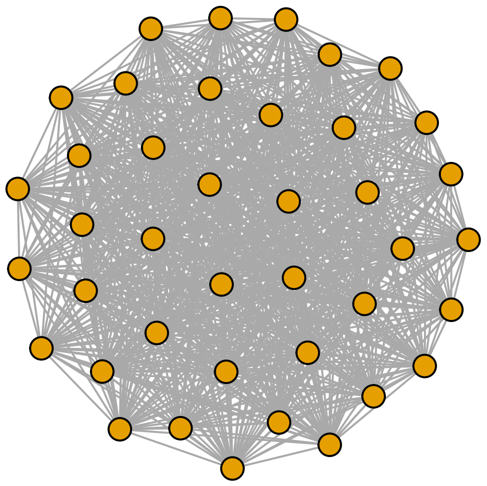
The simplified graph `g4s` is created using the `simplify` function, which removes multiple edges and loops from the graph. The `remove.multiple` argument is set to `TRUE`, and the `remove.loop` argument is set to `FALSE`. The `edge.attr.comb` argument is used to combine the edge attributes, where the weight is summed and the type is ignored. The `plot` function is used to visualize the simplified graph, and we can see that the vertices are represented as circles with different styles of edges connecting them.

```
eg <- make_empty_graph(40)
plot(eg,vertex.size=10,vertex.label=NA)
```



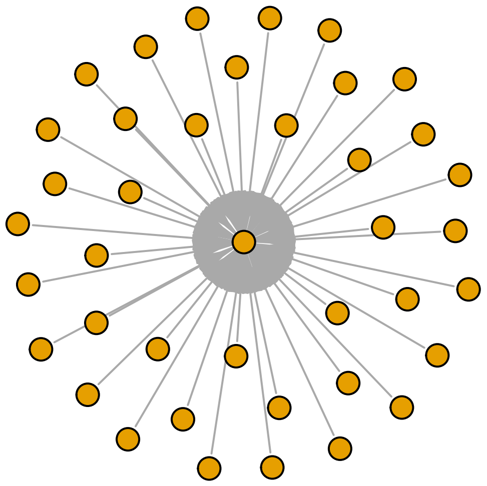
Here we created an empty graph with 40 vertices using the `make_empty_graph` function. The `plot` function is used to visualize the graph, and we can see that the vertices are represented as circles with a size of 10 and no labels.

```r
fg <- make_full_graph(40)
plot(fg,vertex.size=10,vertex.label=NA)
```



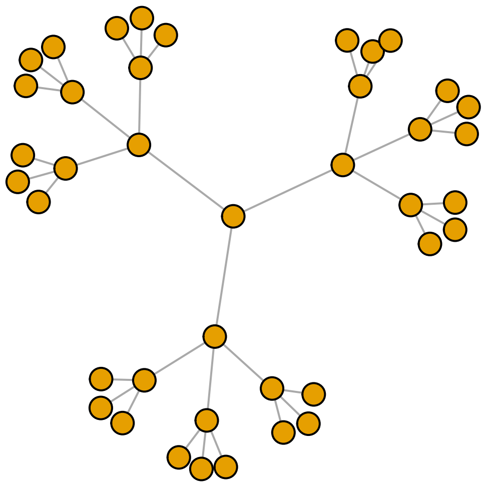
We created a full graph with 40 vertices using the `make_full_graph` function.

```r
st <- make_star(40)
plot(st,vertex.size=10,vertex.label=NA)
```



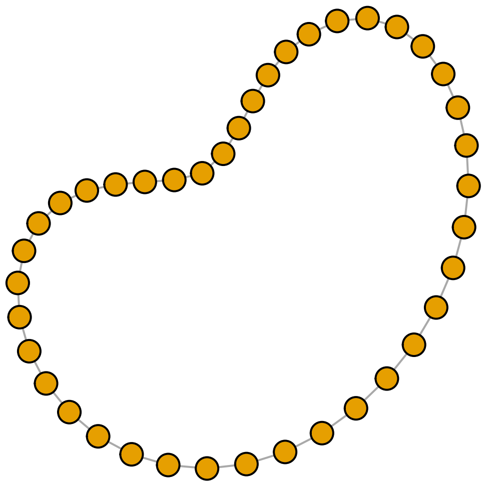
We created a star graph with 40 vertices using the `make_star` function.

```r
tr <- make_tree(40,children=3,mode="undirected")
plot(tr,vertex.size=10,vertex.label=NA)
```

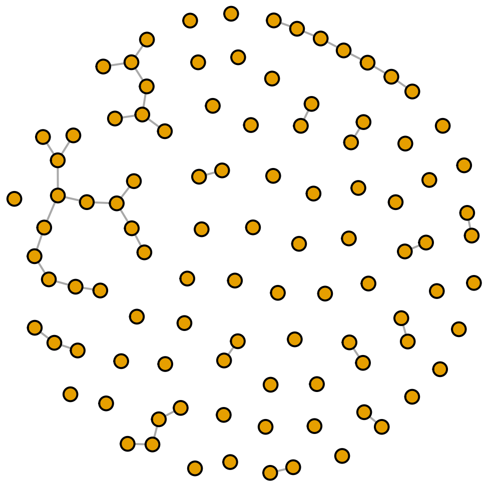


We created a tree graph with 40 vertices and 3 children per node using the `make_tree` function. The `mode` argument is set to “undirected” to create an undirected tree graph. The `plot` function is used to visualize the graph, and we can see that the vertices are represented as circles with a size of 10 and no labels.

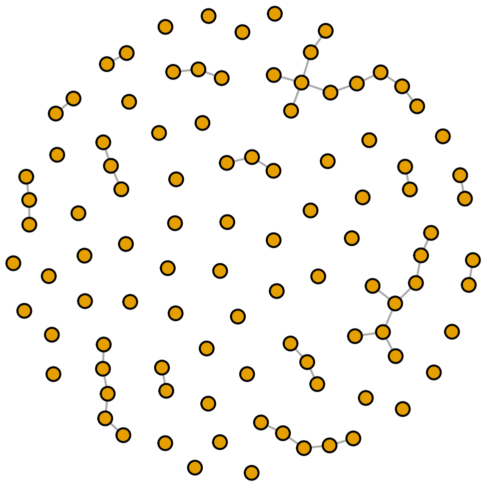
```r
rn <- make_ring(40)
plot(rn,vertex.size=10,vertex.label=NA)
```



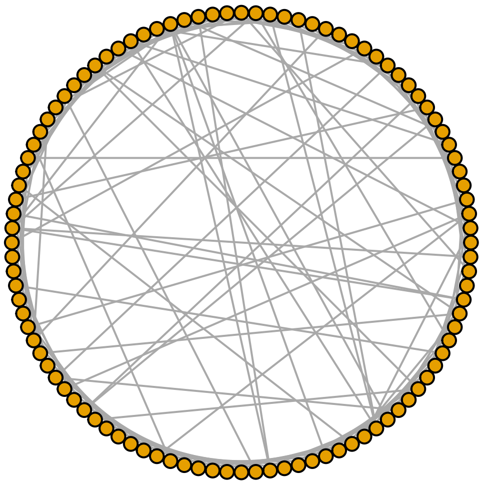
```r
er <- sample_gnm(n=100,m=40)
plot(er,vertex.size=6,vertex.label=NA)
```



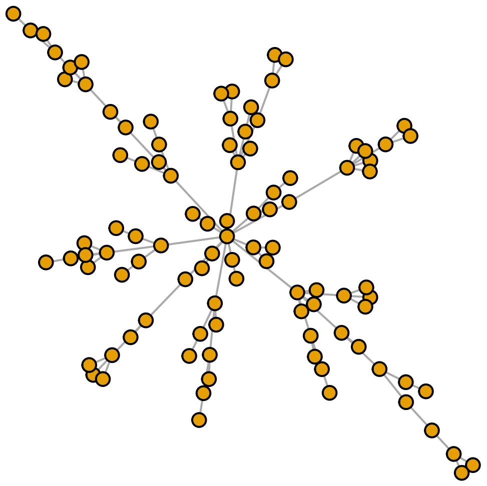
```
er <- sample_gnm(n=100,m=40)
plot(er,vertex.size=6,vertex.label=NA)
```



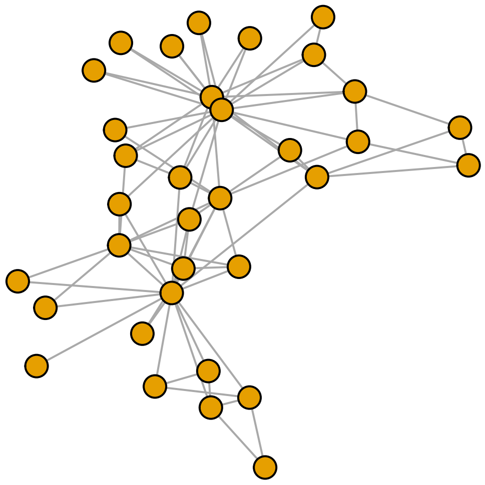
```
sw <- sample_smallworld(dim=2,size=10,nei=1,p=0.1)
plot(sw,vertex.size=6,vertex.label=NA,layout=layout_in_circle)
```



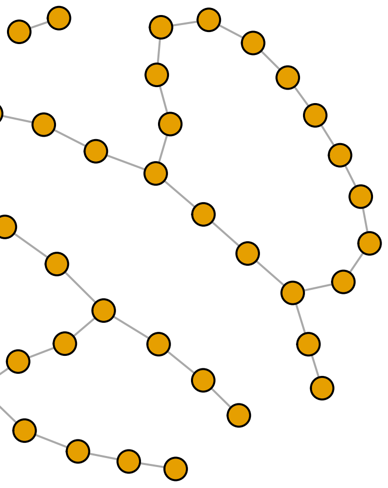
```
ba <- sample_pa(n=100,power=1,m=1,directed=F)
plot(ba,vertex.size=6,vertex.label=NA)
```



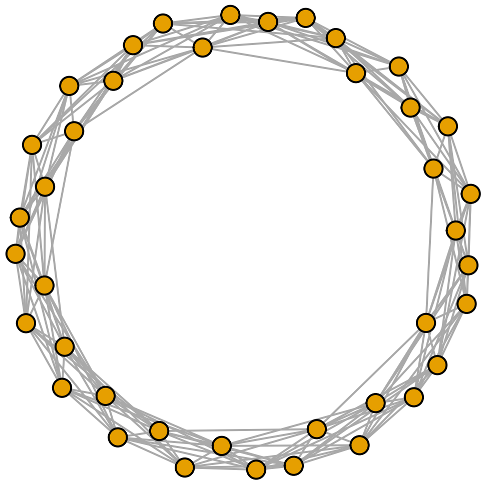
```
zach <- graph("Zachary")
plot(zach,vertex.size=10,vertex.label=NA)
```



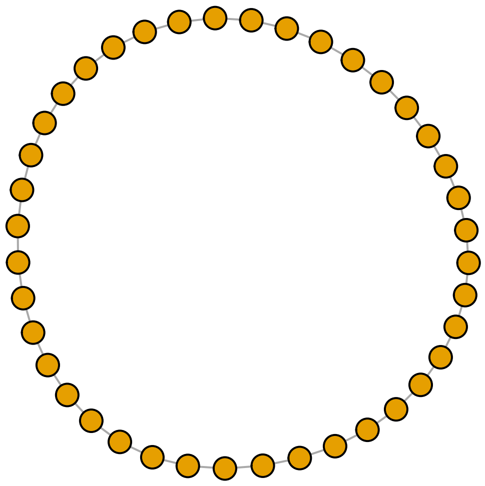
```
rn.rewired <- rewire(rn,each_edge(prob=0.1))
plot(rn.rewired,vertex.size=10,vertex.label=NA)
```



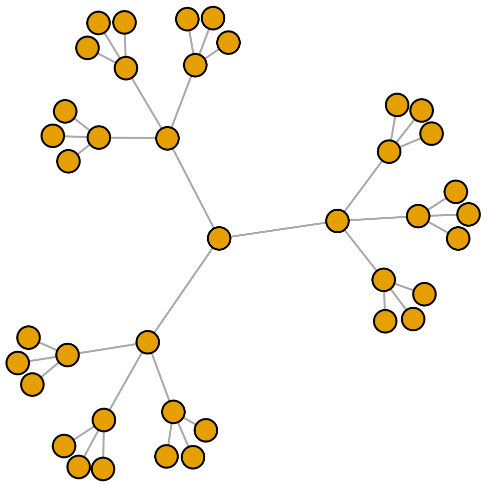
```
rn.neigh=connect(rn,5)
plot(rn.neigh,vertex.size=8,vertex.label=NA)
```



```r
plot(rn,vertex.size=10,vertex.label=NA)
```

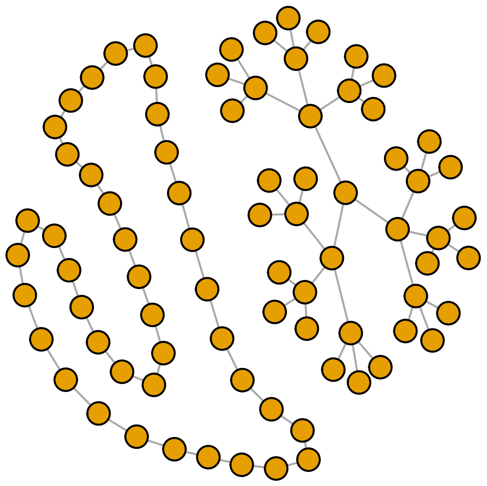


```r
plot(tr,vertex.size=10,vertex.label=NA)
```



Here we plotted the tree graph `tr` with 40 vertices and 3 children per node using the `plot` function. The `vertex.size` argument is set to 10, and the `vertex.label` argument is set to `NA` to hide the vertex labels.

```
plot(rn %du% tr,vertex.size=10,vertex.label=NA)
```



Here we plotted the union of the ring graph `rn` and the tree graph `tr` using the `%du%` operator. The `plot` function is used to visualize the graph, and we can see that the vertices are represented as circles with a size of 10 and no labels.

