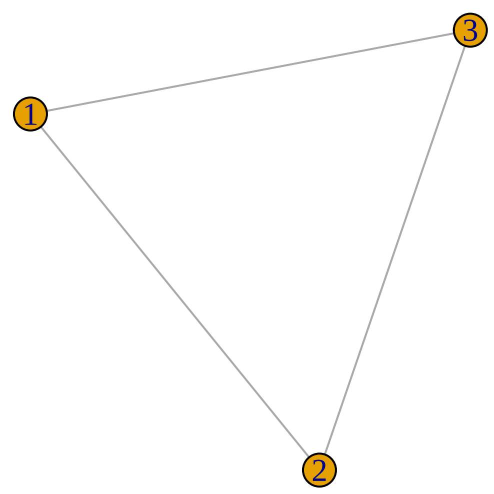
Network Analysis

Tanay Mukherjee

7/1/2020

```r
#-------Create networks--------

g1 <- graph( edges=c(1,2, 2,3, 3,1), n=3, directed=F ) # an undirected graph with 3 edges
# The numbers are interpreted as vertex IDs, so the edges are 1-->2, 2-->3, 3-->1
plot(g1) # A simple plot of the network - we'll talk more about plots later
```



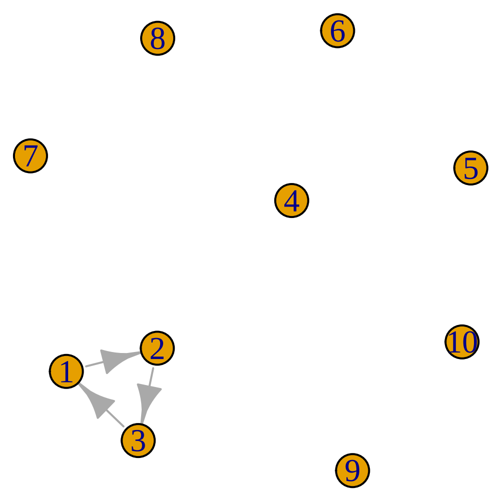

```r
class(g1)
```

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

```r
g1
```

```
## IGRAPH 1a6f225 U--- 3 3 --
## + edges from 1a6f225:
## [1] 1--2 2--3 1--3
```

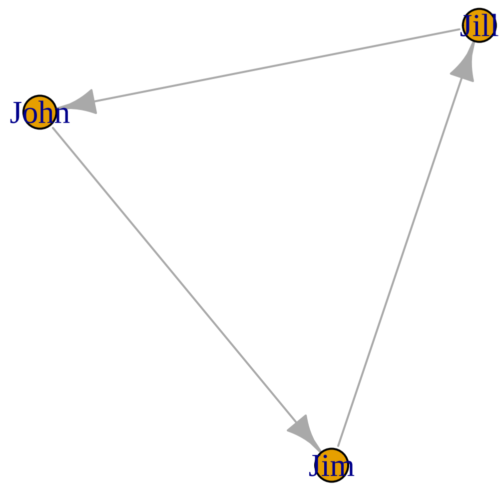
```r
g2 <- graph( edges=c(1,2, 2,3, 3,1), n=10 ) # now with 10 vertices, and directed by default
plot(g2)
```




```r
g2
```

```
## IGRAPH 1a77811 D--- 10 3 --
## + edges from 1a77811:
## [1] 1->2 2->3 3->1
```

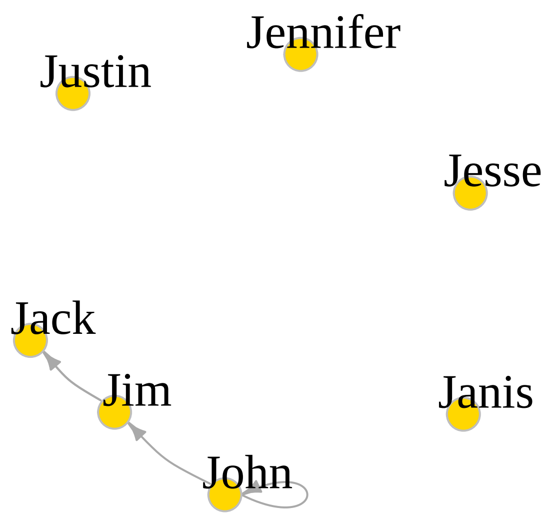
```r
g3 <- graph( c("John", "Jim", "Jim", "Jill", "Jill", "John")) # named vertices
# When the edge list has vertex names, the number of nodes is not needed
plot(g3)
```



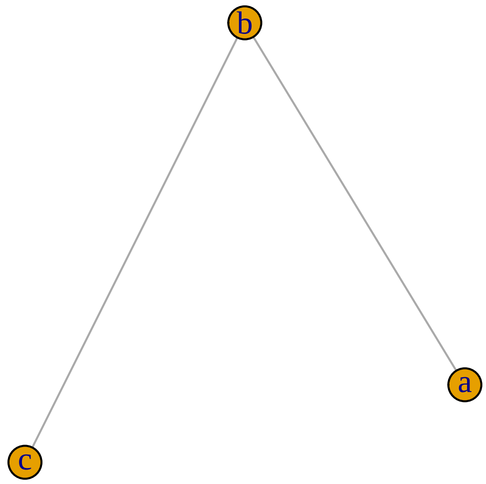

```
g3
```

```
## IGRAPH 1a792e2 DN-- 3 3 --
## + attr: name (v/c)
## + edges from 1a792e2 (vertex names):
## [1] John->Jim  Jim ->Jill Jill->John
```

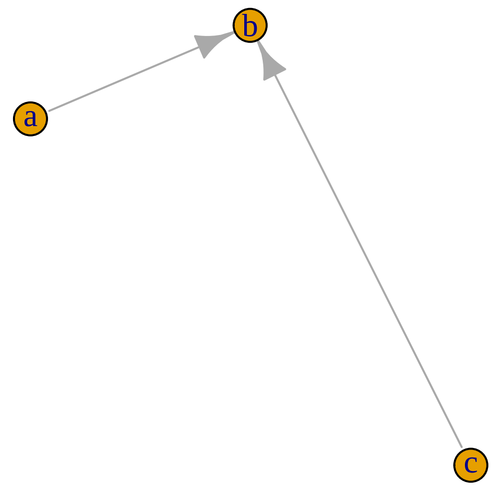
```
g4 <- graph( c("John", "Jim", "Jim", "Jack", "Jim", "Jack", "John", "John"),
             isolates=c("Jesse", "Janis", "Jennifer", "Justin") )
# In named graphs we can specify isolates by providing a list of their names.

plot(g4, edge.arrow.size=.5, vertex.color="gold", vertex.size=15,
     vertex.frame.color="gray", vertex.label.color="black",
     vertex.label.cex=1.5, vertex.label.dist=2, edge.curved=0.2)
```



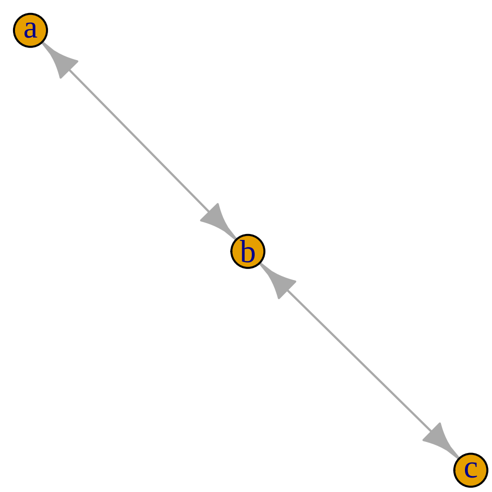

```r
# Small graphs can also be generated with a description of this kind:
# '-' for undirected tie, "+-' or "-+" for directed ties pointing left & right,
# "++" for a symmetric tie, and ":" for sets of vertices

plot(graph_from_literal(a---b, b---c)) # the number of dashes doesn't matter
```



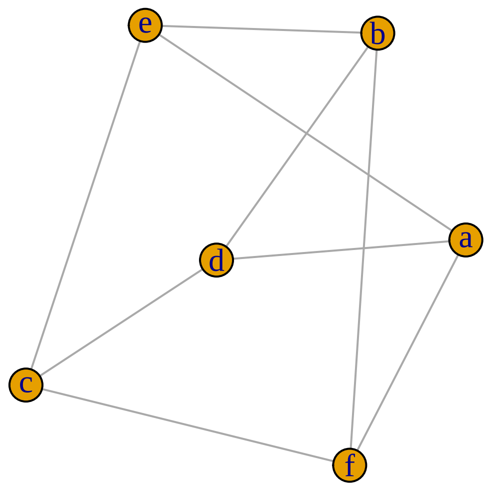

```
plot(graph_from_literal(a--+b, b+--c))
```



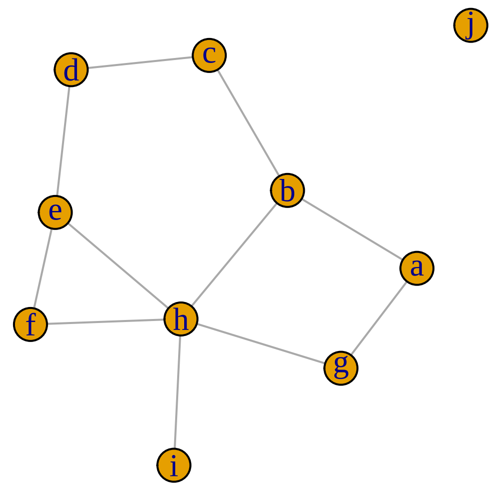

```
plot(graph_from_literal(a+-+b, b+-+c))
```




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




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




```
#  ------->> Edge, vertex, and network attributes --------

# Access vertices and edges:
E(g4) # The edges of the object
```

```
## + 4/4 edges from 1a7b012 (vertex names):
## [1] John->Jim  Jim ->Jack Jim ->Jack John->John
```

```
V(g4) # The vertices of the object
```

```
## + 7/7 vertices, named, from 1a7b012:
## [1] John     Jim      Jack     Jesse    Janis    Jennifer Justin
```

```
# You can examine the network matrix directly:
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
g4[1,]
```

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

```r
# Add attributes to the network, vertices, or edges:
V(g4)$name # automatically generated when we created the network.
```

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

```r
V(g4)$gender <- c("male", "male", "male", "male", "female", "female", "male")
E(g4)$type <- "email" # Edge attribute, assign "email" to all edges
E(g4)$weight <- 10    # Edge weight, setting all existing edges to 10

# Examine attributes
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()
```

```r
# Another way to set attributes
# (you can similarly use set_edge_attr(), set_vertex_attr(), etc.)
g4 <- set_graph_attr(g4, "name", "Email Network")
g4 <- set_graph_attr(g4, "something", "A thing")

graph_attr_names(g4)
```

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

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

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

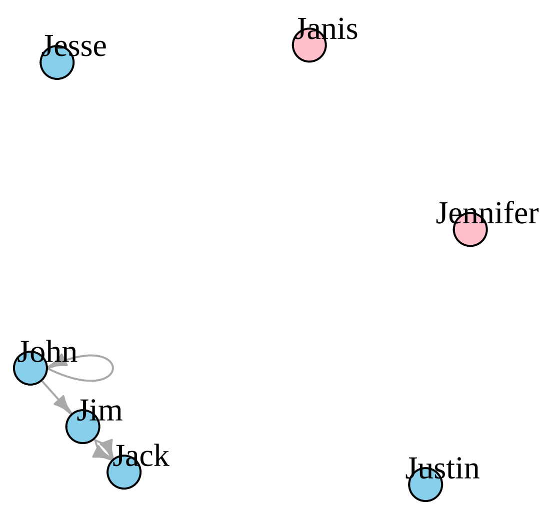
```r
graph_attr(g4)
```

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

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

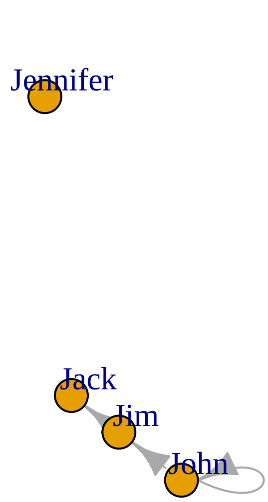
```
## $name
## [1] "Email Network"
```

```r
plot(g4, edge.arrow.size=.5, vertex.label.color="black", vertex.label.dist=1.5,
     vertex.color=c( "pink", "skyblue")[1+(V(g4)$gender=="male")] )
```




```r
# g4 has two edges going from Jim to Jack, and a loop from John to himself.
# We can simplify our graph to remove loops & multiple edges between the same nodes.
# Use 'edge.attr.comb' to indicate how edge attributes are to be combined - possible
# options include "sum", "mean", "prod" (product), min, max, first/last (selects
# the first/last edge's attribute). Option "ignore" says the attribute should be
# disregarded and dropped.
```

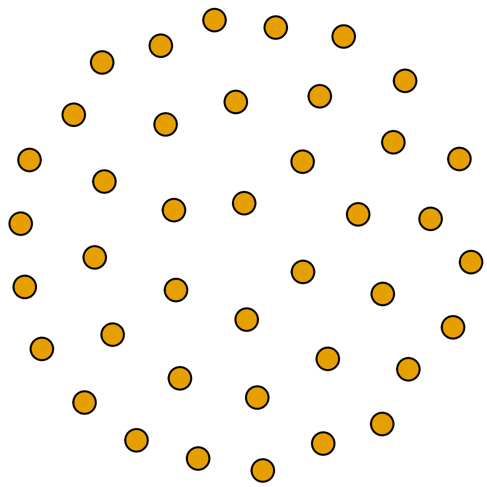
```r
g4s <- simplify( g4, remove.multiple = T, remove.loops = F,
                 edge.attr.comb=list(weight="sum", type="ignore") )
plot(g4s, vertex.label.dist=1.5)
```



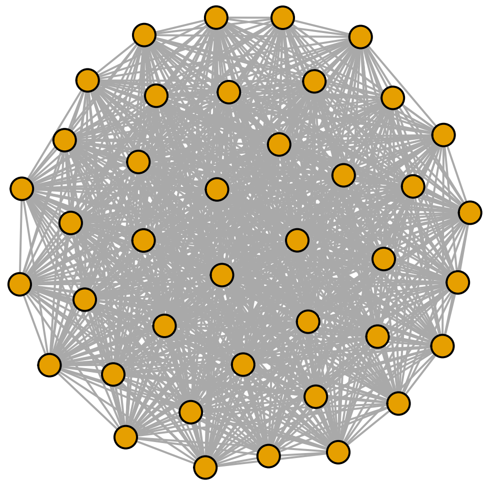

```r
g4s
```

```
## IGRAPH 1b38035 DNW- 7 3 -- Email Network
## + attr: name (g/c), name (v/c), gender (v/c), weight (e/n)
## + edges from 1b38035 (vertex names):
## [1] John->John John->Jim  Jim ->Jack
```