Reading network data from files

```
nodes <- read.csv("Dataset1-Media-Example-NODES.csv", header=T, as.is=T)
links <- read.csv("Dataset1-Media-Example-EDGES.csv", header=T, as.is=T)
```

Examine the data:

```
head(nodes)
```

```
##    id               media media.type type.label audience.size
## 1 s01            NY Times          1  Newspaper            20
## 2 s02     Washington Post          1  Newspaper            25
## 3 s03 Wall Street Journal          1  Newspaper            30
## 4 s04           USA Today          1  Newspaper            32
## 5 s05            LA Times          1  Newspaper            20
## 6 s06       New York Post          1  Newspaper            50
```

```
head(links)
```

```
##   from  to weight      type
## 1  s01 s02     10 hyperlink
## 2  s01 s02     12 hyperlink
## 3  s01 s03     22 hyperlink
## 4  s01 s04     21 hyperlink
## 5  s04 s11     22   mention
## 6  s05 s15     21   mention
```

```
nrow(nodes); length(unique(nodes$id))
```

```
## [1] 17
```

```
## [1] 17
```

```
nrow(links); nrow(unique(links[,c("from", "to")]))
```

```
## [1] 52
```

```
## [1] 49
```

There are more links than unique from-to combinations. That means we have cases in the data where there are multiple links between the same two nodes. We will collapse all links of the same type between the same two nodes by summing their weights, using aggregate() by "from", "to", & "type". We don't use simplify() here so as not to collapse different link types.

```
links <- aggregate(links[,3], links[,-3], sum)
links <- links[order(links$from, links$to),]
colnames(links)[4] <- "weight"
rownames(links) <- NULL

head(links)
```

```
##   from  to      type weight
## 1  s01 s02 hyperlink     22
## 2  s01 s03 hyperlink     22
## 3  s01 s04 hyperlink     21
## 4  s01 s15   mention     20
## 5  s02 s01 hyperlink     23
## 6  s02 s03 hyperlink     21
```

Data set 2: matrix

Two-mode or bipartite graphs have two different types of actors and links that go across, but not within each type. Second media example is a network of that kind, examining links between news sources and their consumers.

```
nodes2 <- read.csv("Dataset2-Media-User-Example-NODES.csv", header=T, as.is=T)
links2 <- read.csv("Dataset2-Media-User-Example-EDGES.csv", header=T, row.names=1)

head(nodes2)
```

```
##    id   media media.type media.name audience.size
## 1 s01     NYT          1  Newspaper            20
## 2 s02    WaPo          1  Newspaper            25
## 3 s03     WSJ          1  Newspaper            30
## 4 s04    USAT          1  Newspaper            32
## 5 s05 LATimes          1  Newspaper            20
## 6 s06     CNN          2         TV            56
```

```
head(links2)
```

```
##     U01 U02 U03 U04 U05 U06 U07 U08 U09 U10 U11 U12 U13 U14 U15 U16 U17 U18 U19
## s01   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## s02   0   0   0   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## s03   0   0   0   0   0   1   1   1   1   0   0   0   0   0   0   0   0   0   0
## s04   0   0   0   0   0   0   0   0   1   1   1   0   0   0   0   0   0   0   0
## s05   0   0   0   0   0   0   0   0   0   0   1   1   1   0   0   0   0   0   0
## s06   0   0   0   0   0   0   0   0   0   0   0   0   1   1   0   0   1   0   0
##     U20
## s01   0
## s02   1
## s03   0
## s04   0
## s05   0
## s06   0
```

We can see that links2 is an adjacency matrix for a two-mode network

```
links2 <- as.matrix(links2)
dim(links2)
```

```
## [1] 10 20
```

```
dim(nodes2)
```

```
## [1] 30  5
```

Turning networks into igraph object

We start by converting the raw data to an igraph network object. Here we use igraph's graph.data.frame function, which takes two data frames: d and vertices.

```
net <- graph_from_data_frame(d=links, vertices=nodes, directed=T)
class(net)
```

```
## [1] "igraph"
```

```
net
```

```
## IGRAPH 19f0a05 DNW- 17 49 --
## + attr: name (v/c), media (v/c), media.type (v/n), type.label (v/c),
## | audience.size (v/n), type (e/c), weight (e/n)
## + edges from 19f0a05 (vertex names):
##  [1] s01->s02 s01->s03 s01->s04 s01->s15 s02->s01 s02->s03 s02->s09 s02->s10
##  [9] s03->s01 s03->s04 s03->s05 s03->s08 s03->s10 s03->s11 s03->s12 s04->s03
## [17] s04->s06 s04->s11 s04->s12 s04->s17 s05->s01 s05->s02 s05->s09 s05->s15
## [25] s06->s06 s06->s16 s06->s17 s07->s03 s07->s08 s07->s10 s07->s14 s08->s03
## [33] s08->s07 s08->s09 s09->s10 s10->s03 s12->s06 s12->s13 s12->s14 s13->s12
## [41] s13->s17 s14->s11 s14->s13 s15->s01 s15->s04 s15->s06 s16->s06 s16->s17
## [49] s17->s04
```

```
E(net)
```

```
## + 49/49 edges from 19f0a05 (vertex names):
##  [1] s01->s02 s01->s03 s01->s04 s01->s15 s02->s01 s02->s03 s02->s09 s02->s10
##  [9] s03->s01 s03->s04 s03->s05 s03->s08 s03->s10 s03->s11 s03->s12 s04->s03
## [17] s04->s06 s04->s11 s04->s12 s04->s17 s05->s01 s05->s02 s05->s09 s05->s15
## [25] s06->s06 s06->s16 s06->s17 s07->s03 s07->s08 s07->s10 s07->s14 s08->s03
## [33] s08->s07 s08->s09 s09->s10 s10->s03 s12->s06 s12->s13 s12->s14 s13->s12
## [41] s13->s17 s14->s11 s14->s13 s15->s01 s15->s04 s15->s06 s16->s06 s16->s17
## [49] s17->s04
```

```
# The edges of the "net" object
V(net)
```

```
## + 17/17 vertices, named, from 19f0a05:
##  [1] s01 s02 s03 s04 s05 s06 s07 s08 s09 s10 s11 s12 s13 s14 s15 s16 s17
```

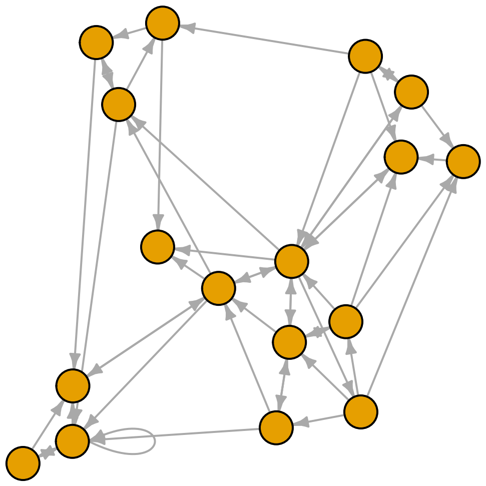
```
# The vertices of the "net" object
E(net)$type # Edge attribute "type"
```

```
##  [1] "hyperlink" "hyperlink" "hyperlink" "mention"   "hyperlink" "hyperlink"
##  [7] "hyperlink" "hyperlink" "hyperlink" "hyperlink" "hyperlink" "hyperlink"
## [13] "mention"   "hyperlink" "hyperlink" "hyperlink" "mention"   "mention"
## [19] "hyperlink" "mention"   "mention"   "hyperlink" "hyperlink" "mention"
## [25] "hyperlink" "hyperlink" "mention"   "mention"   "mention"   "hyperlink"
## [31] "mention"   "hyperlink" "mention"   "mention"   "mention"   "hyperlink"
## [37] "mention"   "hyperlink" "mention"   "hyperlink" "mention"   "mention"
## [43] "mention"   "hyperlink" "hyperlink" "hyperlink" "hyperlink" "mention"
## [49] "hyperlink"
```

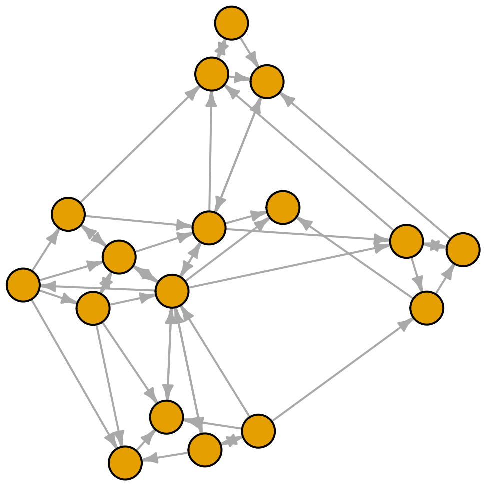
```
V(net)$media # Vertex attribute "media"
```

```
##  [1] "NY Times"              "Washington Post"     "Wall Street Journal"
##  [4] "USA Today"             "LA Times"            "New York Post"
##  [7] "CNN"                   "MSNBC"               "FOX News"
## [10] "ABC"                   "BBC"                 "Yahoo News"
## [13] "Google News"           "Reuters.com"         "NYTimes.com"
## [16] "WashingtonPost.com"    "AOL.com"
```

```r
plot(net, edge.arrow.size=.4,vertex.label=NA)
```



```r
net <- simplify(net, remove.multiple = F, remove.loops = T)
plot(net, edge.arrow.size=.4,vertex.label=NA)
```



```r
# Data set 2
head(nodes2)
```

```
##    id   media media.type media.name audience.size
## 1 s01     NYT          1  Newspaper            20
## 2 s02    WaPo          1  Newspaper            25
## 3 s03     WSJ          1  Newspaper            30
## 4 s04    USAT          1  Newspaper            32
## 5 s05 LATimes          1  Newspaper            20
## 6 s06     CNN          2         TV            56
```

```r
head(links2)
```

```
##     U01 U02 U03 U04 U05 U06 U07 U08 U09 U10 U11 U12 U13 U14 U15 U16 U17 U18 U19
## s01   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## s02   0   0   0   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## s03   0   0   0   0   0   1   1   1   1   0   0   0   0   0   0   0   0   0   0
## s04   0   0   0   0   0   0   0   0   1   1   1   0   0   0   0   0   0   0   0
## s05   0   0   0   0   0   0   0   0   0   0   1   1   1   0   0   0   0   0   0
## s06   0   0   0   0   0   0   0   0   0   0   0   0   1   1   0   0   1   0   0
##     U20
## s01   0
## s02   1
## s03   0
## s04   0
## s05   0
## s06   0
```

```
net2 <- graph_from_biadjacency_matrix(links2)
table(V(net2)$type)
```

```
##
## FALSE  TRUE
##    10    20
```

```
net2.bp <- bipartite_projection(net2)
```

We generate bipartite projections for the two-mode network using the bipartite_projection function. This function creates two separate networks, one for each type of actor, and we can analyze them separately.

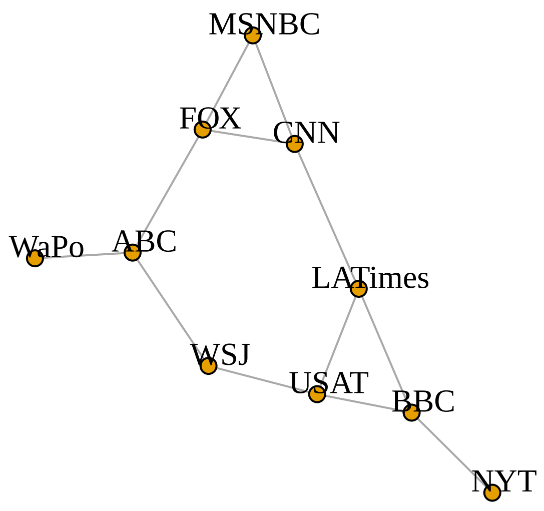
```
as_biadjacency_matrix(net2) %*% t(as_biadjacency_matrix(net2))
```

```
##     s01 s02 s03 s04 s05 s06 s07 s08 s09 s10
## s01   3   0   0   0   0   0   0   0   0   1
## s02   0   3   0   0   0   0   0   0   1   0
## s03   0   0   4   1   0   0   0   0   1   0
## s04   0   0   1   3   1   0   0   0   0   1
## s05   0   0   0   1   3   1   0   0   0   1
## s06   0   0   0   0   1   3   1   1   0   0
## s07   0   0   0   0   0   1   3   1   0   0
## s08   0   0   0   0   0   1   1   4   1   0
## s09   0   1   1   0   0   0   0   1   3   0
## s10   1   0   0   1   1   0   0   0   0   2
```

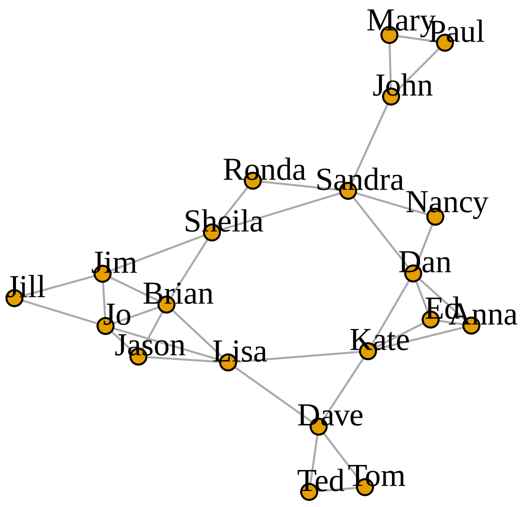
```
t(as_biadjacency_matrix(net2)) %*%
as_biadjacency_matrix(net2)
```

```
##     U01 U02 U03 U04 U05 U06 U07 U08 U09 U10 U11 U12 U13 U14 U15 U16 U17 U18 U19
## U01   2   1   1   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0
## U02   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## U03   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## U04   0   0   0   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## U05   0   0   0   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## U06   0   0   0   0   0   2   1   1   1   0   0   0   0   0   0   0   0   0   1
## U07   0   0   0   0   0   1   1   1   1   0   0   0   0   0   0   0   0   0   0
## U08   0   0   0   0   0   1   1   1   1   0   0   0   0   0   0   0   0   0   0
## U09   0   0   0   0   0   1   1   1   2   1   1   0   0   0   0   0   0   0   0
## U10   0   0   0   0   0   0   0   0   1   1   1   0   0   0   0   0   0   0   0
## U11   1   0   0   0   0   0   0   0   1   1   3   1   1   0   0   0   0   0   0
## U12   0   0   0   0   0   0   0   0   0   0   1   1   1   0   0   0   0   0   0
## U13   0   0   0   0   0   0   0   0   0   0   1   1   2   1   0   0   1   0   0
## U14   0   0   0   0   0   0   0   0   0   0   0   0   1   2   1   1   1   0   0
## U15   0   0   0   0   0   0   0   0   0   0   0   0   0   1   1   1   0   0   0
## U16   0   0   0   0   0   0   0   0   0   0   0   0   0   1   1   2   1   1   1
## U17   0   0   0   0   0   0   0   0   0   0   0   0   1   1   0   1   2   1   1
## U18   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   1   1   1
## U19   0    0    0    0    0    1    0    0    0    0    0    0    0    0    0    1    1    1    2
## U20   0    0    0    1    1    1    0    0    0    0    0    0    0    0    0    0    0    0    1
##     U20
## U01   0
## U02   0
## U03   0
## U04   1
## U05   1
## U06   1
## U07   0
## U08   0
## U09   0
## U10   0
## U11   0
## U12   0
## U13   0
## U14   0
## U15   0
## U16   0
## U17   0
## U18   0
## U19   1
## U20   2
```

```
plot(net2.bp$proj1, vertex.label.color="black", vertex.label.dist=1,
vertex.size=7, vertex.label=nodes2$media[!is.na(nodes2$media.type)])
```



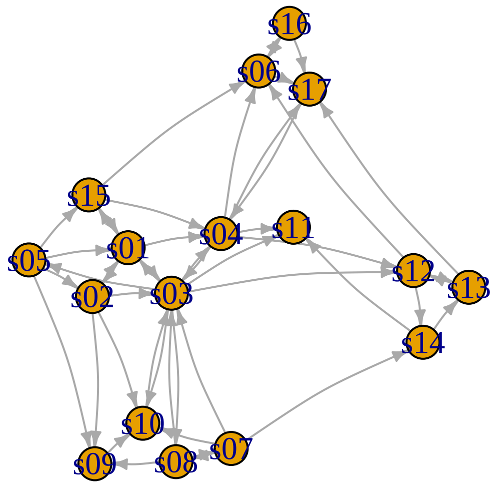
```r
plot(net2.bp$proj2, vertex.label.color="black", vertex.label.dist=1,
vertex.size=7, vertex.label=nodes2$media[ is.na(nodes2$media.type)])
```



We plotted the bipartite projections of the two-mode network using the `plot` function. The `vertex.label.color`, `vertex.label.dist`, `vertex.size`, and `vertex.label` arguments are used to customize the appearance of the graph. The `nodes2$media[!is.na(nodes2$media.type)]` and `nodes2$media[is.na(nodes2$media.type)]` arguments are used to label the vertices in the graph with the media names.