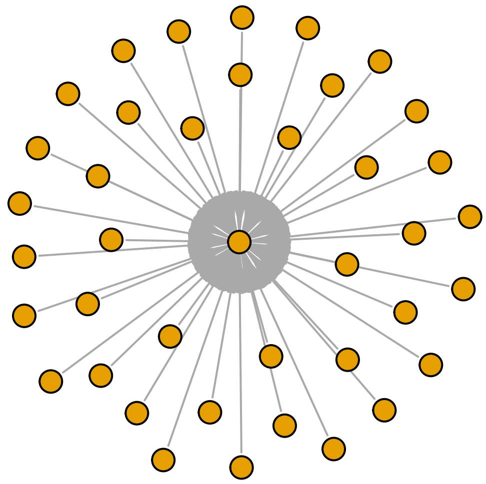
```r
#-------Specific graphs and graph models--------

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



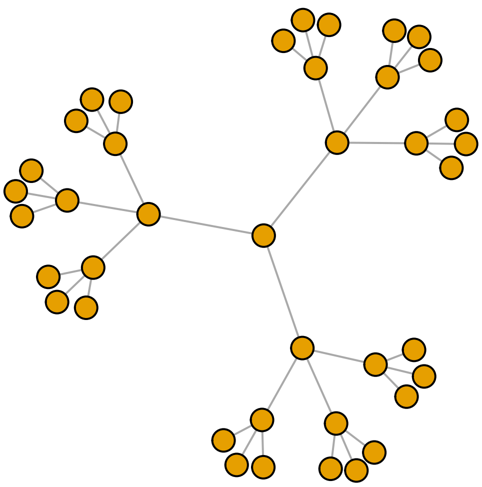
```r
# Full graph
fg <- make_full_graph(40)
plot(fg, vertex.size=10, vertex.label=NA)
```



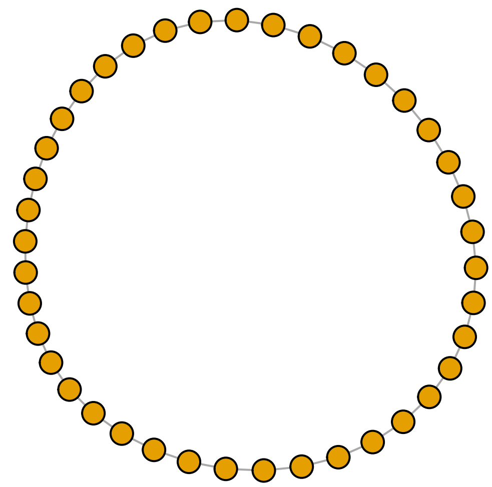
```r
# Star graph
st <- make_star(40)
plot(st, vertex.size=10, vertex.label=NA)
```



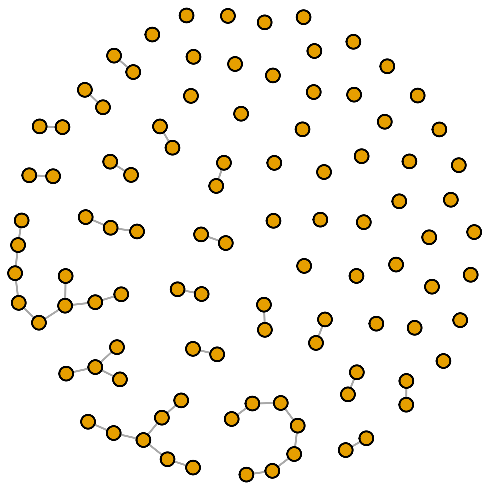
```r
# Tree graph
tr <- make_tree(40, children = 3, mode = "undirected")
plot(tr, vertex.size=10, vertex.label=NA)
```



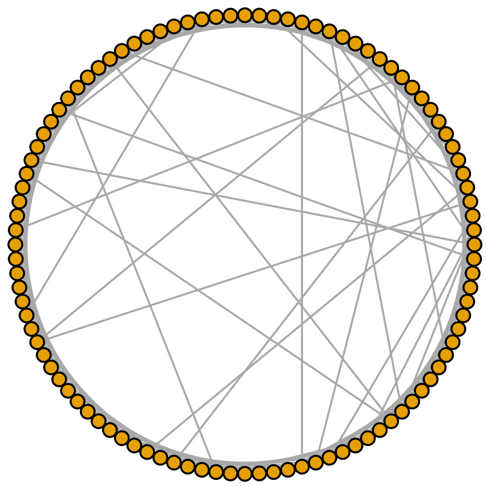
```
# Ring graph
rn <- make_ring(40)
plot(rn, vertex.size=10, vertex.label=NA)
```



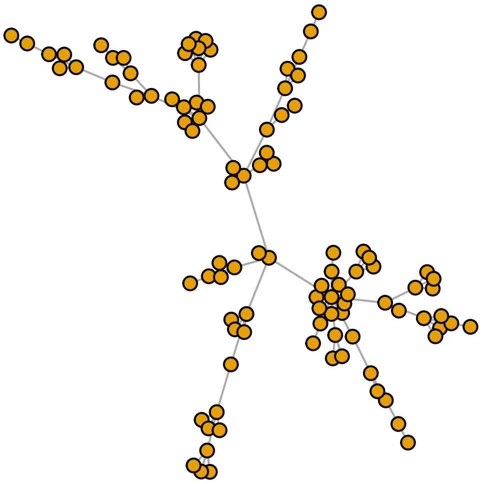
```r
# Erdos-Renyi random graph
# ('n' is number of nodes, 'm' is the number of edges)
er <- sample_gnm(n=100, m=40)
plot(er, vertex.size=6, vertex.label=NA)
```



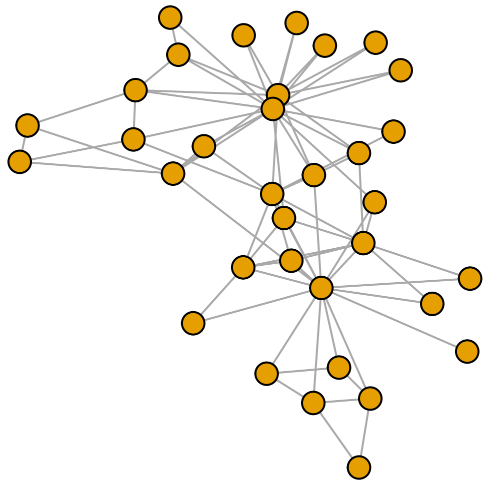
```r
# Watts-Strogatz small-world graph
# Creates a lattice with 'dim' dimensions of 'size' nodes each, and rewires edges
# randomly with probability 'p'. You can allow 'loops' and 'multiple' edges.
# The neighborhood in which edges are connected is 'nei'.
sw <- sample_smallworld(dim=2, size=10, nei=1, p=0.1)
plot(sw, vertex.size=6, vertex.label=NA, layout=layout_in_circle)
```



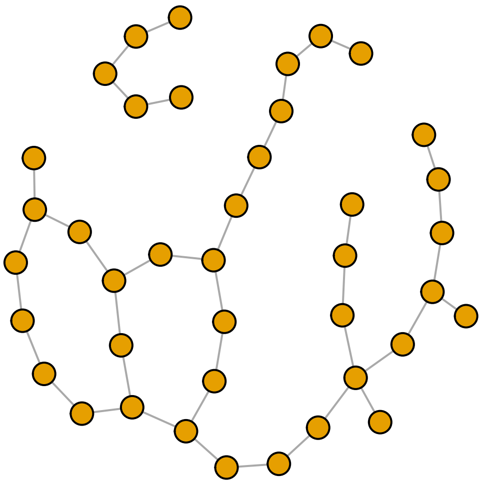
```r
# Barabasi-Albert preferential attachment model for scale-free graphs
# 'n' is number of nodes, 'power' is the power of attachment (1 is linear)
# 'm' is the number of edges added on each time step
ba <-  sample_pa(n=100, power=1, m=1,  directed=F)
plot(ba, vertex.size=6, vertex.label=NA)
```



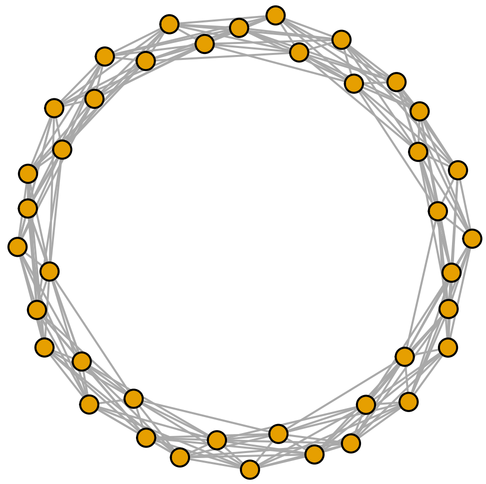
```r
#igraph can also give you some notable historical graphs. For instance:
zach <- graph("Zachary") # the Zachary carate club
plot(zach, vertex.size=10, vertex.label=NA)
```



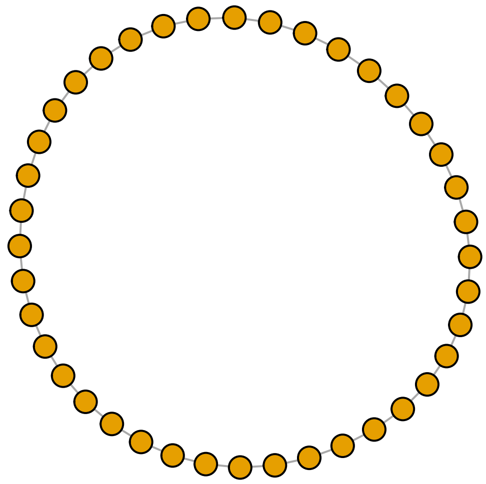
```r
# Rewiring a graph
# 'each_edge()' is a rewiring method that changes the edge endpoints
# uniformly randomly with a probability 'prob'.
rn.rewired <- rewire(rn, each_edge(prob=0.1))
plot(rn.rewired, vertex.size=10, vertex.label=NA)
```



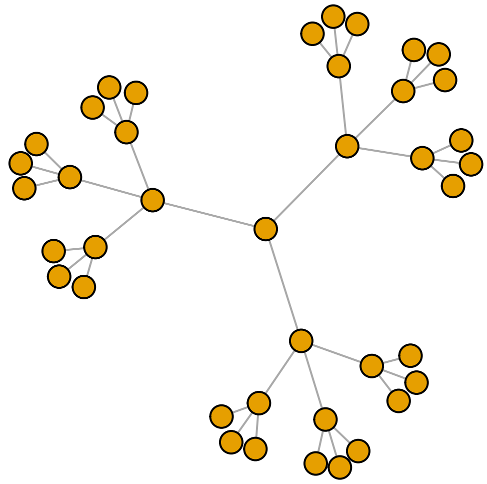
```
# Rewire to connect vertices to other vertices at a certain distance.
rn.neigh = connect.neighborhood(rn, 5)
plot(rn.neigh, vertex.size=8, vertex.label=NA)
```



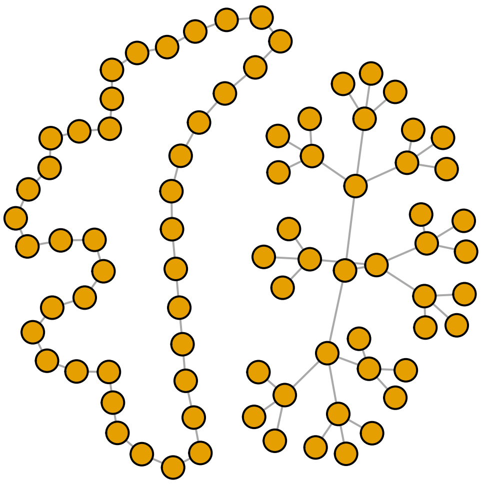
```
# Combine graphs (disjoint union, assuming separate vertex sets): %du%
plot(rn, vertex.size=10, vertex.label=NA)
```



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



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



```
# ================ 3. Reading network data from files ================

rm(list = ls()) # clear the workspace again

# Download the archive with the data files from http://bitly.com/netscix2016

# Set the working directory to the folder containing the workshop files:
setwd("~/CUNY/Fall 2019/9750 - Software Tools and Techniques_Data Science")

# DATASET 1: edgelist
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
```

```r
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
```

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

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

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

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

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

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

```r
# Collapse multiple 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


# DATASET 2: matrix
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)

# Examine the data:
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
```

```
# links2 is an adjacency matrix for a two-mode network:
links2 <- as.matrix(links2)
dim(links2)
```

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

```
dim(nodes2)
```

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

```
# ================ 4. Turning networks into igraph objects ================

library(igraph)

#  ------->> DATASET 1 --------

# Converting the data to an igraph object:
# The graph.data.frame function, which takes two data frames: 'd' and 'vertices'.
# 'd' describes the edges of the network - it should start with two columns
# containing the source and target node IDs for each network tie.
# 'vertices' should start with a column of node IDs.
# Any additional columns in either data frame are interpreted as attributes.

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

# Examine the resulting object:
class(net)
```

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

```
net
```

```
## IGRAPH 1ba7b74 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 1ba7b74 (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
```

```r
# We can look at the nodes, edges, and their attributes:
E(net)
```

```
## + 49/49 edges from 1ba7b74 (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
```

```r
V(net)
```

```
## + 17/17 vertices, named, from 1ba7b74:
##  [1] s01 s02 s03 s04 s05 s06 s07 s08 s09 s10 s11 s12 s13 s14 s15 s16 s17
```

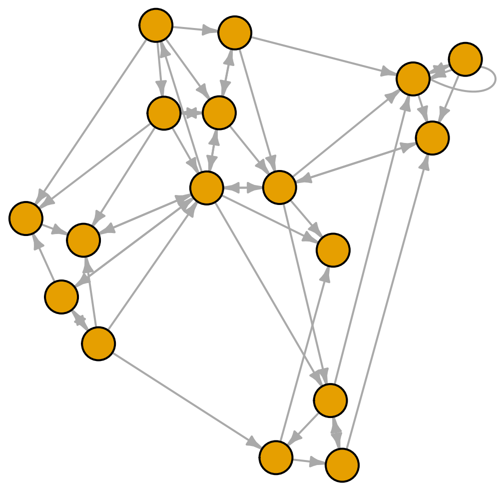
```r
E(net)$type
```

```
##  [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"
```

```r
V(net)$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)
```



```
# Removing loops from the graph:
net <- simplify(net, remove.multiple = F, remove.loops = T)

# If you need them, you can extract an edge list or a matrix from igraph networks.
as_edgelist(net, names=T)
```

```
##       [,1]  [,2]
##  [1,] "s01" "s02"
##  [2,] "s01" "s03"
##  [3,] "s01" "s04"
##  [4,] "s01" "s15"
##  [5,] "s02" "s01"
##  [6,] "s02" "s03"
##  [7,] "s02" "s09"
##  [8,] "s02" "s10"
##  [9,] "s03" "s01"
## [10,] "s03" "s04"
## [11,] "s03" "s05"
## [12,] "s03" "s08"
## [13,] "s03" "s10"
## [14,] "s03" "s11"
## [15,] "s03" "s12"
## [16,] "s04" "s03"
## [17,] "s04" "s06"
## [18,] "s04" "s11"
## [19,] "s04" "s12"
```

```
## [20,] "s04" "s17"
## [21,] "s05" "s01"
## [22,] "s05" "s02"
## [23,] "s05" "s09"
## [24,] "s05" "s15"
## [25,] "s06" "s16"
## [26,] "s06" "s17"
## [27,] "s07" "s03"
## [28,] "s07" "s08"
## [29,] "s07" "s10"
## [30,] "s07" "s14"
## [31,] "s08" "s03"
## [32,] "s08" "s07"
## [33,] "s08" "s09"
## [34,] "s09" "s10"
## [35,] "s10" "s03"
## [36,] "s12" "s06"
## [37,] "s12" "s13"
## [38,] "s12" "s14"
## [39,] "s13" "s12"
## [40,] "s13" "s17"
## [41,] "s14" "s11"
## [42,] "s14" "s13"
## [43,] "s15" "s01"
## [44,] "s15" "s04"
## [45,] "s15" "s06"
## [46,] "s16" "s06"
## [47,] "s16" "s17"
## [48,] "s17" "s04"
```

```
as_adjacency_matrix(net, attr="weight")
```

```
## 17 x 17 sparse Matrix of class "dgCMatrix"

##    [[ suppressing 17 column names 's01', 's02', 's03' ... ]]

##
## s01  . 22 22 21 .  .  .  .  .  .  .  .  .  .  . 20  .  .
## s02 23  . 21  . .  .  .  .  1  5  .  .  .  .  .  .  .
## s03 21  .  . 22 1  .  .  4  .  2  1  1  .  .  .  .  .
## s04  .  . 23  . .  1  .  .  .  . 22  3  .  .  .  .  2
## s05  1 21  .  . .  .  .  .  2  .  .  .  .  . 21  .  .
## s06  .  .  .  . .  .  .  .  .  .  .  .  .  .  . 21 21
## s07  .  .  1  . .  .  . 22  . 21  .  .  .  4  .  .  .
## s08  .  .  2  . .  . 21  . 23  .  .  .  .  .  .  .  .
## s09  .  .  .  . .  .  .  .  . 21  .  .  .  .  .  .  .
## s10  .  .  2  . .  .  .  .  .  .  .  .  .  .  .  .  .
## s11  .  .  .  . .  .  .  .  .  .  .  .  .  .  .  .  .
## s12  .  .  .  . .  2  .  .  .  .  .  . 22 22  .  .  .
## s13  .  .  .  . .  .  .  .  .  .  . 21  .  .  .  .  1
## s14  .  .  .  . .  .  .  .  .  .  1  . 21  .  .  .  .
## s15 22  .  .  1 .  4  .  .  .  .  .  .  .  .  .  .  .
## s16  .  .  .  . . 23  .  .  .  .  .  .  .  .  .  . 21
## s17  .  .  .  4 .  .  .  .  .  .  .  .  .  .  .  .  .
```

```r
# Or data frames describing nodes and edges:
as_data_frame(net, what="edges")
```

```
##    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
## 7   s02 s09 hyperlink      1
## 8   s02 s10 hyperlink      5
## 9   s03 s01 hyperlink     21
## 10  s03 s04 hyperlink     22
## 11  s03 s05 hyperlink      1
## 12  s03 s08 hyperlink      4
## 13  s03 s10   mention      2
## 14  s03 s11 hyperlink      1
## 15  s03 s12 hyperlink      1
## 16  s04 s03 hyperlink     23
## 17  s04 s06   mention      1
## 18  s04 s11   mention     22
## 19  s04 s12 hyperlink      3
## 20  s04 s17   mention      2
## 21  s05 s01   mention      1
## 22  s05 s02 hyperlink     21
## 23  s05 s09 hyperlink      2
## 24  s05 s15   mention     21
## 25  s06 s16 hyperlink     21
## 26  s06 s17   mention     21
## 27  s07 s03   mention      1
## 28  s07 s08   mention     22
## 29  s07 s10 hyperlink     21
## 30  s07 s14   mention      4
## 31  s08 s03 hyperlink      2
## 32  s08 s07   mention     21
## 33  s08 s09   mention     23
## 34  s09 s10   mention     21
## 35  s10 s03 hyperlink      2
## 36  s12 s06   mention      2
## 37  s12 s13 hyperlink     22
## 38  s12 s14   mention     22
## 39  s13 s12 hyperlink     21
## 40  s13 s17   mention      1
## 41  s14 s11   mention      1
## 42  s14 s13   mention     21
## 43  s15 s01 hyperlink     22
## 44  s15 s04 hyperlink      1
## 45  s15 s06 hyperlink      4
## 46  s16 s06 hyperlink     23
## 47  s16 s17   mention     21
## 48  s17 s04 hyperlink      4
```

```r
as_data_frame(net, what="vertices")
```

```
##     name               media media.type type.label audience.size
## s01  s01            NY Times          1  Newspaper            20
## s02  s02     Washington Post          1  Newspaper            25
## s03  s03 Wall Street Journal          1  Newspaper            30
## s04  s04           USA Today          1  Newspaper            32
## s05  s05            LA Times          1  Newspaper            20
## s06  s06       New York Post          1  Newspaper            50
## s07  s07                 CNN          2         TV            56
## s08  s08               MSNBC          2         TV            34
## s09  s09            FOX News          2         TV            60
## s10  s10                 ABC          2         TV            23
## s11  s11                 BBC          2         TV            34
## s12  s12          Yahoo News          3     Online            33
## s13  s13         Google News          3     Online            23
## s14  s14         Reuters.com          3     Online            12
## s15  s15         NYTimes.com          3     Online            24
## s16  s16  WashingtonPost.com          3     Online            28
## s17  s17             AOL.com          3     Online            33
```

```r
#  ------->> DATASET 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
```

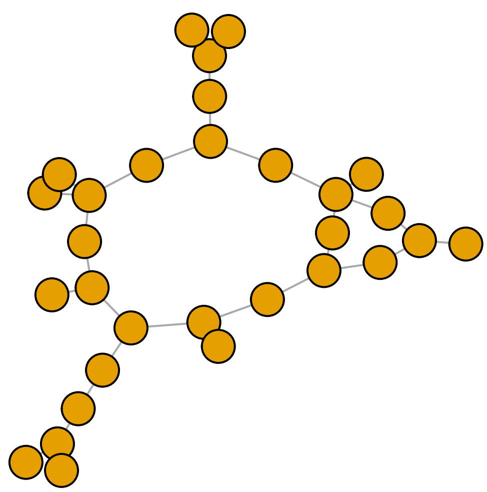
```r
head(links2)
```