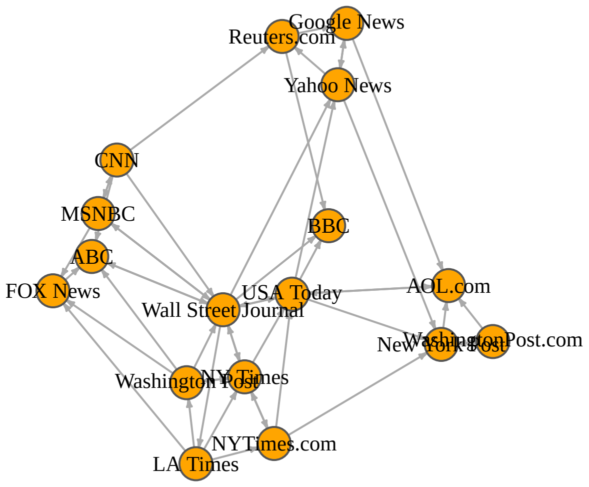
Plotting networks with igraph

```r
# Plot with curved edges (edge.curved=.1) and reduce arrow size:
plot(net, edge.arrow.size=.4, edge.curved=.1)
```



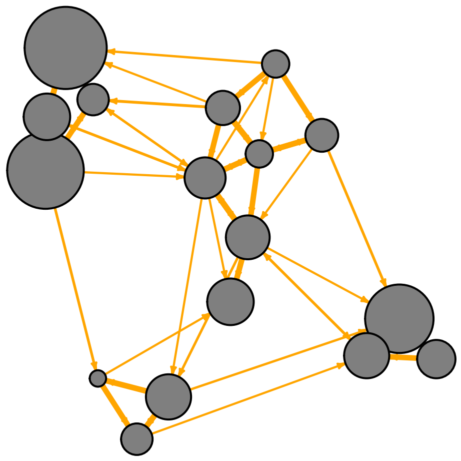
We plotted the network with curved edges using the `edge.curved` argument. The `edge.arrow.size` argument is set to 0.4 to reduce the size of the arrows.

```
# Set edge color to gray, and the node color to orange.
# Replace the vertex label with the node names stored in "media"
plot(net, edge.arrow.size=.2, edge.curved=0,
vertex.color="orange", vertex.frame.color="#555555",
vertex.label=V(net)$media, vertex.label.color="black",
vertex.label.cex=.7)
```



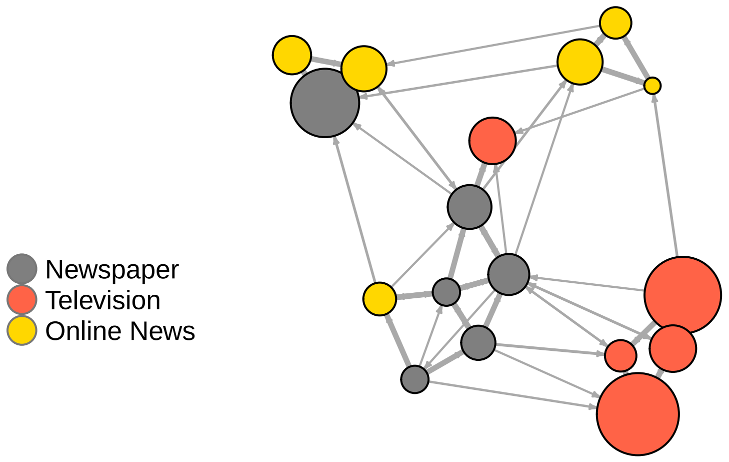
We plotted the network with gray edges and orange nodes using the `vertex.color` and `edge.color` arguments. The `vertex.frame.color` argument is set to "#555555" to set the frame color of the vertices. The `vertex.label` argument is used to label the vertices with the media names stored in the `media` attribute of the vertices. The `vertex.label.color`, `vertex.label.cex`, and `vertex.label.dist` arguments are used to customize the appearance of the vertex labels.

```
# Generate colors based on media type:
colrs <- c("gray50", "tomato", "gold")
V(net)$color <- colrs[V(net)$media.type]
# Set node size based on audience size:
V(net)$size <- V(net)$audience.size*0.7
# The labels are currently node IDs.
# Setting them to NA will render no labels:
V(net)$label.color <- "black"
V(net)$label <- NA
# Set edge width based on weight:
E(net)$width <- E(net)$weight/6
#change arrow size and edge color:
E(net)$arrow.size <- .2
E(net)$edge.color <- "gray80"
E(net)$width <- 1+E(net)$weight/12

plot(net, edge.color="orange", vertex.color="gray50")
```



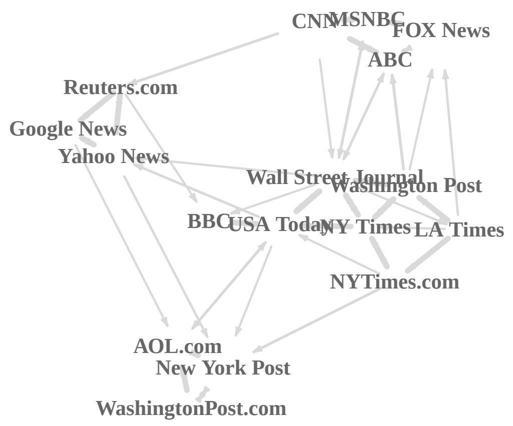
Here we generated colors based on the media type using the `colrs` vector. The `V(net)$color` argument is used to set the color of the vertices based on their media type. The `V(net)$size` argument is used to set the size of the vertices based on their audience size. The `V(net)$label.color`, `V(net)$label`, and `E(net)$width` arguments are used to customize the appearance of the vertex labels and edges. The `plot` function is used to visualize the graph with orange edges and gray vertices.

```
plot(net)
legend(x=-2.5, y=-0.1, c("Newspaper","Television", "Online News"), pch=21,
col="#777777", pt.bg=colrs, pt.cex=2, cex=.8, bty="n", ncol=1)
```



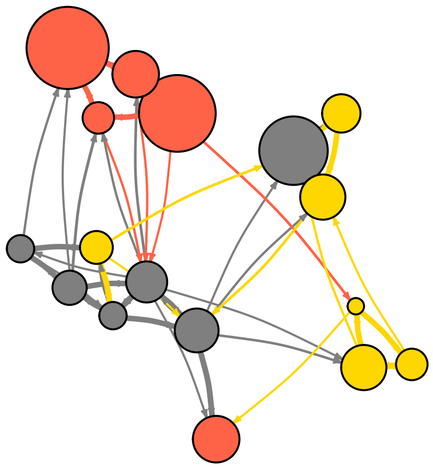
Here we add a legend to the plot using the `legend` function. The `x` and `y` arguments are used to set the position of the legend, and the `col`, `pt.bg`, `pt.cex`, `cex`, `bty`, and `ncol` arguments are used to customize the appearance of the legend. The `pch` argument is used to set the point character for the legend.

```r
plot(net, vertex.shape="none", vertex.label=V(net)$media,
vertex.label.font=2, vertex.label.color="gray40",
vertex.label.cex=.7, edge.color="gray85")
```



We plot only the vertex labels using the `vertex.shape` argument set to “none”. The `vertex.label.font`, `vertex.label.color`, and `vertex.label.cex` arguments are used to customize the appearance of the vertex labels. The `edge.color` argument is used to set the color of the edges.

```
edge.start <- ends(net, es=E(net), names=F)[,1]
edge.col <- V(net)$color[edge.start]
plot(net, edge.color=edge.col, edge.curved=.1)
```

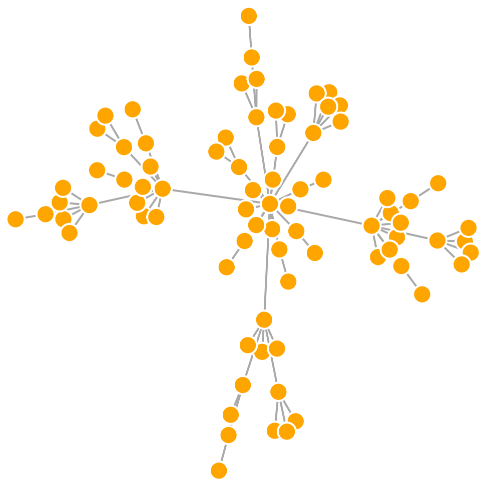


We plot the network with colored edges based on the starting vertex using the `ends` function to get the starting vertices of the edges. The `edge.color` argument is used to set the color of the edges based on the starting vertex color. The `edge.curved` argument is set to 0.1 to create curved edges.

Network layouts

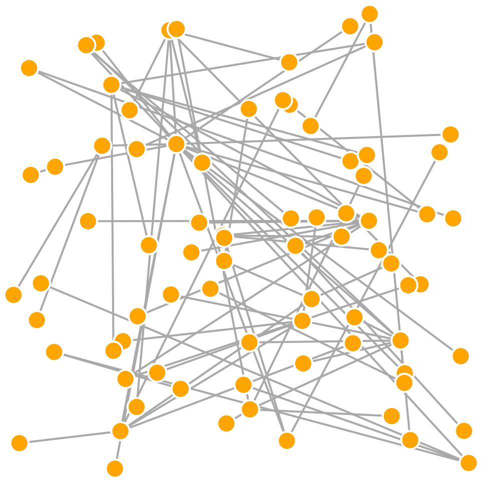
Network layouts are simply algorithms that return coordinates for each node in a network.

```
net.bg <- sample_pa(80)
V(net.bg)$size <- 8
V(net.bg)$frame.color <- "white"
V(net.bg)$color <- "orange"
V(net.bg)$label <- ""
E(net.bg)$arrow.mode <- 0
plot(net.bg)
```



Here we created a random graph using the `sample_pa` function with 80 vertices. The `V(net.bg)$size`, `V(net.bg)$frame.color`, `V(net.bg)$color`, and `V(net.bg)$label` arguments are used to customize the appearance of the vertices. The `E(net.bg)$arrow.mode` argument is set to 0 to remove the arrows from the edges.

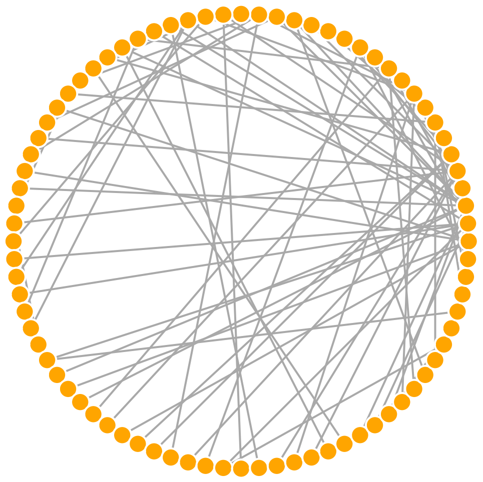
```
plot(net.bg, layout=layout_randomly)
```



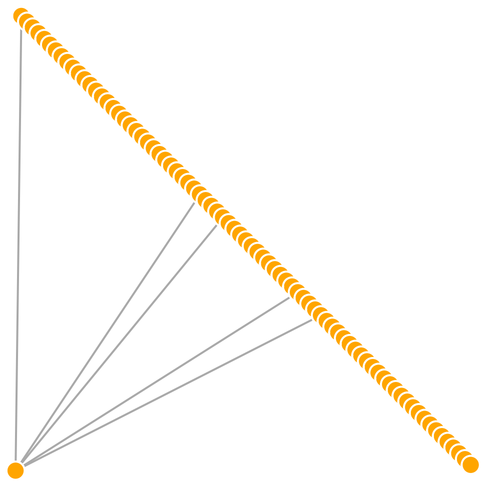
We plot the random graph using the `layout_randomly` function to generate a random layout for the vertices.

We can calculate the vertex coordinate in advance

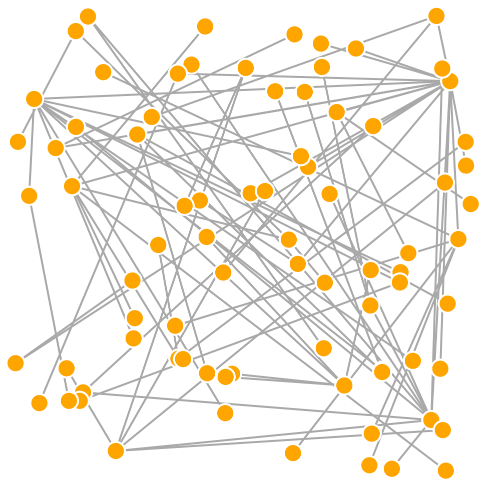
```r
l <- layout_in_circle(net.bg)
plot(net.bg, layout=l)
```



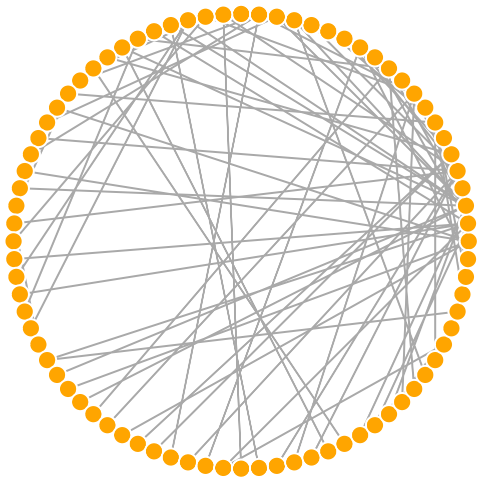
```
l <- cbind(1:vcount(net.bg), c(1, vcount(net.bg):2))
plot(net.bg, layout=l)
```



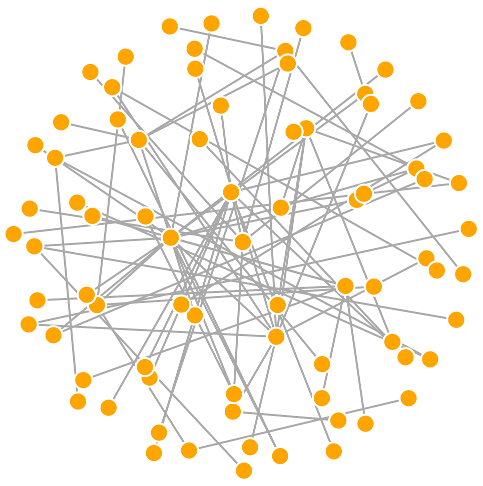
```r
# Randomly placed vertices
l <- layout_randomly(net.bg)
plot(net.bg, layout=l)
```



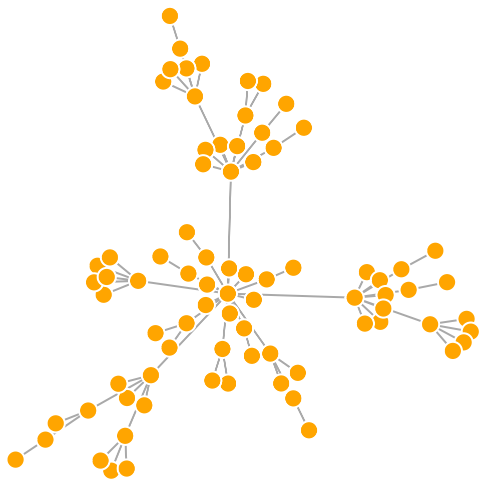
```
# Circle layout
l <- layout_in_circle(net.bg)
plot(net.bg, layout=l)
```



```r
# 3D sphere layout
l <- layout_on_sphere(net.bg)
plot(net.bg, layout=l)
```

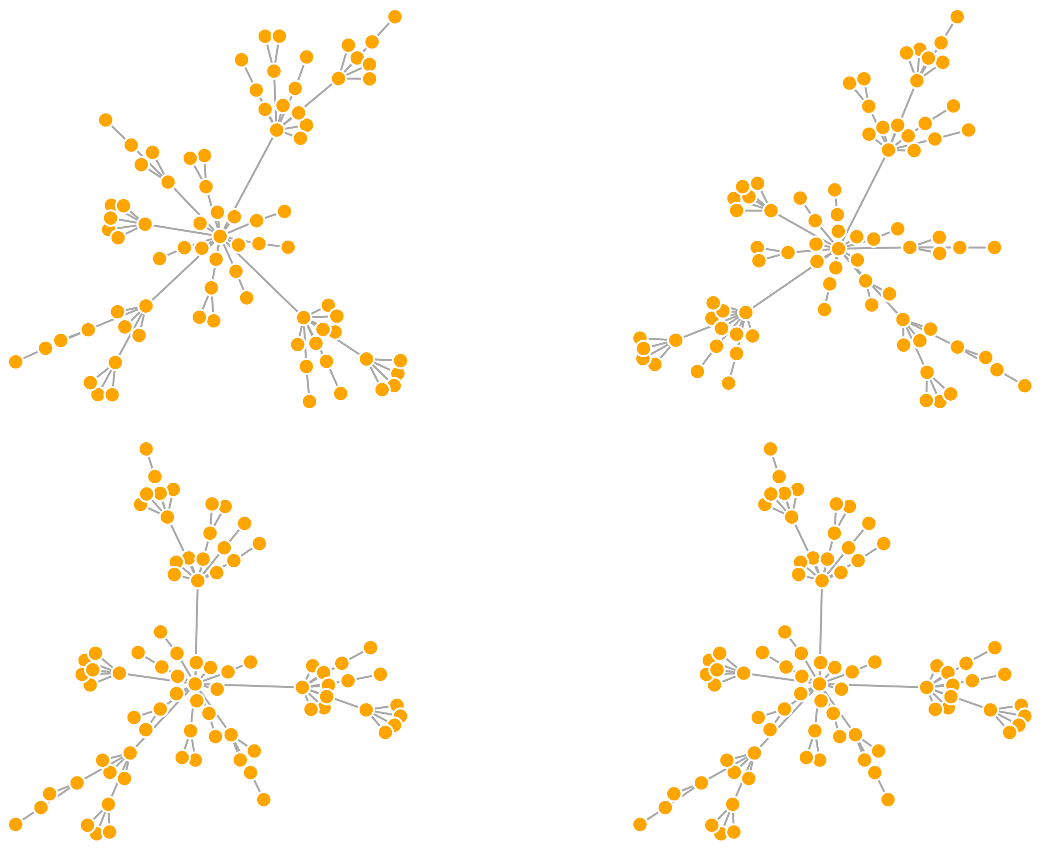


```r
l <- layout_with_fr(net.bg)
plot(net.bg, layout=l)
```