```
##     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
```

```r
net2 <- graph_from_incidence_matrix(links2)

# A built-in vertex attribute 'type' shows which mode vertices belong to.
table(V(net2)$type)
```

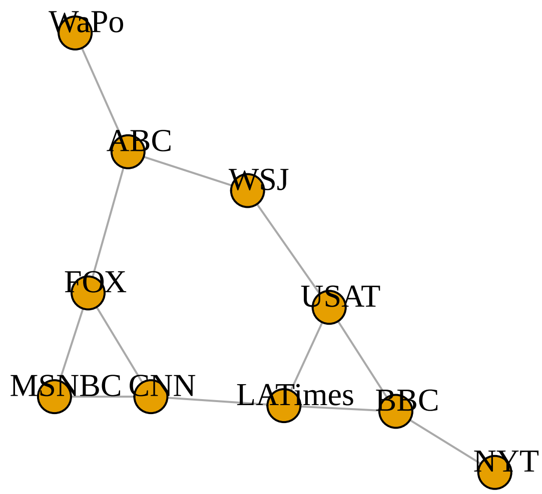
```
##
## FALSE  TRUE
##    10    20
```

```r
plot(net2,vertex.label=NA)
```

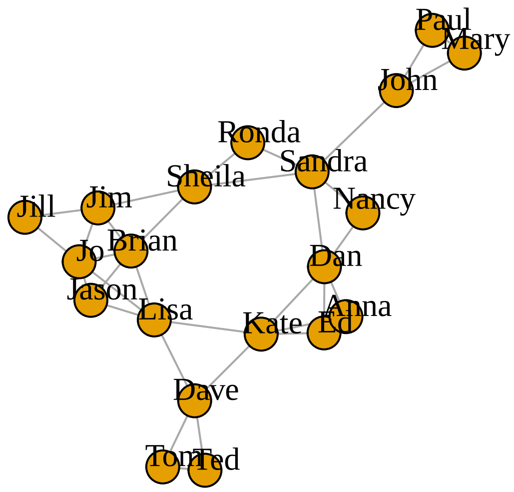


```r
# To transform a one-mode network matrix into an igraph object,
# use graph_from_adjacency_matrix()

# We can also easily generate bipartite projections for the two-mode network:
# (co-memberships are easy to calculate by multiplying the network matrix by
# its transposed matrix, or using igraph's bipartite.projection function)

net2.bp <- bipartite.projection(net2)

# We can calculate the projections manually as well:
#   as_incidence_matrix(net2)  %*% t(as_incidence_matrix(net2))
# t(as_incidence_matrix(net2)) %*%   as_incidence_matrix(net2)
```

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



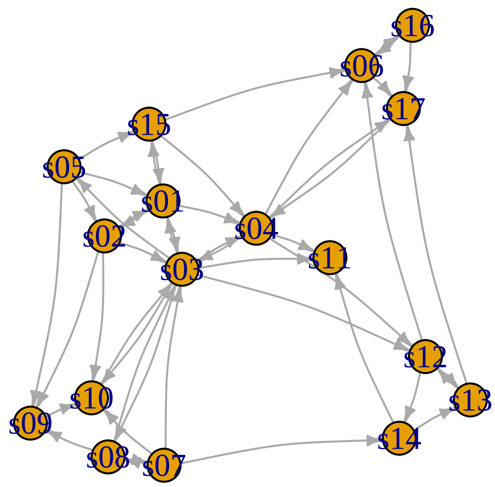

```r
plot(net2.bp$proj2, vertex.label.color="black", vertex.label.dist=1,
     vertex.label=nodes2$media[ is.na(nodes2$media.type)])
```



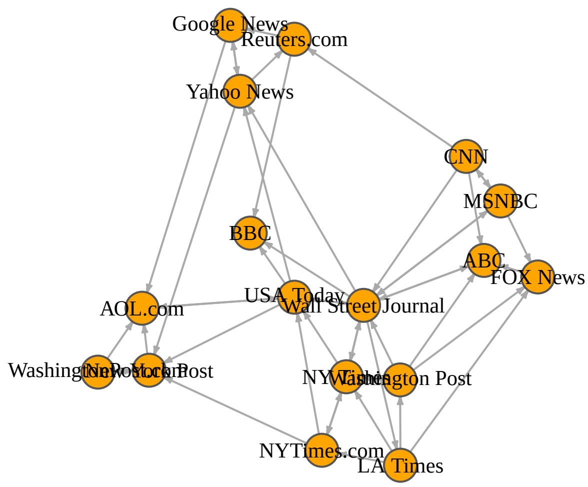

```
# ================ 5. Plotting networks with igraph ================


#  ------->> Plot parameters in igraph --------

# Plotting with igraph: node options (starting with 'vertex.') and edge options
# (starting with 'edge.'). A list of options is included in your handout.
?igraph.plotting
```

```
## starting httpd help server ... done
```

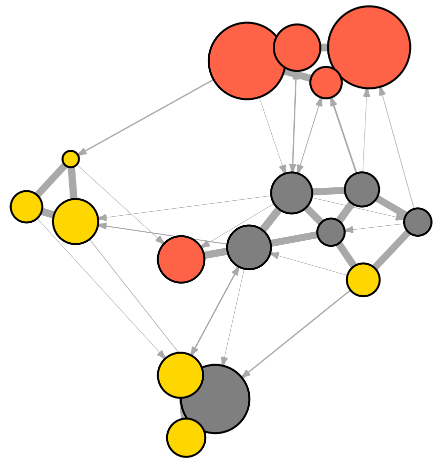
```
# We can set the node & edge options in two ways - one is to specify
# them in the plot() function, as we are doing below.

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



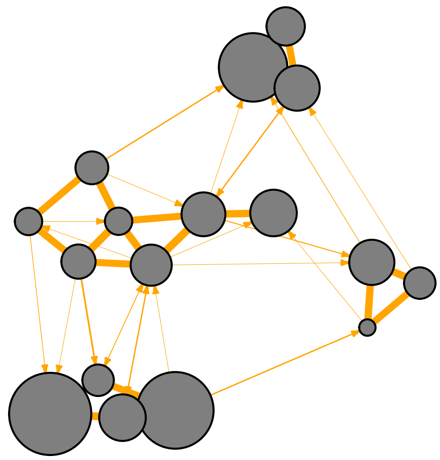

```
# Set node color to orange and the border color to hex #555555
# Replace the vertex label with the node names stored in "media"
plot(net, edge.arrow.size=.2, edge.curved=0,
     vertex.color="orange", vertex.frame.color="#555555",
     vertex.label=V(net)$media, vertex.label.color="black",
     vertex.label.cex=.7)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



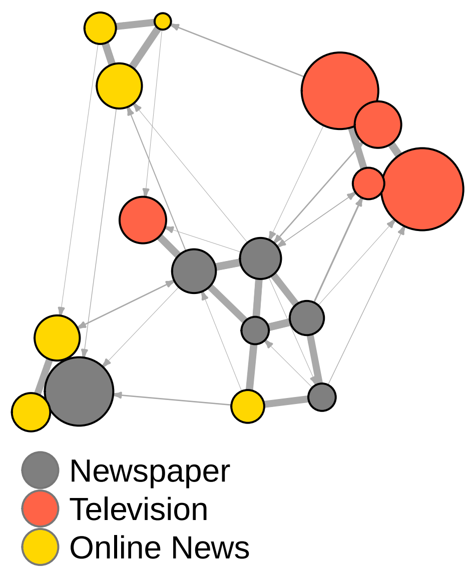

```r
# The second way to set attributes is to add them to the igraph object.

# 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"

plot(net)
```



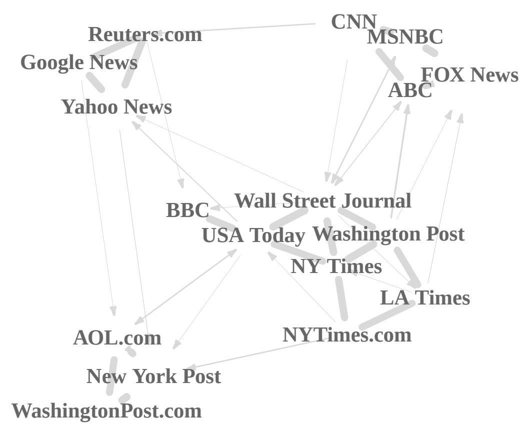
```
# We can also override the attributes explicitly in the plot:
plot(net, edge.color="orange", vertex.color="gray50")
```



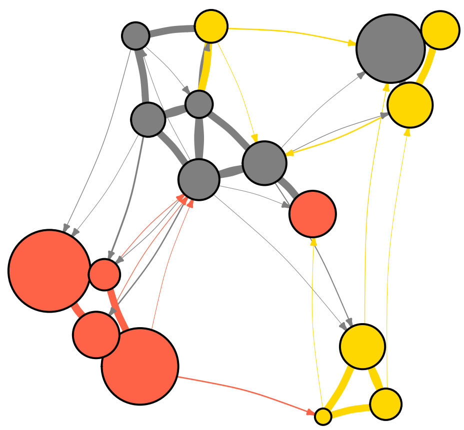
```r
# We can also add a legend explaining the meaning of the colors we used:
plot(net)
legend(x=-1.1, y=-1.1, c("Newspaper","Television", "Online News"), pch=21,
       col="#777777", pt.bg=colrs, pt.cex=2.5, bty="n", ncol=1)
```



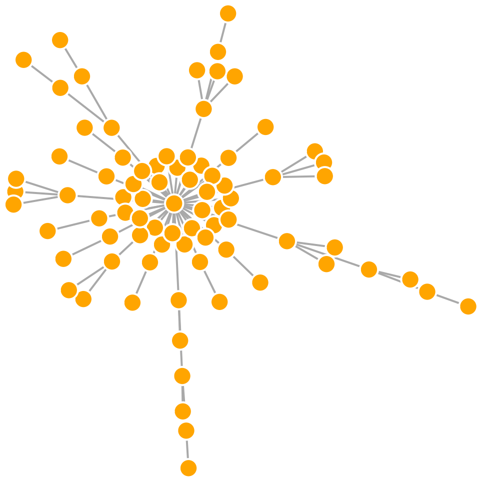

```
# Sometimes, especially with semantic networks, we may be interested in
# plotting only the labels of the nodes:

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")
```



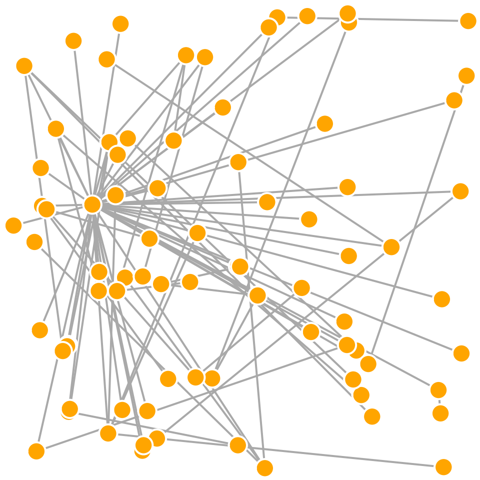

```
# Let's color the edges of the graph based on their source node color.
# We'll get the starting node for each edge with "ends()".
edge.start <- ends(net, es=E(net), names=F)[,1]
edge.col <- V(net)$color[edge.start]

plot(net, edge.color=edge.col, edge.curved=.1)
```



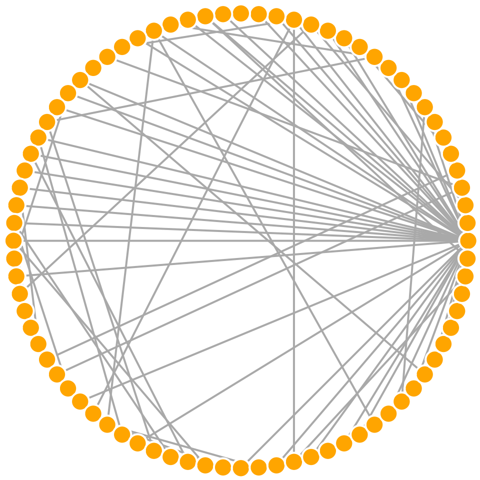
```
#  ------->> Network Layouts --------

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

# Let's generate a slightly larger 80-node graph.

net.bg <- sample_pa(80, 1.2)
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)
```



```
# You can set the layout in the plot function:
plot(net.bg, layout=layout_randomly)
```



```
# Or calculate the vertex coordinates in advance:
l <- layout_in_circle(net.bg)
plot(net.bg, layout=l)
```



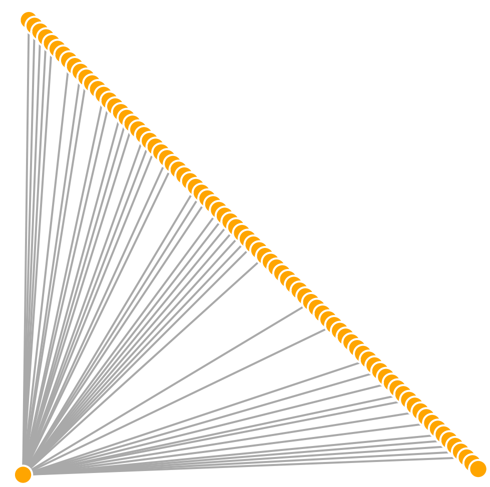
```
# l is simply a matrix of x,y coordinates (N x 2) for the N nodes in the graph.
# You can generate your own:
l
```

```
##               [,1]          [,2]
##  [1,]  1.000000e+00  0.000000e+00
##  [2,]  9.969173e-01  7.845910e-02
##  [3,]  9.876883e-01  1.564345e-01
##  [4,]  9.723699e-01  2.334454e-01
##  [5,]  9.510565e-01  3.090170e-01
##  [6,]  9.238795e-01  3.826834e-01
##  [7,]  8.910065e-01  4.539905e-01
##  [8,]  8.526402e-01  5.224986e-01
##  [9,]  8.090170e-01  5.877853e-01
## [10,]  7.604060e-01  6.494480e-01
## [11,]  7.071068e-01  7.071068e-01
## [12,]  6.494480e-01  7.604060e-01
## [13,]  5.877853e-01  8.090170e-01
## [14,]  5.224986e-01  8.526402e-01
## [15,]  4.539905e-01  8.910065e-01
## [16,]  3.826834e-01  9.238795e-01
## [17,]  3.090170e-01  9.510565e-01
## [18,]  2.334454e-01  9.723699e-01
## [19,]  1.564345e-01  9.876883e-01
## [20,]  7.845910e-02  9.969173e-01
## [21,]  6.123032e-17  1.000000e+00
```

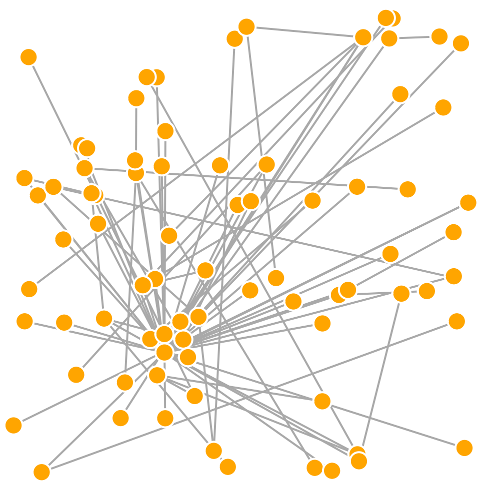
```
## [22,] -7.845910e-02  9.969173e-01
## [23,] -1.564345e-01  9.876883e-01
## [24,] -2.334454e-01  9.723699e-01
## [25,] -3.090170e-01  9.510565e-01
## [26,] -3.826834e-01  9.238795e-01
## [27,] -4.539905e-01  8.910065e-01
## [28,] -5.224986e-01  8.526402e-01
## [29,] -5.877853e-01  8.090170e-01
## [30,] -6.494480e-01  7.604060e-01
## [31,] -7.071068e-01  7.071068e-01
## [32,] -7.604060e-01  6.494480e-01
## [33,] -8.090170e-01  5.877853e-01
## [34,] -8.526402e-01  5.224986e-01
## [35,] -8.910065e-01  4.539905e-01
## [36,] -9.238795e-01  3.826834e-01
## [37,] -9.510565e-01  3.090170e-01
## [38,] -9.723699e-01  2.334454e-01
## [39,] -9.876883e-01  1.564345e-01
## [40,] -9.969173e-01  7.845910e-02
## [41,] -1.000000e+00  1.224606e-16
## [42,] -9.969173e-01 -7.845910e-02
## [43,] -9.876883e-01 -1.564345e-01
## [44,] -9.723699e-01 -2.334454e-01
## [45,] -9.510565e-01 -3.090170e-01
## [46,] -9.238795e-01 -3.826834e-01
## [47,] -8.910065e-01 -4.539905e-01
## [48,] -8.526402e-01 -5.224986e-01
## [49,] -8.090170e-01 -5.877853e-01
## [50,] -7.604060e-01 -6.494480e-01
## [51,] -7.071068e-01 -7.071068e-01
## [52,] -6.494480e-01 -7.604060e-01
## [53,] -5.877853e-01 -8.090170e-01
## [54,] -5.224986e-01 -8.526402e-01
## [55,] -4.539905e-01 -8.910065e-01
## [56,] -3.826834e-01 -9.238795e-01
## [57,] -3.090170e-01 -9.510565e-01
## [58,] -2.334454e-01 -9.723699e-01
## [59,] -1.564345e-01 -9.876883e-01
## [60,] -7.845910e-02 -9.969173e-01
## [61,] -1.836910e-16 -1.000000e+00
## [62,]  7.845910e-02 -9.969173e-01
## [63,]  1.564345e-01 -9.876883e-01
## [64,]  2.334454e-01 -9.723699e-01
## [65,]  3.090170e-01 -9.510565e-01
## [66,]  3.826834e-01 -9.238795e-01
## [67,]  4.539905e-01 -8.910065e-01
## [68,]  5.224986e-01 -8.526402e-01
## [69,]  5.877853e-01 -8.090170e-01
## [70,]  6.494480e-01 -7.604060e-01
## [71,]  7.071068e-01 -7.071068e-01
## [72,]  7.604060e-01 -6.494480e-01
## [73,]  8.090170e-01 -5.877853e-01
## [74,]  8.526402e-01 -5.224986e-01
## [75,]  8.910065e-01 -4.539905e-01
```