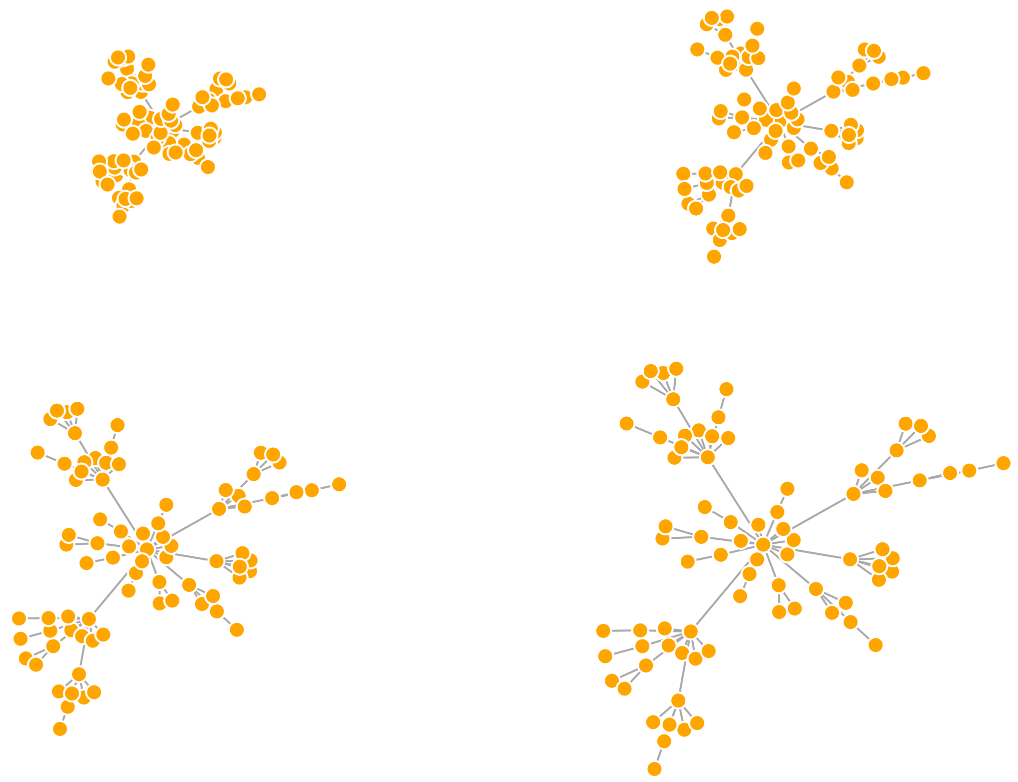
### Plot multilple networks in a single plot

```
par(mfrow=c(2,2), mar=c(0,0,0,0))
# plot four figures - 2 rows, 2 columns
plot(net.bg, layout=layout_with_fr)
plot(net.bg, layout=layout_with_fr)
plot(net.bg, layout=l)
plot(net.bg, layout=l)
```



Here we used the `par` function to set the layout of the plot to 2 rows and 2 columns using the `mfrow` argument. The `mar` argument is used to set the margins of the plot. The `plot` function is used to visualize the graph with different layouts.

```r
l <- layout_with_fr(net.bg)
l <- norm_coords(l, ymin=-1, ymax=1, xmin=-1, xmax=1)

par(mfrow=c(2,2), mar=c(0,0,0,0))
plot(net.bg, rescale=F, layout=l*0.4)
plot(net.bg, rescale=F, layout=l*0.6)
plot(net.bg, rescale=F, layout=l*0.8)
plot(net.bg, rescale=F, layout=l*1.0)
```



Here we used the `norm_coords` function to normalize the coordinates of the layout to fit within a specified range. The `rescale` argument is set to `FALSE` to prevent rescaling of the layout. The `layout` argument is multiplied by different factors to create different sizes of the graph.

All available layouts in igraph:

```r
layouts <- grep("^layout_", ls("package:igraph"), value=TRUE)[-1]
# Remove layouts that do not apply to our graph.
layouts <- layouts[!grepl("bipartite|merge|norm|sugiyama|tree", layouts)]
par(mfrow=c(3,3), mar=c(1,1,1,1))
for (layout in layouts) {
print(layout)
l <- do.call(layout, list(net))
plot(net, edge.arrow.mode=0, layout=l, main=layout) }
```

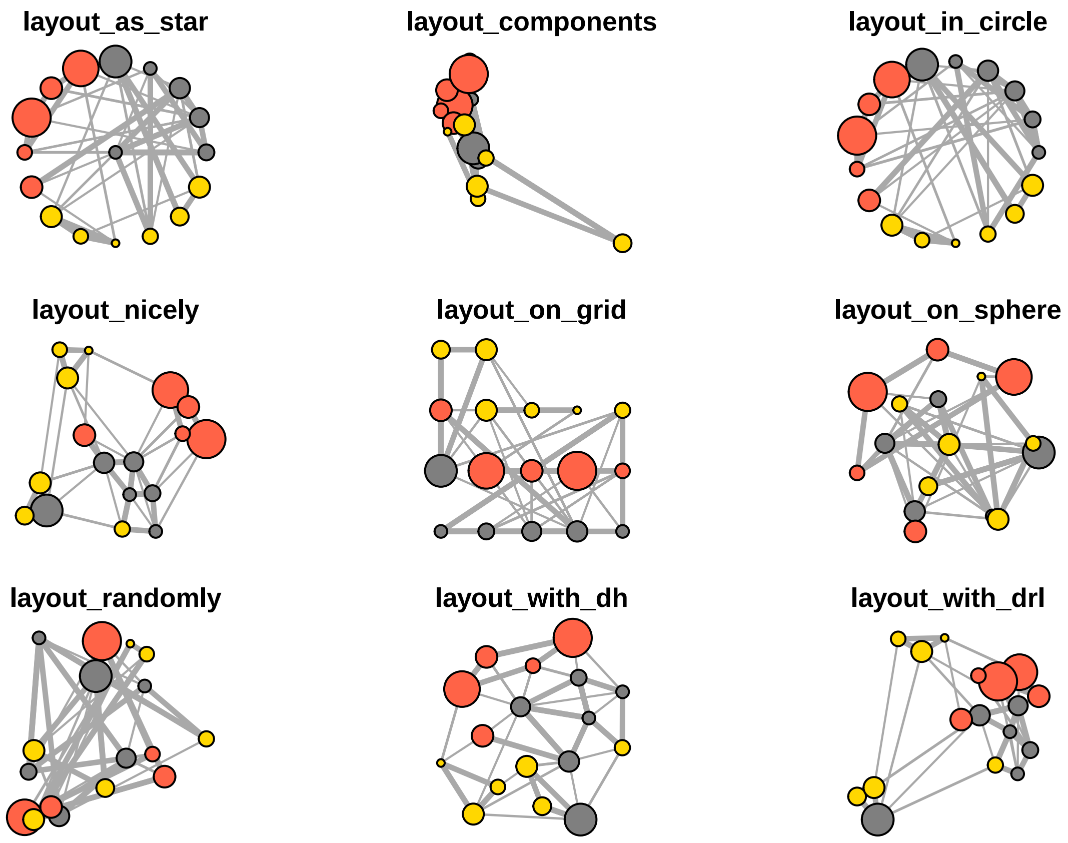
```
## [1] "layout_as_star"
```

```
## [1] "layout_components"
```

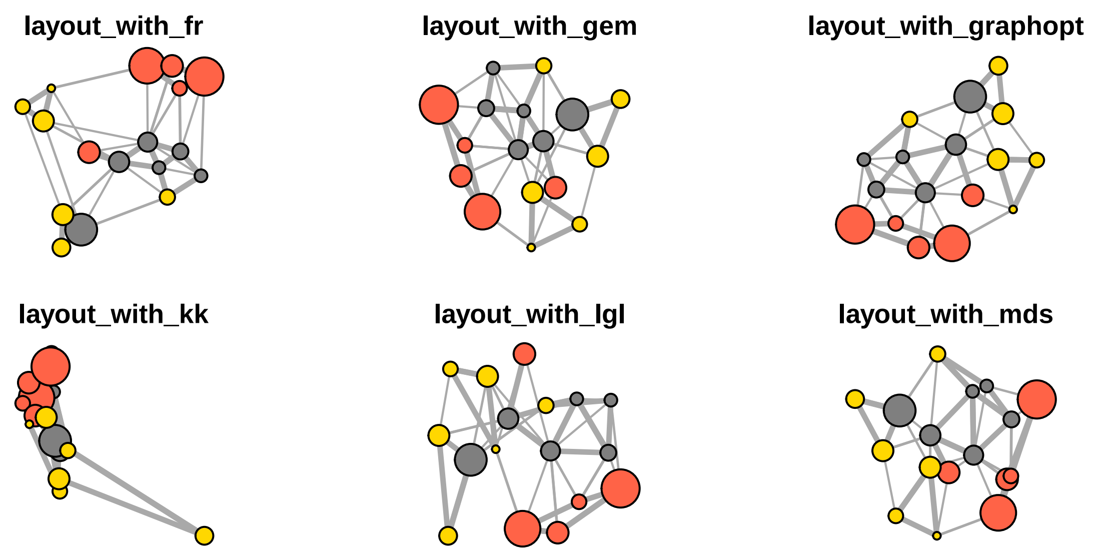
```
## [1] "layout_in_circle"
```

```
## [1] "layout_nicely"
```

```
## [1] "layout_on_grid"
## [1] "layout_on_sphere"

## [1] "layout_randomly"

## [1] "layout_with_dh"

## [1] "layout_with_drl"
```

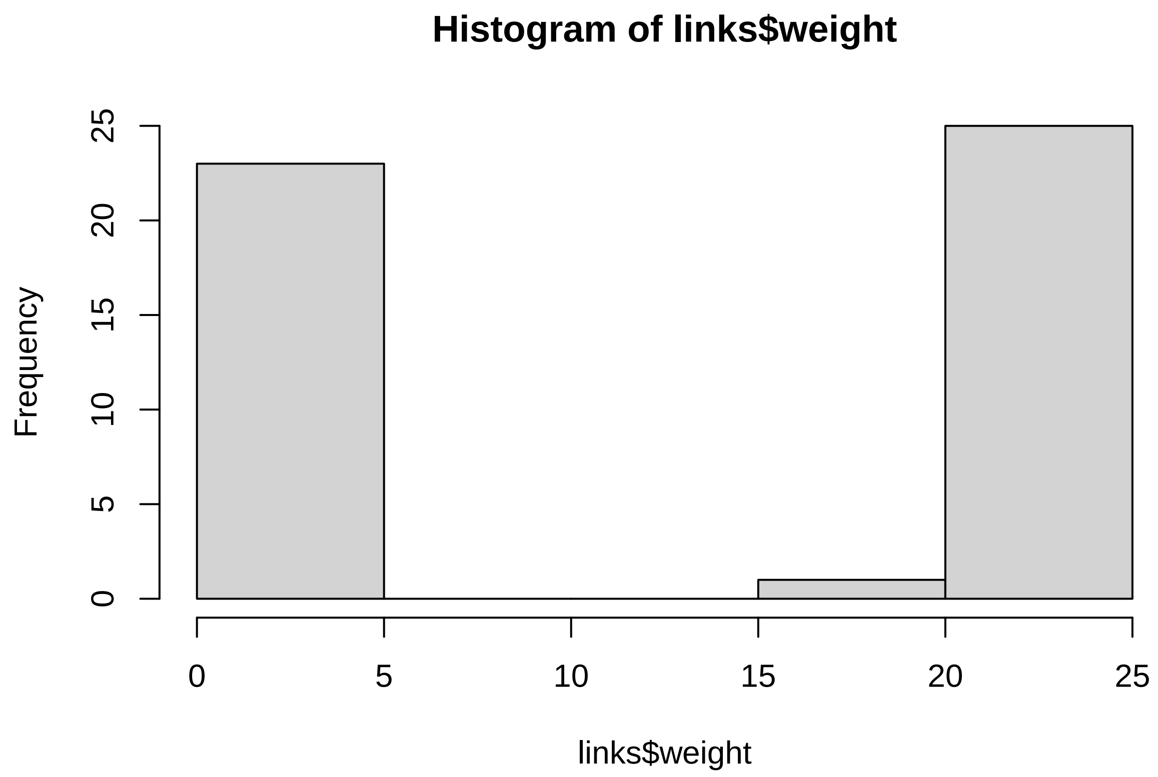


```
## [1] "layout_with_fr"

## [1] "layout_with_gem"

## [1] "layout_with_graphopt"

## [1] "layout_with_kk"

## [1] "layout_with_lgl"

## [1] "layout_with_mds"
```



We can identify the type and size of nodes, but cannot see much about the structure since the links we're examining are so dense. One way to approach this is to see if we can sparsify the network, keeping only the most important ties and discarding the rest.

```r
hist(links$weight)
```



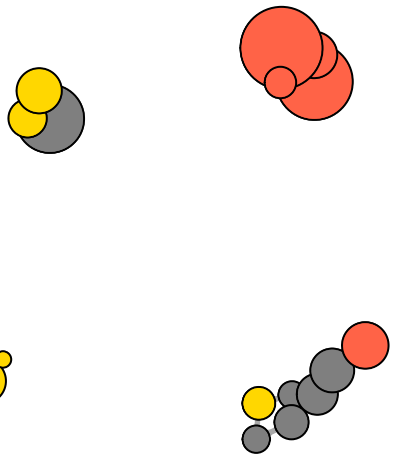
```
mean(links$weight)
```

```
## [1] 12.40816
```

```
sd(links$weight)
```

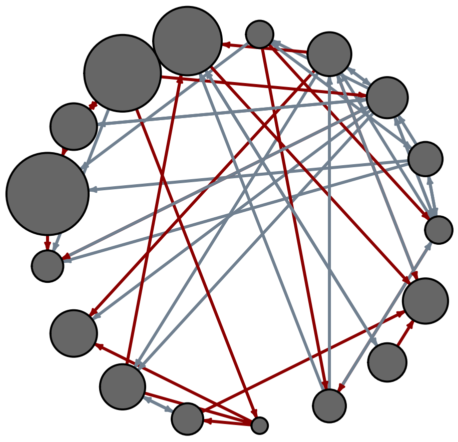
```
## [1] 9.905635
```

```
cut.off <- mean(links$weight)
net.sp <- delete_edges(net, E(net)[weight<cut.off])
plot(net.sp)
```



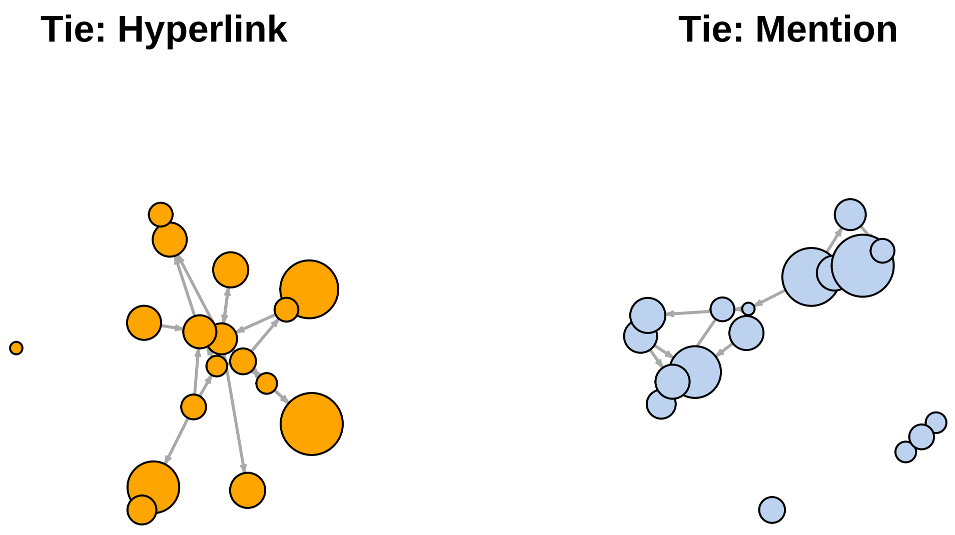
Here we calculated the mean and standard deviation of the edge weights using the `mean` and `sd` functions. The `cut.off` variable is set to the mean weight, and the `delete_edges` function is used to remove edges with weights less than the cut-off value. The `plot` function is used to visualize the sparsified network.

```
E(net)$width <- 1.5
plot(net, edge.color=c("dark red", "slategrey")[(E(net)$type=="hyperlink")+1],
vertex.color="gray40", layout=layout.circle)
```



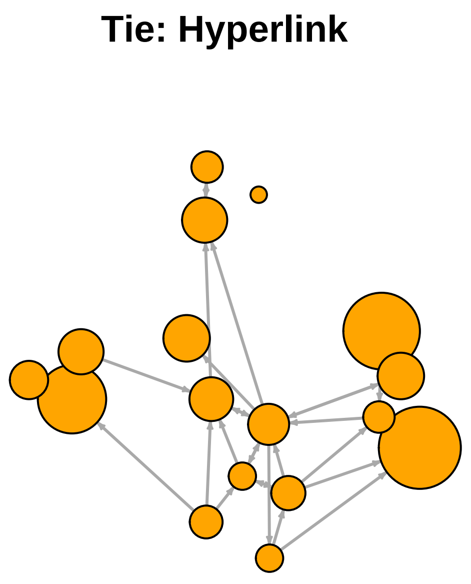
Here we set the edge width to 1.5 using the `E(net)$width` argument. The `plot` function is used to visualize the graph with dark red and slategrey edges based on the edge type. The `vertex.color` argument is set to "gray40" to set the color of the vertices, and the `layout` argument is set to `layout.circle` to create a circular layout for the graph.

```
net.m <- net - E(net)[E(net)$type=="hyperlink"] # another way to delete edges
net.h <- net - E(net)[E(net)$type=="mention"]
# Plot the two links separately:
par(mfrow=c(1,2))
plot(net.h, vertex.color="orange", main="Tie: Hyperlink")
plot(net.m, vertex.color="lightsteelblue2", main="Tie: Mention")
```



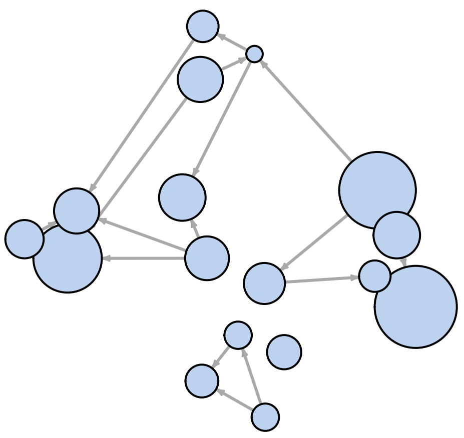
Here we created two separate networks, `net.m` and `net.h`, by removing edges of type "hyperlink" and "mention" respectively. The `plot` function is used to visualize the two networks separately with different vertex colors and titles.

```
# Make sure the nodes stay in place in both plots:
l <- layout_with_fr(net)
plot(net.h, vertex.color="orange", layout=l, main="Tie: Hyperlink")
```



```r
plot(net.m, vertex.color="lightsteelblue2", layout=l, main="Tie: Mention")
```

Tie: Mention

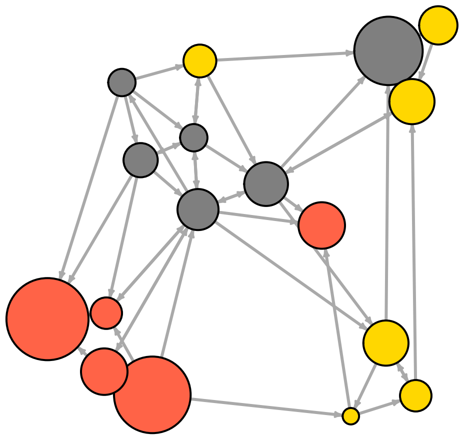


Here we used the `layout_with_fr` function to create a layout for the graph, and the `plot` function is used to visualize the two networks with the same layout. The `vertex.color` argument is used to set the color of the vertices, and the `main` argument is used to set the title of the plot.

Interactive plotting with tkplot

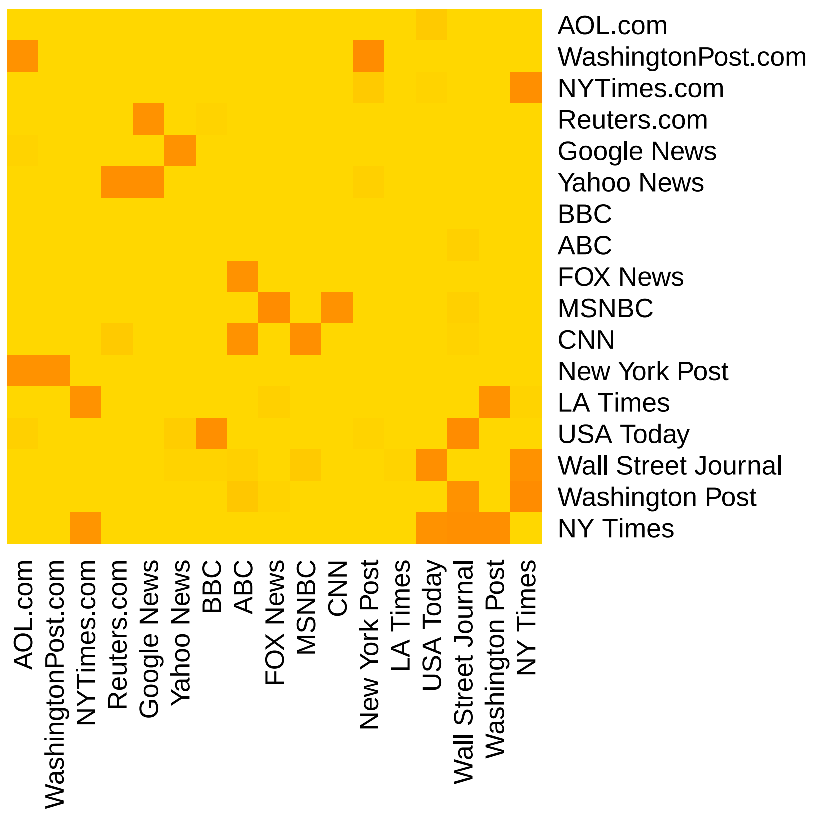
R and igraph allow for interactive plotting of networks. This might be a useful option for you if you want to tweak slightly the layout of a small graph. After adjusting the layout manually, you can get the coordinates of the nodes and use them for other plots.

```
tkid <- tkplot(net) #tkid is the id of the tkplot that will open
l <- tk_coords(tkid) # grab the coordinates from tkplot
tk_close(tkid, window.close = T)
plot(net, layout=l)
```



Other ways to represent a network

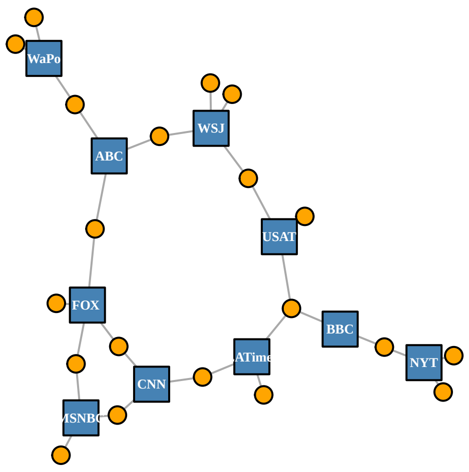
```r
netm <- as_adjacency_matrix(net, attr="weight", sparse=F)
colnames(netm) <- V(net)$media
rownames(netm) <- V(net)$media
palf <- colorRampPalette(c("gold", "dark orange"))
heatmap(netm[,17:1], Rowv = NA, Colv = NA, col = palf(100),
scale="none", margins=c(10,10) )
```



Here we converted the graph **net** to an adjacency matrix using the `as_adjacency_matrix` function. The `attr` argument is set to "weight" to include the edge weights in the matrix. The `colnames` and `rownames` functions are used to set the row and column names of the matrix to the media names. The `colorRampPalette` function is used to create a color palette for the heatmap, and the `heatmap` function is used to visualize the adjacency matrix as a heatmap with gold and dark orange colors.

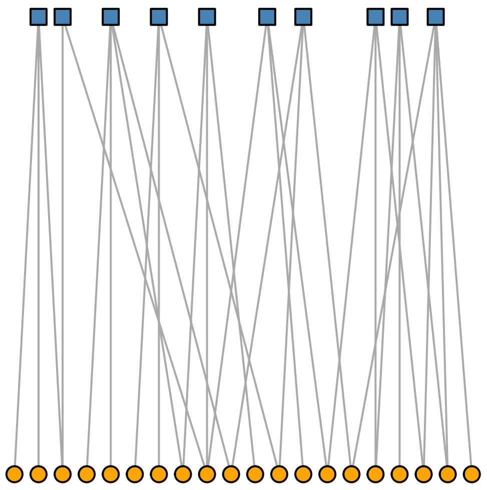
Plotting two-mode networks with igraph

```r
V(net2)$color <- c("steel blue", "orange")[V(net2)$type+1]
V(net2)$shape <- c("square", "circle")[V(net2)$type+1]
V(net2)$label <- ""
V(net2)$label[V(net2)$type==F] <- nodes2$media[V(net2)$type==F]
V(net2)$label.cex=.4
V(net2)$label.font=2
plot(net2, vertex.label.color="white", vertex.size=(2-V(net2)$type)*8)
```



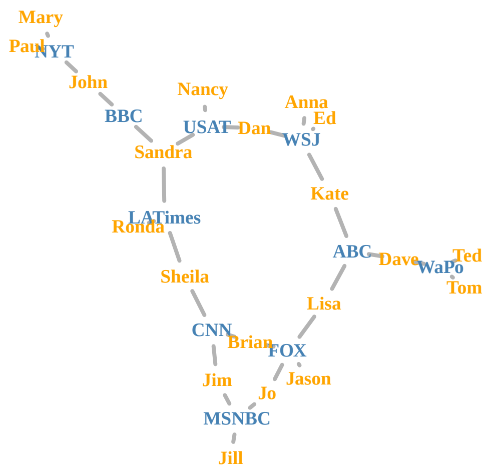
Here we set the vertex color and shape based on the vertex type using the `V(net2)$color` and `V(net2)$shape` arguments. The `V(net2)$label` argument is used to set the labels of the vertices, and the `V(net2)$label.cex` and `V(net2)$label.font` arguments are used to customize the appearance of the vertex labels. The `plot` function is used to visualize the graph with white vertex labels and different sizes based on the vertex type.

```r
plot(net2, vertex.label=NA, vertex.size=7, layout=layout_as_bipartite)
```



Here we plotted the two-mode network using the `layout_as_bipartite` function to create a bipartite layout for the vertices. The `vertex.label` argument is set to `NA` to hide the vertex labels, and the `vertex.size` argument is set to 7 to set the size of the vertices.

```
plot(net2, vertex.shape="none", vertex.label=nodes2$media,
vertex.label.color=V(net2)$color, vertex.label.font=2.5,
vertex.label.cex=.6, edge.color="gray70", edge.width=2)
```



Here we are using text as node labels, and the `vertex.shape` argument is set to "none" to hide the vertex shapes. The `vertex.label` argument is used to set the labels of the vertices to the media names, and the `vertex.label.color`, `vertex.label.font`, and `vertex.label.cex` arguments are used to customize the appearance of the vertex labels. The `edge.color` and `edge.width` arguments are used to set the color and width of the edges.

6. Network and node desctiptives

6.1 Density

It is the proportion of edges in the network to the maximum possible number of edges.

```
edge_density(net, loops = F)
```

```
## [1] 0.1764706
```

```
ecount(net)/(vcount(net)*(vcount(net)-1)) #for a directed network
```

```
## [1] 0.1764706
```

Here we calculated the edge density of the network using the `edge_density` function. The `loops` argument is set to `FALSE` to exclude loops from the calculation. The second line calculates the edge density manually by dividing the number of edges by the maximum possible number of edges in a directed network.

6.2 Reciprocity

Reciprocity is the proportion of edges in the network that are reciprocated.i.e The proportion of reciprocated ties (for a directed network).

```r
reciprocity(net)
```

```
## [1] 0.4166667
```

```r
dyad_census(net) # Mutual, asymmetric, and nyll node pairs
```

```
## $mut
## [1] 10
##
## $asym
## [1] 28
##
## $null
## [1] 98
```

```r
2*dyad_census(net)$mut/ecount(net) # Calculating reciprocity
```

```
## [1] 0.4166667
```

6.3 Transitivity

Transitivity is the proportion of triangles in the network to the number of connected triples. global - ratio of triangles to connected triples local - ratio of triangles to connected triples for each node

```r
transitivity(net, type="global") # net is treated as an undirected network
```

```
## [1] 0.372549
```

```r
transitivity(as_undirected(net, mode="collapse"))
```

```
## [1] 0.372549
```

```r
transitivity(net, type="local")
```

```
##       s01       s02       s03       s04       s05       s06       s07       s08 
## 0.6000000 0.6000000 0.2500000 0.3333333 0.5000000 0.4000000 0.3333333 0.3333333 
##       s09       s10       s11       s12       s13       s14       s15       s16 
## 0.3333333 0.5000000 0.3333333 0.3000000 0.3333333 0.1666667 0.5000000 1.0000000 
##       s17 
## 0.3333333
```

```r
triad_census(net) # for directed networks
```

```
##  [1] 244 241  80  13  11  27  15  22   4   1   8   4   4   3   3   0
```

Here we can observe that the transitivity of the network is 0.5, which indicates that the network has a moderate level of clustering. The `triad_census` function is used to count the number of different types of triads in the directed network.