```
## [76,]  9.238795e-01 -3.826834e-01
## [77,]  9.510565e-01 -3.090170e-01
## [78,]  9.723699e-01 -2.334454e-01
## [79,]  9.876883e-01 -1.564345e-01
## [80,]  9.969173e-01 -7.845910e-02
```

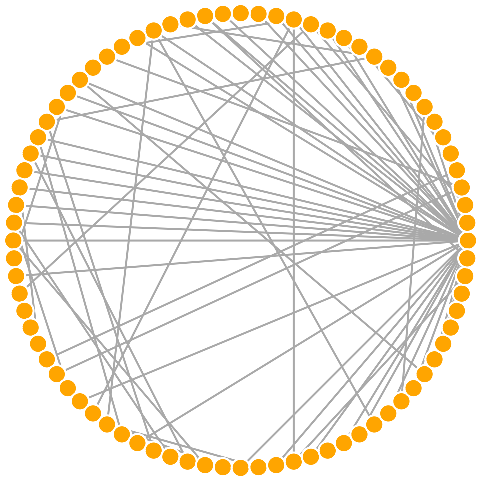
```
l <- cbind(1:vcount(net.bg), c(1, vcount(net.bg):2))
plot(net.bg, layout=l)
```



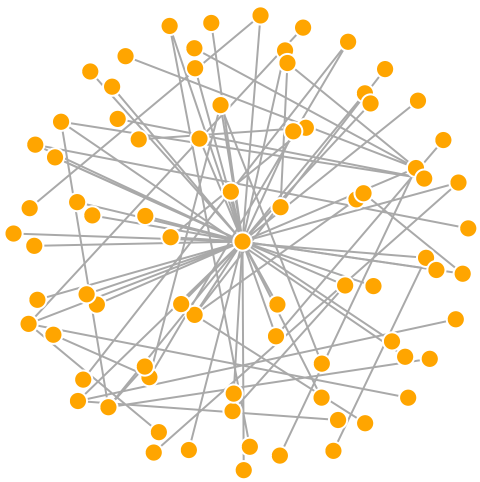
```
# This layout is just an example and not very helpful - thankfully
# igraph has a number of built-in layouts, including:

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



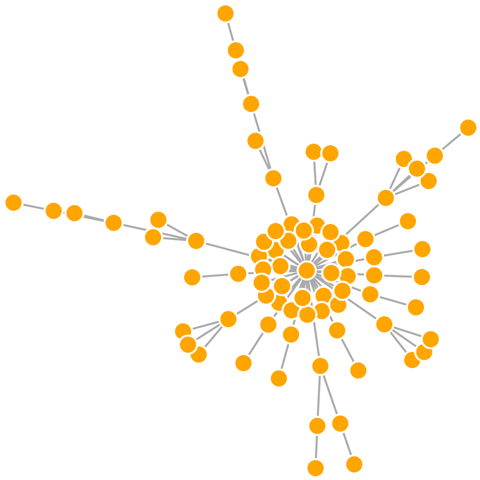
```
# Circle layout
l <- layout_in_circle(net.bg)
plot(net.bg, layout=l)
```



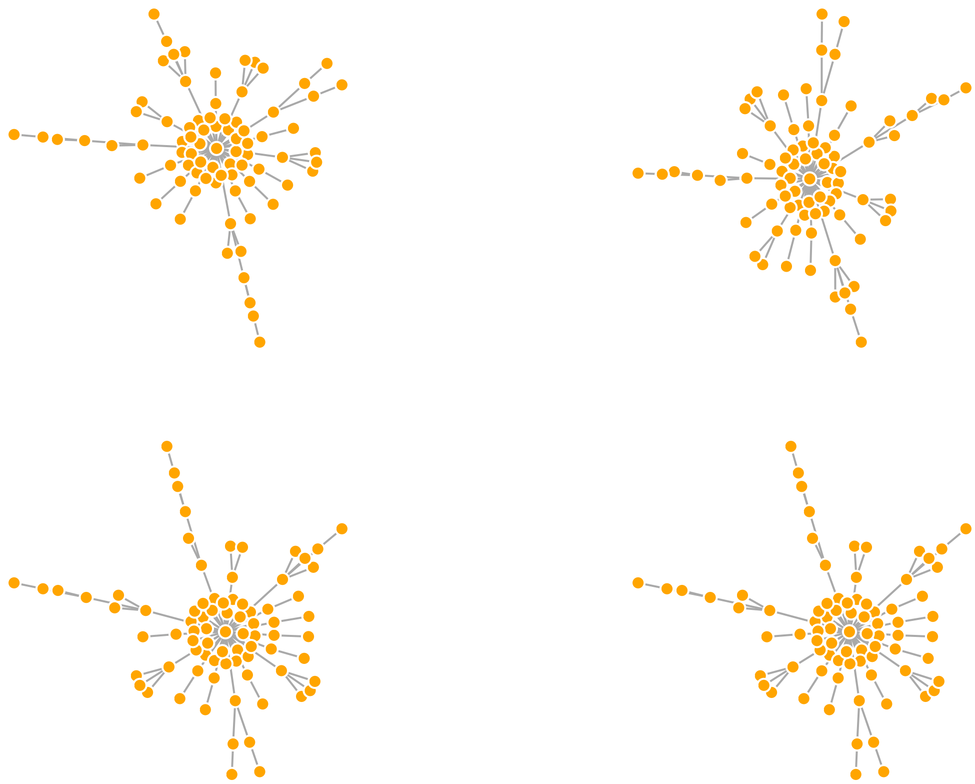
```r
# 3D sphere layout
l <- layout_on_sphere(net.bg)
plot(net.bg, layout=l)
```



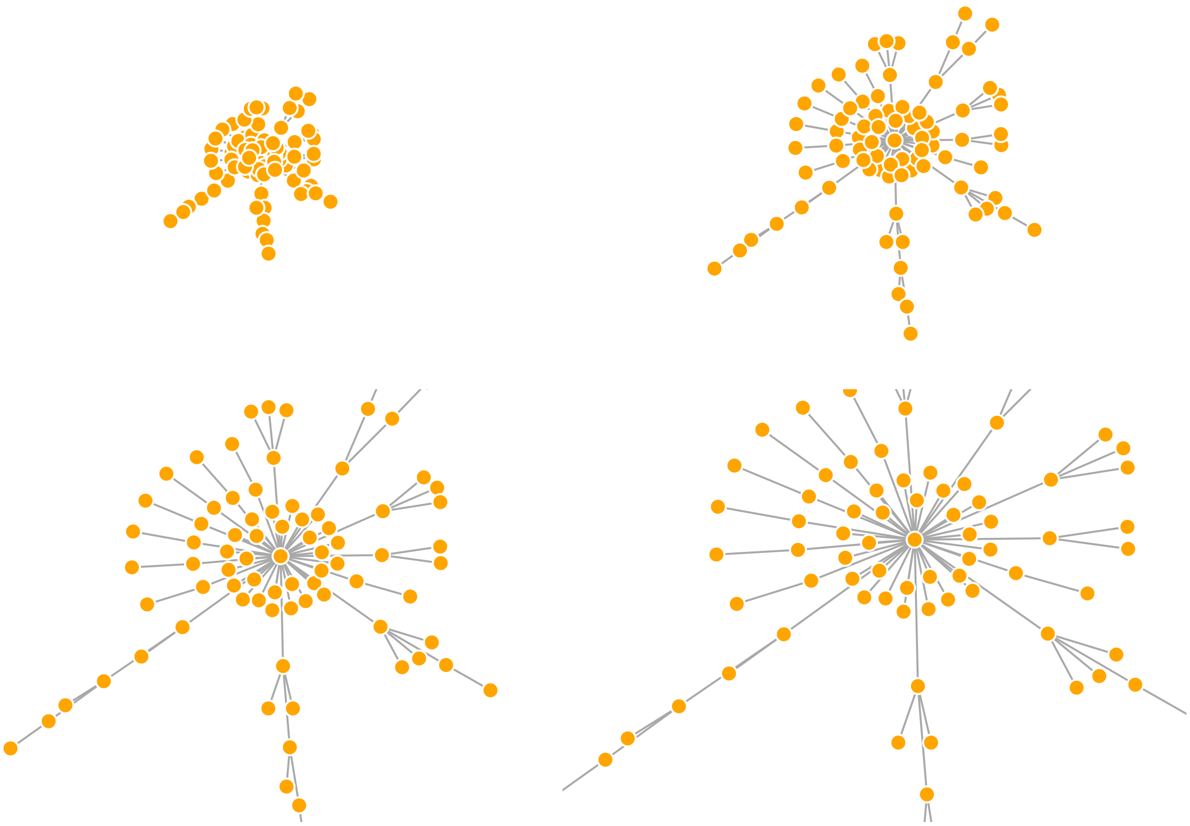
```
# The Fruchterman-Reingold force-directed algorithm
# Nice but slow, most often used in graphs smaller than ~1000 vertices.
l <- layout_with_fr(net.bg)
plot(net.bg, layout=l)
```