6.4 Diameter

Diameter is the maximum distance between any two nodes in the network.

```r
diameter(net, directed=F, weights=NA)
```

```
## [1] 4
```

It means that the maximum distance between any two nodes in the network is 4. The `directed` argument is set to `FALSE` to treat the network as undirected, and the `weights` argument is set to `NA` to ignore edge weights.

```r
diameter(net, directed=T)
```

```
## [1] 75
```

```r
diam <- get_diameter(net, directed=T)
diam
```

```
## + 7/17 vertices, named, from d0bd4fb:
## [1] s12 s06 s17 s04 s03 s08 s07
```

We can see that the diameter of the directed network is 4, which indicates that the maximum distance between any two nodes in the directed network is 4. The `get_diameter` function is used to get the diameter of the directed network.

```r
class(diam)
```

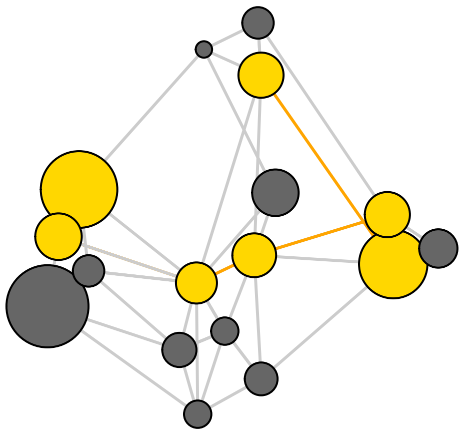
```
## [1] "igraph.vs"
```

```r
as.vector(diam)
```

```
## [1] 12  6 17  4  3  8  7
```

Color nodes along the diameter

```r
vcol <- rep("gray40", vcount(net))
vcol[diam] <- "gold"
ecol <- rep("gray80", ecount(net))
ecol[E(net, path=diam)] <- "orange"
# E(net, path=diam) finds edges along a path, here 'diam'
plot(net, vertex.color=vcol, edge.color=ecol, edge.arrow.mode=0)
```

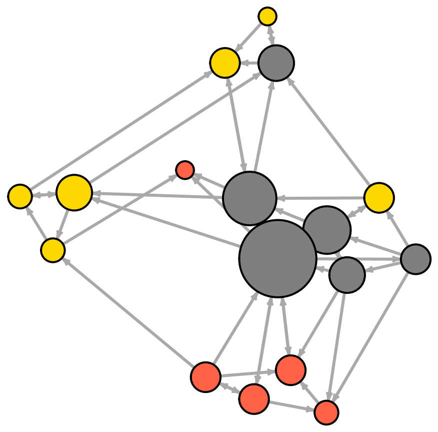


Here we colored the nodes along the diameter of the network using the `vcol` vector. The `ecol` vector is used to set the color of the edges along the diameter path. The `plot` function is used to visualize the graph with gold nodes and orange edges along the diameter path.

6.5 Node degree

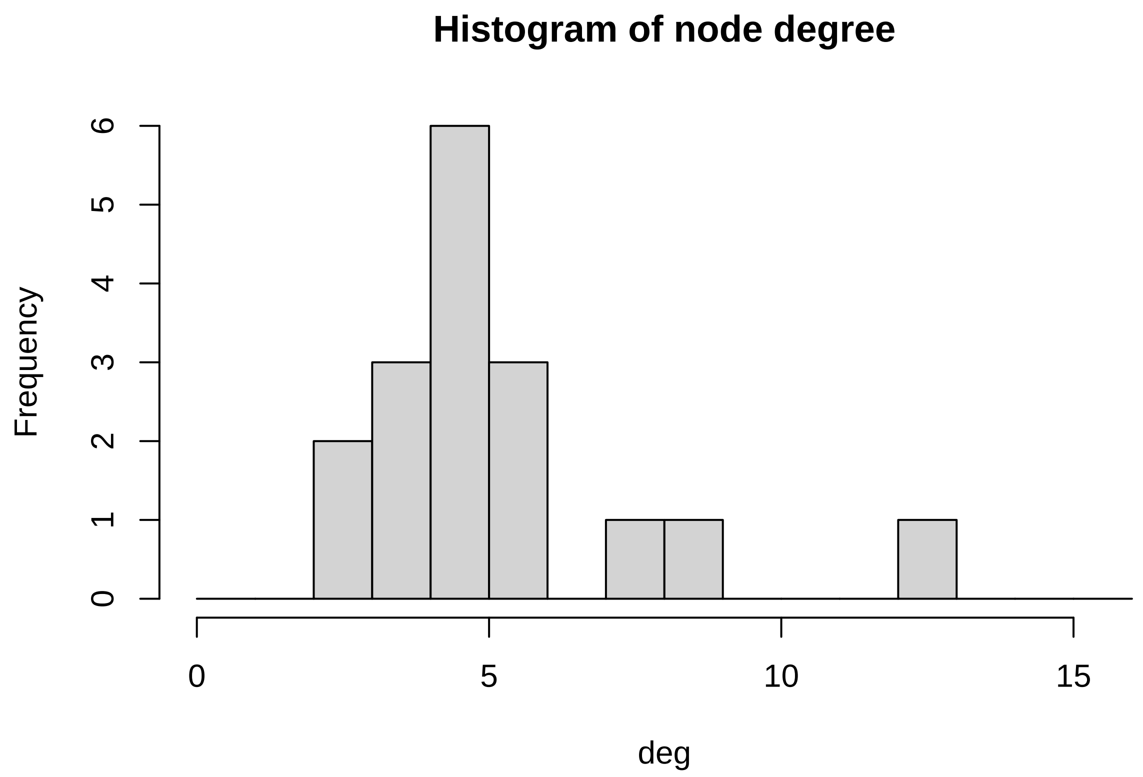
It is the number of edges connected to a node. It is the most basic measure of centrality.

```r
deg <- degree(net, mode="all")
plot(net, vertex.size=deg*3)
```



Here we calculated the degree of each node in the network using the `degree` function. The `mode` argument is set to "all" to calculate the degree for all nodes. The `plot` function is used to visualize the graph with vertex sizes proportional to their degrees.

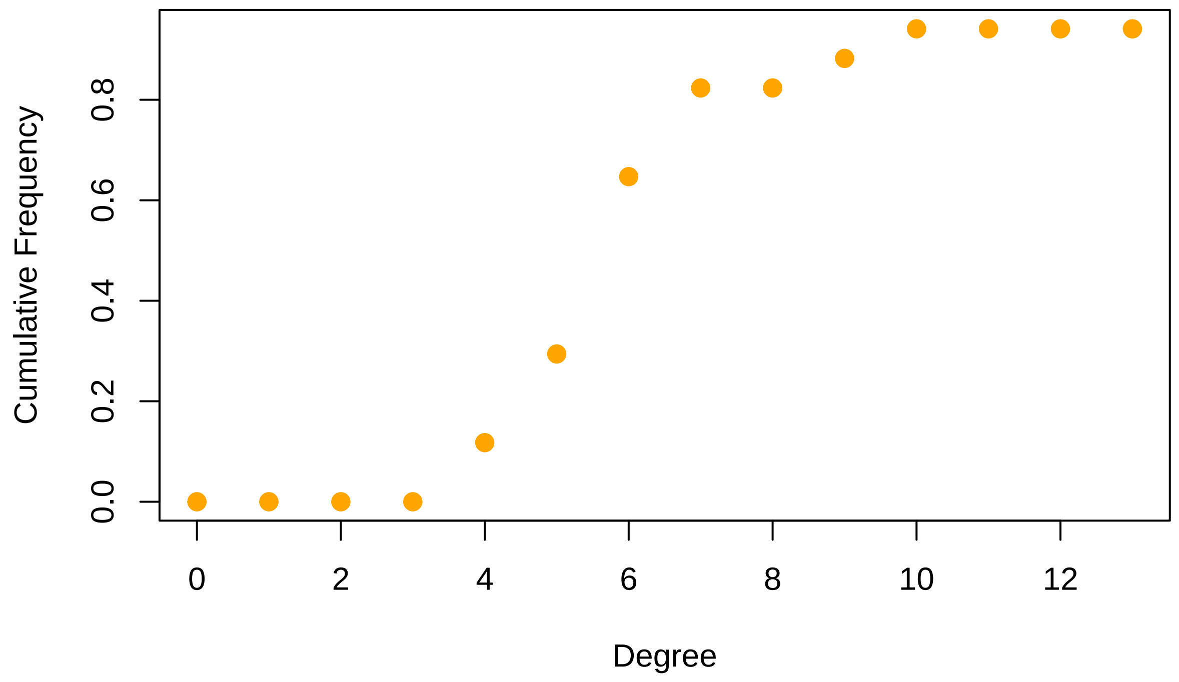
```
hist(deg, breaks=1:vcount(net)-1, main="Histogram of node degree")
```



The histogram shows that the degree distribution of the network is right-skewed, indicating that most nodes have a low degree while a few nodes have a high degree.

6.6 Degree distribution

```
deg.dist <- degree_distribution(net, cumulative=T, mode="all")
plot( x=0:max(deg), y=1-deg.dist, pch=19, cex=1.2, col="orange",
xlab="Degree", ylab="Cumulative Frequency")
```



The degree distribution plot shows the cumulative frequency of nodes with a degree less than or equal to a certain value. The `degree_distribution` function is used to calculate the degree distribution, and the `plot` function is used to visualize the cumulative frequency.

6.7 Centrality and centralization

Centrality is a measure of the importance of a node in a network. There are several measures of centrality, including degree centrality, closeness centrality, betweenness centrality, and eigenvector centrality.

Centralization is a measure of the extent to which a network is centered around a few nodes. It is calculated as the sum of the differences between the maximum centrality score and the centrality scores of all nodes in the network.

```
degree(net, mode="in")
```

Degree (number of ties)

```
## s01 s02 s03 s04 s05 s06 s07 s08 s09 s10 s11 s12 s13 s14 s15 s16 s17
##   4   2   6   4   1   4   1   2   3   4   3   3   2   2   2   1   4
```

```
centr_degree(net, mode="in", normalized=T)
```

```
## $res
##  [1] 4 2 6 4 1 4 1 2 3 4 3 3 2 2 2 1 4
##
## $centralization
## [1] 0.1985294
##
## $theoretical_max
## [1] 272
```

The centralization is 0.1985 . It means that the network is moderately centralized around a few nodes with high degree centrality. The `centr_degree` function is used to calculate the degree centrality of the nodes in the network, and the `normalized` argument is set to `TRUE` to normalize the centrality scores.

Closeness It is the average distance from a node to all other nodes in the network. It is calculated as the reciprocal of the sum of the distances from a node to all other nodes.

```
closeness(net, mode="all", weights=NA)
```

```
##        s01        s02        s03        s04        s05        s06        s07
## 0.03333333 0.03030303 0.04166667 0.03846154 0.03225806 0.03125000 0.03030303
##        s08        s09        s10        s11        s12        s13        s14
## 0.02857143 0.02564103 0.02941176 0.03225806 0.03571429 0.02702703 0.02941176
##        s15        s16        s17
## 0.03030303 0.02222222 0.02857143
```

```
centr_clo(net, mode="all", normalized=T)
```

```
## $res
##  [1] 0.5333333 0.4848485 0.6666667 0.6153846 0.5161290 0.5000000 0.4848485
##  [8] 0.4571429 0.4102564 0.4705882 0.5161290 0.5714286 0.4324324 0.4705882
## [15] 0.4848485 0.3555556 0.4571429
##
## $centralization
## [1] 0.3753596
##
## $theoretical_max
## [1] 7.741935
```

The closeness centrality of network is 0.375. It means that the network is moderately centralized around a few nodes with high closeness centrality. The `centr_clo` function is used to calculate the closeness centrality of the nodes in the network, and the `normalized` argument is set to `TRUE` to normalize the centrality scores.

Betweenness It is the number of times a node lies on the shortest path between two other nodes. It is calculated as the sum of the fraction of all-pairs shortest paths that pass through a node.

```
betweenness(net, directed=T, weights=NA)
```

```
##         s01         s02         s03         s04         s05         s06
##  24.0000000   5.8333333 127.0000000  93.5000000  16.5000000  20.3333333
##         s07         s08         s09         s10         s11         s12
##   1.8333333  19.5000000   0.8333333  15.0000000   0.0000000  33.5000000
##         s13         s14         s15         s16         s17
##  20.0000000   4.0000000   5.6666667   0.0000000  58.5000000
```

```r
edge_betweenness(net, directed=T, weights=NA)
```

```
##  [1] 10.833333 11.333333  8.333333  9.500000  4.000000 12.500000  3.000000
##  [8]  2.333333 24.000000 16.000000 31.500000 32.500000  9.500000  6.500000
## [15] 23.000000 65.333333 11.000000  6.500000 18.000000  8.666667  5.333333
## [22] 10.000000  6.000000 11.166667 15.000000 21.333333 10.000000  2.000000
## [29]  1.333333  4.500000 11.833333 16.833333  6.833333 16.833333 31.000000
## [36] 17.000000 18.000000 14.500000  7.500000 28.500000  3.000000 17.000000
## [43]  5.666667  9.666667  6.333333  1.000000 15.000000 74.500000
```

```r
centr_betw(net, directed=T, normalized=T)
```

```
## $res
##  [1]  24.0000000   5.8333333 127.0000000  93.5000000  16.5000000  20.3333333
##  [7]   1.8333333  19.5000000   0.8333333  15.0000000   0.0000000  33.5000000
## [13]  20.0000000   4.0000000   5.6666667   0.0000000  58.5000000
##
## $centralization
## [1] 0.4460938
##
## $theoretical_max
## [1] 3840
```

The centralization is 0.446. It shows that the network is moderately centralized around a few nodes with high betweenness centrality. The `centr_betw` function is used to calculate the betweenness centrality of the nodes in the network, and the `normalized` argument is set to `TRUE` to normalize the centrality scores.

Eigenvector It is a measure of the influence of a node in the network. It is calculated as the sum of the eigenvector centrality scores of all nodes that are connected to a node.

```r
eigen_centrality(net, directed=T, weights=NA)
```

```
## $vector
##       s01       s02       s03       s04       s05       s06       s07       s08
## 0.6638179 0.3314674 1.0000000 0.9133129 0.3326443 0.7468249 0.1244195 0.3740317
##       s09       s10       s11       s12       s13       s14       s15       s16
## 0.3453324 0.5991652 0.7334202 0.7519086 0.3470857 0.2915055 0.3314674 0.2484270
##       s17
## 0.7503292
##
## $value
## [1] 3.006215
##
## $options
## $options$bmat
## [1] "I"
##
## $options$n
## [1] 17
##
## $options$which
## [1] "LR"
##
## $options$nev
## [1] 1
##
## $options$tol
## [1] 0
##
## $options$ncv
## [1] 17
##
## $options$ldv
## [1] 0
##
## $options$ishift
## [1] 1
##
## $options$maxiter
## [1] 3000
##
## $options$nb
## [1] 1
##
## $options$mode
## [1] 1
##
## $options$start
## [1] 1
##
## $options$sigma
## [1] 0
##
## $options$sigmai
## [1] 0
##
## $options$info
## [1] 0
##
## $options$iter
## [1] 1
##
## $options$nconv
## [1] 1
##
## $options$numop
## [1] 17
##
## $options$numopb
## [1] 0
##
## $options$numreo
## [1] 11
```

```r
centr_eigen(net, directed=T, normalized=T)
```

```
## $vector
##  [1] 0.6638179 0.3314674 1.0000000 0.9133129 0.3326443 0.7468249 0.1244195
##  [8] 0.3740317 0.3453324 0.5991652 0.7334202 0.7519086 0.3470857 0.2915055
## [15] 0.3314674 0.2484270 0.7503292
## 
## $value
## [1] 3.006215
## 
## $options
## $options$bmat
## [1] "I"
## 
## $options$n
## [1] 17
## 
## $options$which
## [1] "LR"
## 
## $options$nev
## [1] 1
## 
## $options$tol
## [1] 0
## 
## $options$ncv
## [1] 17
## 
## $options$ldv
## [1] 0
## 
## $options$ishift
## [1] 1
## 
## $options$maxiter
## [1] 3000
## 
## $options$nb
## [1] 1
## 
## $options$mode
## [1] 1
## 
## $options$start
## [1] 1
## 
## $options$sigma
## [1] 0
## 
## $options$sigmai
## [1] 0
## 
```

```
## $options$info
## [1] 0
##
## $options$iter
## [1] 1
##
## $options$nconv
## [1] 1
##
## $options$numop
## [1] 17
##
## $options$numopb
## [1] 0
##
## $options$numreo
## [1] 11
##
##
## $centralization
## [1] 0.5071775
##
## $theoretical_max
## [1] 16
```

6.8 Hubs and authorities

Hubs and authorities are measures of the importance of nodes in a directed network. Hubs are nodes that point to many other nodes, while authorities are nodes that are pointed to by many other nodes.

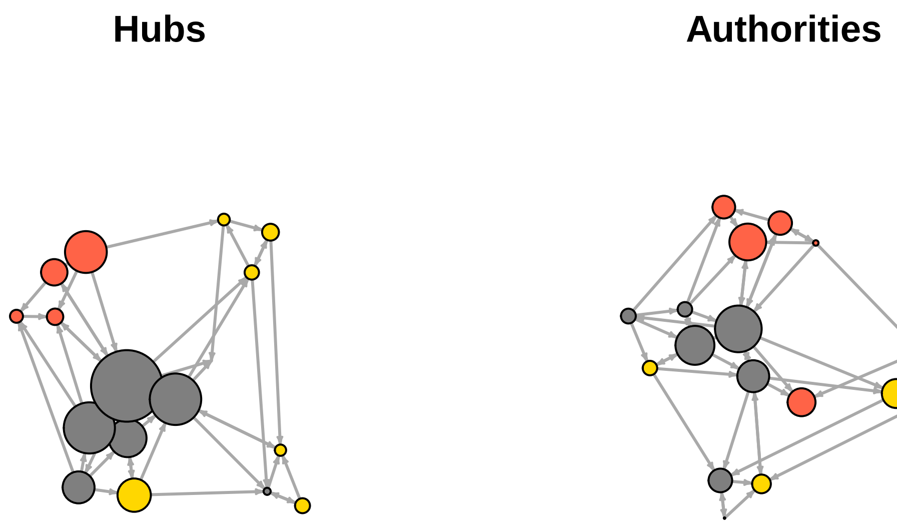
```
hs <- hub_score(net, weights=NA)$vector
```

```
## Warning: `hub_score()` was deprecated in igraph 2.0.3.
## i Please use `hits_scores()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
as <- authority_score(net, weights=NA)$vector
```

```
## Warning: `authority_score()` was deprecated in igraph 2.1.0.
## i Please use `hits_scores()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
par(mfrow=c(1,2))
plot(net, vertex.size=hs*50, main="Hubs")
plot(net, vertex.size=as*30, main="Authorities")
```



7. Distance and paths

Average path length is the average number of edges in the shortest path between all pairs of nodes in the network. It is calculated as the sum of the lengths of all shortest paths divided by the number of pairs of nodes.

```
no_directed <- mean_distance(net, directed=F)
print(no_directed)
```

```
## [1] 6.948529
```

```
directed_mean <-mean_distance(net, directed=T)
print(directed_mean)
```

```
## [1] 27.55078
```

We can also find the length of all shortest paths in the graph:

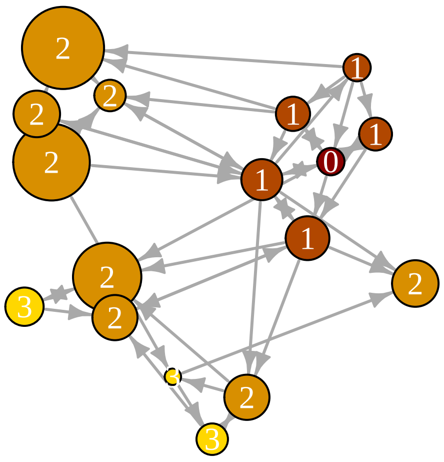
```
distances(net) # with edge weights
```

```
##     s01 s02 s03 s04 s05 s06 s07 s08 s09 s10 s11 s12 s13 s14 s15 s16 s17
## s01   0   4   2   6   1   5   3   4   3   4   3   3   9   4   7  26   8
## s02   4   0   4   8   3   7   5   6   1   5   5   5  11   6   9  28  10
## s03   2   4   0   4   1   3   1   2   3   2   1   1   7   2   5  24   6
## s04   6   8   4   0   5   1   5   6   7   6   5   3   3   6   1  22   2
## s05   1   3   1   5   0   4   2   3   2   3   2   2   8   3   6  25   7
## s06   5   7   3   1   4   0   4   5   6   5   4   2   4   5   2  21   3
## s07   3   5   1   5   2   4   0   3   4   3   2   2   8   3   6  25   7
## s08   4   6   2   6   3   5   3   0   5   4   3   3   9   4   7  26   8
## s09   3   1   3   7   2   6   4   5   0   5   4   4  10   5   8  27   9
## s10   4   5   2   6   3   5   3   4   5   0   3   3   9   4   7  26   8
## s11   3   5   1   5   2   4   2   3   4   3   0   2   8   1   6  25   7
## s12   3   5   1   3   2   2   2   3   4   3   2   0   6   3   4  23   5
## s13   9  11   7   3   8   4   8   9  10   9   8   6   0   9   4  22   1
## s14   4   6   2   6   3   5   3   4   5   4   1   3   9   0   7  26   8
## s15   7   9   5   1   6   2   6   7   8   7   6   4   4   7   0  23   3
## s16  26  28  24  22  25  21  25  26  27  26  25  23  22  26  23   0  21
## s17   8  10   6   2   7   3   7   8   9   8   7   5   1   8   3  21   0
```

```
distances(net, weights=NA) # ignore weights
```

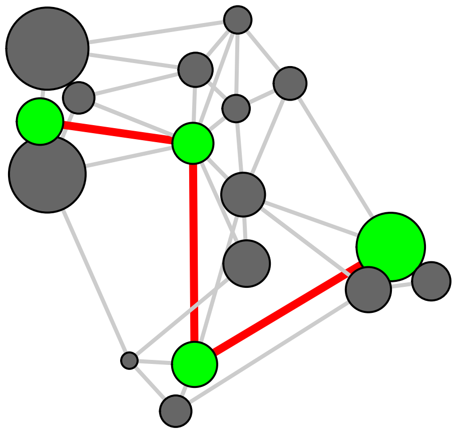
```
##     s01 s02 s03 s04 s05 s06 s07 s08 s09 s10 s11 s12 s13 s14 s15 s16 s17
## s01   0   1   1   1   1   2   2   2   2   2   2   2   3   3   1   3   2
## s02   1   0   1   2   1   3   2   2   1   1   2   2   3   3   2   4   3
## s03   1   1   0   1   1   2   1   1   2   1   1   1   2   2   2   3   2
## s04   1   2   1   0   2   1   2   2   3   2   1   1   2   2   1   2   1
## s05   1   1   1   2   0   2   2   2   1   2   2   2   3   3   1   3   3
## s06   2   3   2   1   2   0   3   3   3   3   2   1   2   2   1   1   1
## s07   2   2   1   2   2   3   0   1   2   1   2   2   2   1   3   4   3
## s08   2   2   1   2   2   3   1   0   1   2   2   2   3   2   3   4   3
## s09   2   1   2   3   1   3   2   1   0   1   3   3   4   3   2   4   4
## s10   2   1   1   2   2   3   1   2   1   0   2   2   3   2   3   4   3
## s11   2   2   1   1   2   2   2   2   3   2   0   2   2   1   2   3   2
## s12   2   2   1   1   2   1   2   2   3   2   2   0   1   1   2   2   2
## s13   3   3   2   2   3   2   2   3   4   3   2   1   0   1   3   2   1
## s14   3   3   2   2   3   2   1   2   3   2   1   1   1   0   3   3   2
## s15   1   2   2   1   1   1   3   3   2   3   2   2   3   3   0   2   2
## s16   3   4   3   2   3   1   4   4   4   4   3   2   2   3   2   0   1
## s17   2   3   2   1   3   1   3   3   4   3   2   2   1   2   2   1   0
```

```
dist.from.NYT <- distances(net, v=V(net)[media=="NY Times"], to=V(net), weights=NA)
# Set colors to plot the distances:
oranges <- colorRampPalette(c("dark red", "gold"))
col <- oranges(max(dist.from.NYT)+1)
col <- col[dist.from.NYT+1]
plot(net, vertex.color=col, vertex.label=dist.from.NYT, edge.arrow.size=.6,
vertex.label.color="white")
```



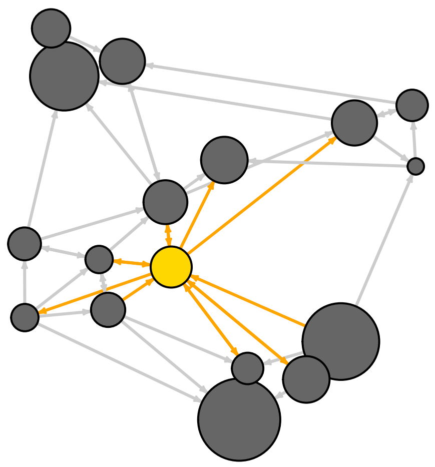
Here we calculated the distances from the "NY Times" node to all other nodes in the network using the `distances` function. The `v` argument is used to specify the starting vertex, and the `to` argument is used to specify the target vertices. The `weights` argument is set to `NA` to ignore edge weights. The `colorRampPalette` function is used to create a color palette for the distances, and the `plot` function is used to visualize the graph with colored vertices based on their distances from the "NY Times" node.

```r
news.path <- shortest_paths(net,
from = V(net)[media=="MSNBC"],
to = V(net)[media=="New York Post"],
output = "both") # both path nodes and edges
# Generate edge color variable to plot the path:
ecol <- rep("gray80", ecount(net))
ecol[unlist(news.path$epath)] <- "red"
# Generate edge width variable to plot the path:
ew <- rep(2, ecount(net))
ew[unlist(news.path$epath)] <- 4
# Generate node color variable to plot the path:
vcol <- rep("gray40", vcount(net))
vcol[unlist(news.path$vpath)] <- "green"
plot(net, vertex.color=vcol, edge.color=ecol,
edge.width=ew, edge.arrow.mode=0)
```



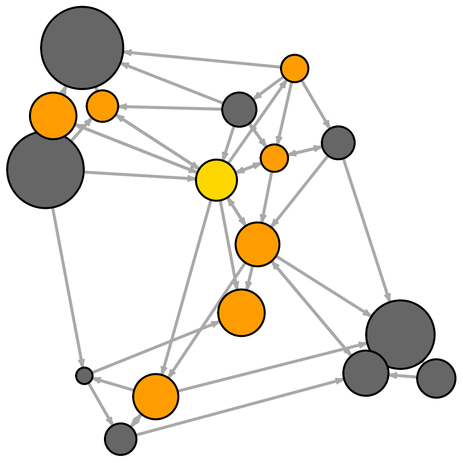
Here we calculated the shortest path from the “MSNBC” node to the “New York Post” node using the `shortest_paths` function. The `from` and `to` arguments are used to specify the starting and target vertices, and the `output` argument is set to “both” to return both the path nodes and edges. The `ecol`, `ew`, and `vcol` variables are used to set the colors and widths of the edges and vertices along the path. The `plot` function is used to visualize the graph with colored vertices and edges along the shortest path.

```
inc.edges <- incident(net,
V(net)[media=="Wall Street Journal"], mode="all")
# Set colors to plot the selected edges.
ecol <- rep("gray80", ecount(net))
ecol[inc.edges] <- "orange"
vcol <- rep("grey40", vcount(net))
vcol[V(net)$media=="Wall Street Journal"] <- "gold"
plot(net, vertex.color=vcol, edge.color=ecol)
```



Here we calculated the incident edges (going into or out) for the “Wall Street Journal” node using the `incident` function. The `mode` argument is set to “all” to include all incident edges. The `ecol` and `vcol` variables are used to set the colors of the edges and vertices. The `plot` function is used to visualize the graph with colored edges and vertices.

```
neigh.nodes <- neighbors(net, V(net)[media=="Wall Street Journal"], mode="out")
# Set colors to plot the neighbors:
vcol[neigh.nodes] <- "#ff9d00"
plot(net, vertex.color=vcol)
```



We identified the neighbor nodes of the "Wall Street Journal" node using the `neighbors` function. The `mode` argument is set to "out" to include only outgoing neighbors. The `vcol` variable is used to set the color of the neighbor nodes, and the `plot` function is used to visualize the graph with colored vertices.

```
E(net)[ V(net)[type.label=="Newspaper"] %->% V(net)[type.label=="Online"] ]
```

```
## + 7/48 edges from d0bd4fb (vertex names):
## [1] s01->s15 s03->s12 s04->s12 s04->s17 s05->s15 s06->s16 s06->s17
```

Here we select the edges from newspaper nodes to online nodes using the `E` function. The `V(net)[type.label=="Newspaper"] %->% V(net)[type.label=="Online"]` syntax is used to specify the source and target vertex types.

```
cocitation(net)
```

```
##     s01 s02 s03 s04 s05 s06 s07 s08 s09 s10 s11 s12 s13 s14 s15 s16 s17
## s01   0   1   1   2   1   1   0   1   2   2   1   1   0   0   1   0   0
## s02   1   0   1   1   0   0   0   0   1   0   0   0   0   0   2   0   0
## s03   1   1   0   1   0   1   1   1   2   2   1   1   0   1   1   0   1
## s04   2   1   1   0   1   1   0   1   0   1   1   1   0   0   1   0   0
## s05   1   0   0   1   0   0   0   1   0   1   1   1   0   0   0   0   0
## s06   1   0   1   1   0   0   0   0   0   0   1   1   1   1   0   0   2
## s07   0   0   1   0   0   0   0   0   1   0   0   0   0   0   0   0   0
## s08   1   0   1   1   1   0   0   0   0   2   1   1   0   1   0   0   0
## s09   2   1   2   0   0   0   1   0   0   1   0   0   0   0   1   0   0
## s10   2   0   2   1   1   0   0   2   1   0   1   1   0   1   0   0   0
## s11   1   0   1   1   1   1   0   1   0   1   0   2   1   0   0   0   1
## s12   1   0   1   1   1   1   0   1   0   1   2   0   0   0   0   0   2
## s13   0   0   0   0   0   1   0   0   0   0   1   0   0   1   0   0   0
## s14   0   0   1   0   0   1   0   1   0   1   0   0   1   0   0   0   0
## s15   1   2   1   1   0   0   0   0   1   0   0   0   0   0   0   0   0
## s16   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
## s17   0   0   1   0   0   2   0   0   0   0   1   2   0   0   0   1   0
```

Here we calculated the cocitation matrix for the network using the `cocitation` function. The cocitation matrix shows the number of times two nodes are cited together in the network.

8. Subgroups and communities

These are subsets of nodes that are more densely connected to each other than to the rest of the network. They provide the insights about the structure and function of the network.

```
net.sym <- as.undirected(net, mode= "collapse",
edge.attr.comb=list(weight="sum", "ignore"))
```

```
## Warning: ‘as.undirected()‘ was deprecated in igraph 2.1.0.
## i Please use ‘as_undirected()‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```
net.sym
```

```
## IGRAPH 8767fa2 UNW- 17 38 --
## + attr: name (v/c), media (v/c), media.type (v/n), type.label (v/c),
## | audience.size (v/n), color (v/c), size (v/n), label.color (v/c),
## | label (v/l), weight (e/n)
## + edges from 8767fa2 (vertex names):
##  [1] s01--s02 s01--s03 s02--s03 s01--s04 s03--s04 s01--s05 s02--s05 s03--s05
##  [9] s04--s06 s03--s07 s03--s08 s07--s08 s02--s09 s05--s09 s08--s09 s02--s10
## [17] s03--s10 s07--s10 s09--s10 s03--s11 s04--s11 s03--s12 s04--s12 s06--s12
## [25] s12--s13 s07--s14 s11--s14 s12--s14 s13--s14 s01--s15 s04--s15 s05--s15
## [33] s06--s15 s06--s16 s04--s17 s06--s17 s13--s17 s16--s17
```

8.1 Cliques

Cliques are subsets of nodes that are all connected to each other. They are the most tightly connected subgroups in a network.

```
cliques(net.sym) # list of cliques
```

```
## [[1]]
## + 1/17 vertex, named, from 8767fa2:
## [1] s03
##
## [[2]]
## + 1/17 vertex, named, from 8767fa2:
## [1] s06
##
## [[3]]
## + 1/17 vertex, named, from 8767fa2:
## [1] s14
##
## [[4]]
## + 1/17 vertex, named, from 8767fa2:
## [1] s09
##
## [[5]]
## + 1/17 vertex, named, from 8767fa2:
## [1] s04
##
## [[6]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s04 s06
##
## [[7]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s03 s04
##
## [[8]]
## + 1/17 vertex, named, from 8767fa2:
## [1] s05
##
## [[9]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s05 s09
##
## [[10]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s03 s05
##
## [[11]]
## + 1/17 vertex, named, from 8767fa2:
## [1] s13
##
## [[12]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s13 s14
##
## [[13]]
## + 1/17 vertex, named, from 8767fa2:
## [1] s10
##
## [[14]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s09 s10
##
## [[15]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s03 s10
```

```
##
## [[16]]
## + 1/17 vertex, named, from 8767fa2:
## [1] s16
##
## [[17]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s06 s16
##
## [[18]]
## + 1/17 vertex, named, from 8767fa2:
## [1] s08
##
## [[19]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s08 s09
##
## [[20]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s03 s08
##
## [[21]]
## + 1/17 vertex, named, from 8767fa2:
## [1] s01
##
## [[22]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s01 s05
##
## [[23]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s01 s03 s05
##
## [[24]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s01 s04
##
## [[25]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s01 s03 s04
##
## [[26]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s01 s03
##
## [[27]]
## + 1/17 vertex, named, from 8767fa2:
## [1] s17
##
## [[28]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s16 s17
##
## [[29]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s06 s16 s17
##
## [[30]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s13 s17
##
## [[31]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s04 s17
##
## [[32]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s04 s06 s17
##
## [[33]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s06 s17
##
## [[34]]
## + 1/17 vertex, named, from 8767fa2:
## [1] s12
##
## [[35]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s12 s13
##
## [[36]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s12 s13 s14
##
## [[37]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s04 s12
##
## [[38]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s04 s06 s12
##
## [[39]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s03 s04 s12
##
## [[40]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s12 s14
##
## [[41]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s06 s12
##
## [[42]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s03 s12
```

```
##
## [[43]]
## + 1/17 vertex, named, from 8767fa2:
## [1] s11
##
## [[44]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s04 s11
##
## [[45]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s03 s04 s11
##
## [[46]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s11 s14
##
## [[47]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s03 s11
##
## [[48]]
## + 1/17 vertex, named, from 8767fa2:
## [1] s07
##
## [[49]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s07 s08
##
## [[50]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s03 s07 s08
##
## [[51]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s07 s10
##
## [[52]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s03 s07 s10
##
## [[53]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s07 s14
##
## [[54]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s03 s07
##
## [[55]]
## + 1/17 vertex, named, from 8767fa2:
## [1] s15
##
## [[56]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s01 s15
##
## [[57]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s01 s05 s15
##
## [[58]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s01 s04 s15
##
## [[59]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s05 s15
##
## [[60]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s04 s15
##
## [[61]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s04 s06 s15
##
## [[62]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s06 s15
##
## [[63]]
## + 1/17 vertex, named, from 8767fa2:
## [1] s02
##
## [[64]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s01 s02
##
## [[65]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s01 s02 s05
##
## [[66]]
## + 4/17 vertices, named, from 8767fa2:
## [1] s01 s02 s03 s05
##
## [[67]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s01 s02 s03
##
## [[68]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s02 s10
##
## [[69]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s02 s09 s10
```

```
##
## [[70]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s02 s03 s10
##
## [[71]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s02 s05
##
## [[72]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s02 s05 s09
##
## [[73]]
## + 3/17 vertices, named, from 8767fa2:
## [1] s02 s03 s05
##
## [[74]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s02 s09
##
## [[75]]
## + 2/17 vertices, named, from 8767fa2:
## [1] s02 s03
```

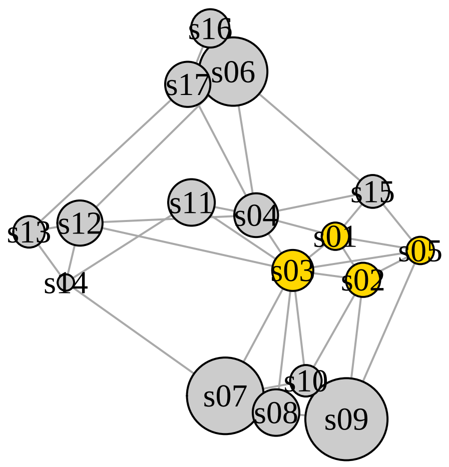
```
sapply(cliques(net.sym), length) # clique sizes
```

```
##  [1] 1 1 1 1 1 2 2 1 2 2 1 2 1 2 2 1 2 1 2 2 1 2 3 2 3 2 1 2 3 2 2 3 2 1 2 3 2 3
## [39] 3 2 2 2 1 2 3 2 2 1 2 3 2 3 2 2 1 2 3 3 2 2 3 2 1 2 3 4 3 2 3 3 2 3 3 2 2
```

```
largest_cliques(net.sym) # cliques with max number of nodes
```

```
## [[1]]
## + 4/17 vertices, named, from 8767fa2:
## [1] s02 s03 s05 s01
```

```
vcol <- rep("grey80", vcount(net.sym))
vcol[unlist(largest_cliques(net.sym))] <- "gold"
plot(as.undirected(net.sym), vertex.label=V(net.sym)$name, vertex.color=vcol)
```



Here we calculated the cliques in the network using the `cliques` function. The `sapply` function is used to get the sizes of the cliques, and the `largest_cliques` function is used to get the cliques with the maximum number of nodes. The `vcol` variable is used to set the color of the vertices in the largest cliques, and the `plot` function is used to visualize the graph with colored vertices.

8.2 Community detection

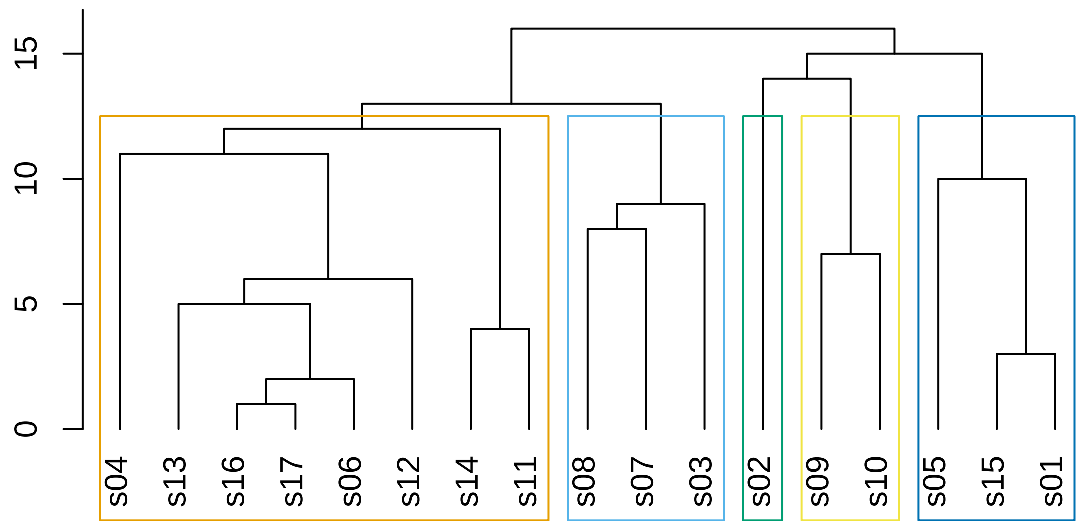
Community detection based on edge betweenness (Newman-Girvan) High-betweenness edges are removed sequentially (recalculating at each step) and the best parti-tioning of the network is selected.

```
ceb <- cluster_edge_betweenness(net)
```

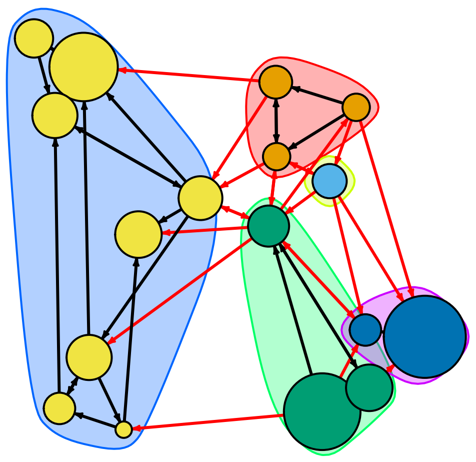
```
## Warning in cluster_edge_betweenness(net): At
## vendor/cigraph/src/community/edge_betweenness.c:503 : Membership vector will be
## selected based on the highest modularity score.
```

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

```
## Warning: ‘dendPlot()‘ was deprecated in igraph 2.0.0.
## i Please use ‘plot_dendrogram()‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```



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



```r
class(ceb)
```

```
## [1] "communities"
```

```r
len <- length(ceb)
print(len)
```

```
## [1] 5
```

```r
membership <- membership(ceb)
print(membership)
```

```
## s01 s02 s03 s04 s05 s06 s07 s08 s09 s10 s11 s12 s13 s14 s15 s16 s17
##   1   2   3   4   1   4   3   3   5   5   4   4   4   4   1   4   4
```

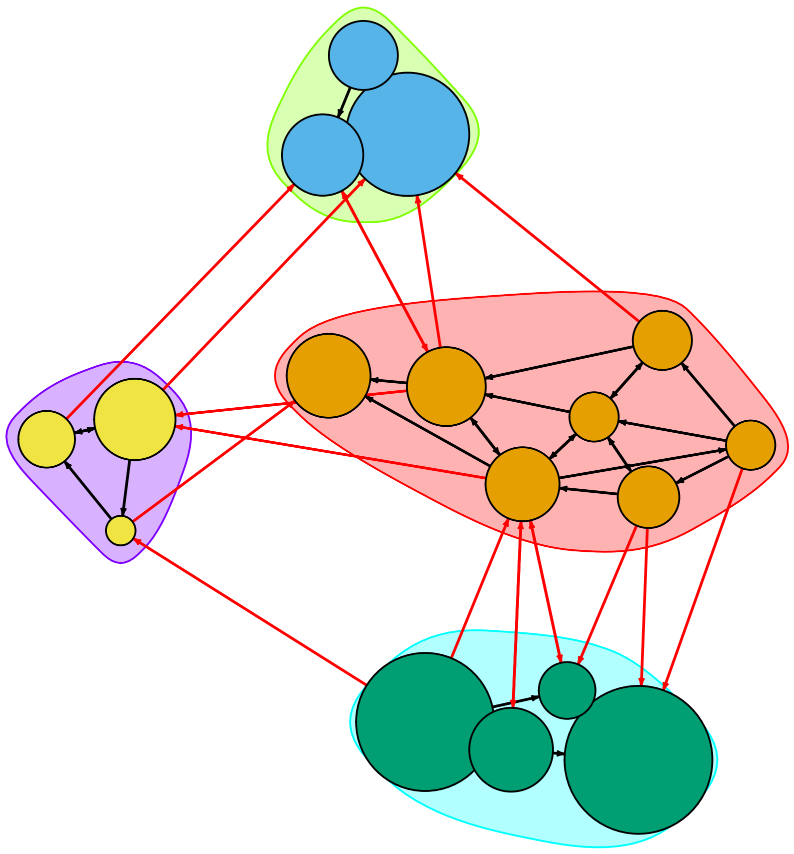
```r
modularity <- modularity(ceb)
print(modularity)
```

```
## [1] 0.2970913
```

High modularity indicates that the network is well partitioned into communities. The `membership` function is used to get the membership of each node in the communities, and the `modularity` function is used to calculate the modularity of the partitioning.

```r
clp <- cluster_label_prop(net)
plot(clp, net)
```

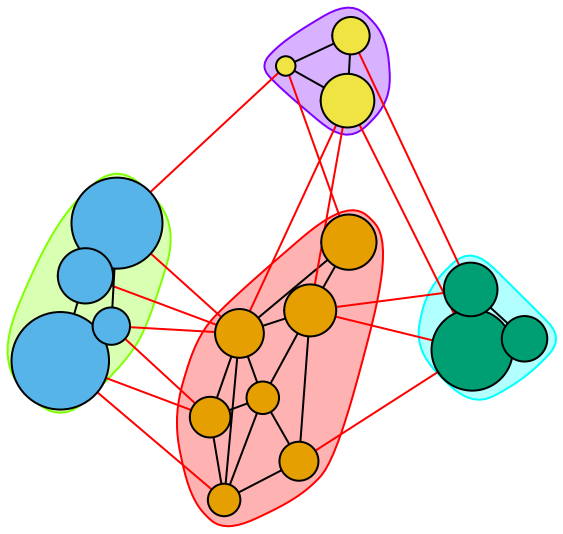
Community detection based on based on propagating labels



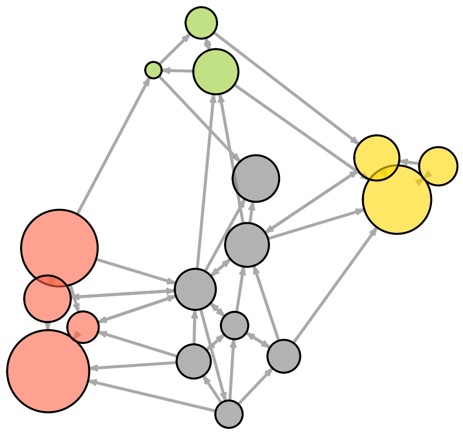
Here we used the `cluster_label_prop` function to detect communities based on propagating labels. The `plot` function is used to visualize the graph with the detected communities.

```r
cfg <- cluster_fast_greedy(as.undirected(net))
plot(cfg, as.undirected(net))
```

Community detection based on greedy optimization of modularity

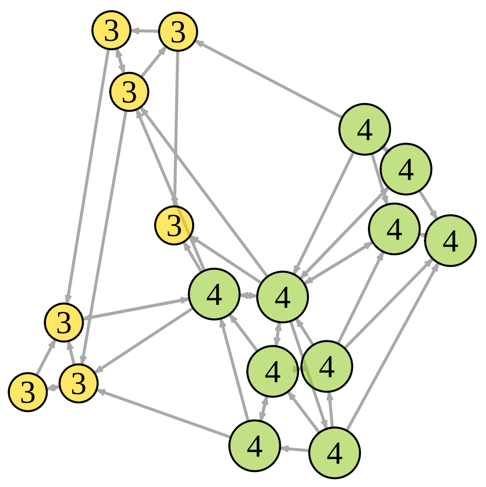


```r
V(net)$community <- cfg$membership
colrs <- adjustcolor( c("gray50", "tomato", "gold", "yellowgreen"), alpha=.6)
plot(net, vertex.color=colrs[V(net)$community])
```



8.3 K-core decomposition K-core decomposition is a method for identifying the core structure of a network by removing nodes with low degree iteratively until all remaining nodes have a degree greater than or equal to k.

```r
kc <- coreness(net, mode="all")
plot(net, vertex.size=kc*6, vertex.label=kc, vertex.color=colrs[kc])
```



We calculated the k-core decomposition of the network using the `coreness` function. The `mode` argument is set to "all" to include all nodes. The `plot` function is used to visualize the graph with vertex sizes and labels based on their k-core values.

9. Assortativity and Homophily

Homophily: the tendency of nodes to connect to others who are similar on some variable.

```r
assortativity_nominal(net, V(net)$media.type, directed=F)
```

```
## [1] 0.1715568
```

The value 0.17 indicates that there is a moderate level of assortativity in the network for media.type. The `assortativity_nominal` function is used to calculate the assortativity based on the media type of the nodes. The `directed` argument is set to `FALSE` to treat the network as undirected.

```r
assortativity(net, V(net)$audience.size, directed=F)
```

```
## [1] -0.1102857
```

-0.11028 indicates that there is a weak level of assortativity in the network for audience size. The `assortativity` function is used to calculate the assortativity based on the audience size of the nodes.

```r
assortativity_degree(net, directed=F)
```

```
## [1] -0.009551146
```

assortativity for the network -0.0095 indicates that there is a weak level of assortativity in the network for degree. The `assortativity_degree` function is used to calculate the assortativity based on the degree of the nodes.