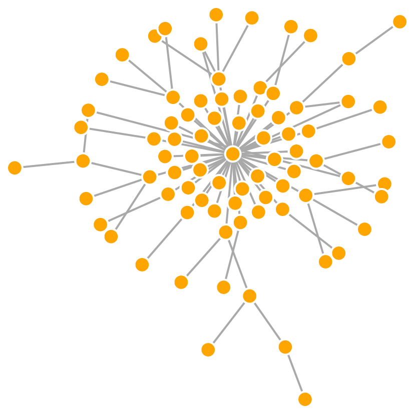
```r
# You will also notice that the layout is not deterministic - different runs
# will result in slightly different configurations. Saving the layout in l
# allows us to get the exact same result multiple times.
par(mfrow=c(2,2), mar=c(1,1,1,1))
plot(net.bg, layout=layout_with_fr)
plot(net.bg, layout=layout_with_fr)
plot(net.bg, layout=l)
plot(net.bg, layout=l)
```

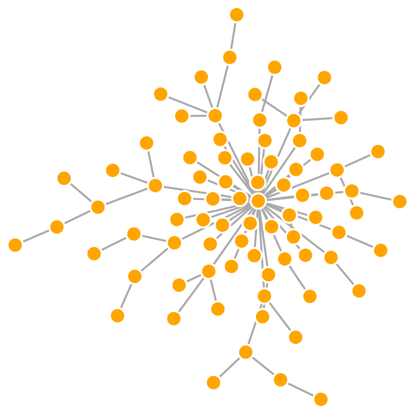


```r
# dev.off()

# By default, the coordinates of the plots are rescaled to the [-1,1] interval
# for both x and y. You can change that with the parameter "rescale=FALSE"
# and rescale your plot manually by multiplying the coordinates by a scalar.
# You can use norm_coords to normalize the plot with the boundaries you want.

# Get the layout coordinates:
l <- layout_with_fr(net.bg)
# Normalize them so that they are in the -1, 1 interval:
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*0.8)
plot(net.bg, rescale=F, layout=l*1.2)
plot(net.bg, rescale=F, layout=l*1.6)
```



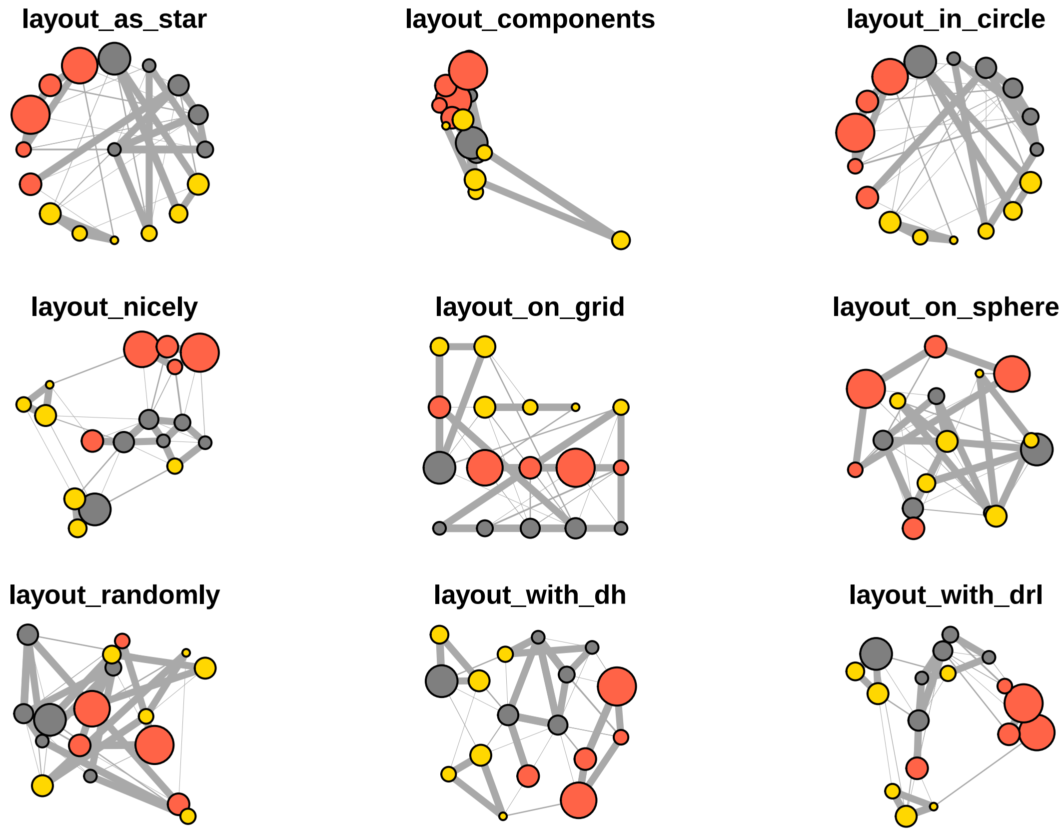
```r
# dev.off()

# Another popular force-directed algorithm that produces nice results for
# connected graphs is Kamada Kawai. Like Fruchterman Reingold, it attempts to
# minimize the energy in a spring system.

l <- layout_with_kk(net.bg)
plot(net.bg, layout=l)

# The LGL algorithm is for large connected graphs. Here you can specify a root -
# the node that will be placed in the middle of the layout.
plot(net.bg, layout=layout_with_lgl)

# By default, igraph uses a layout called layout_nicely which selects
# an appropriate layout algorithm based on the properties of the graph.

# Check out all available layouts in igraph:
?igraph::layout_

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))
```





```r
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"
```

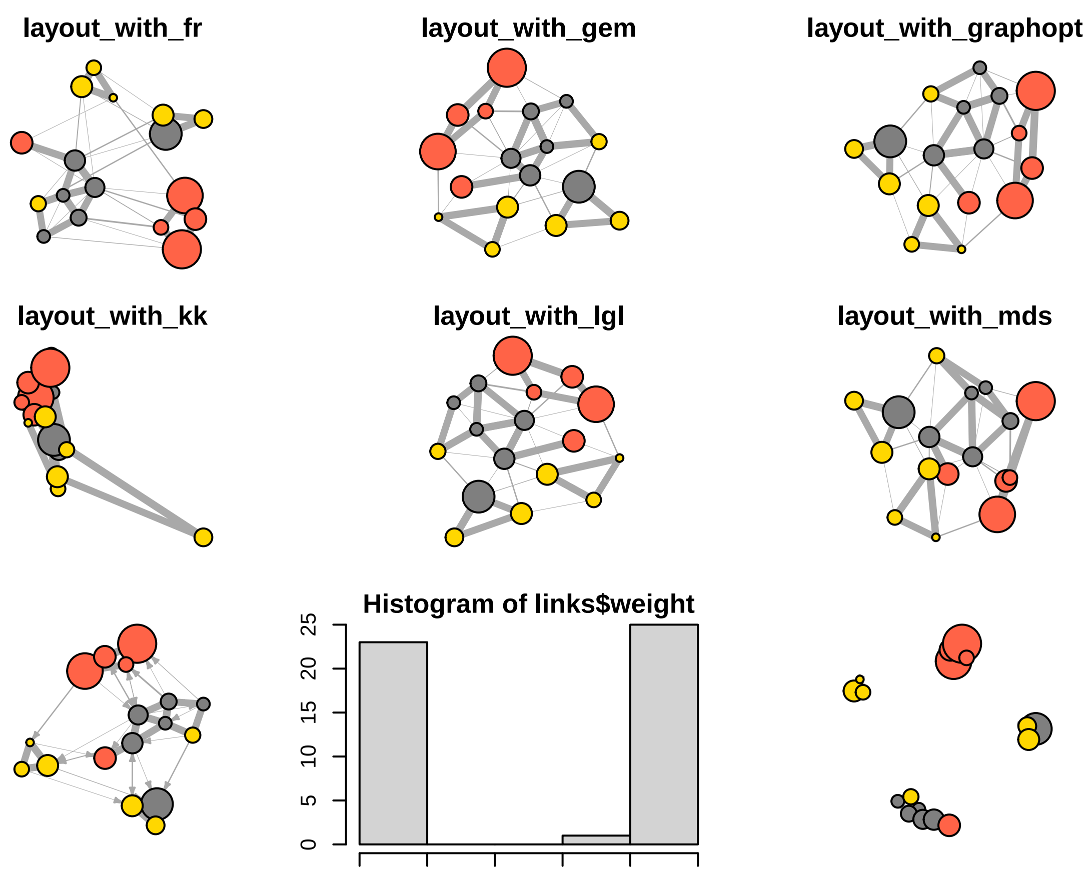
```r
# dev.off()

# ------->> Improving network plots --------
plot(net)

# Notice that this network plot is still not too helpful.
# 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.

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

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

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

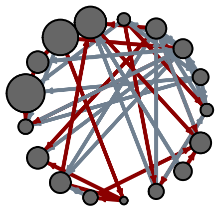
```
## [1] 9.905635
```

```r
# There are more sophisticated ways to extract the key edges,
# but for the purposes of this excercise we'll only keep ones
# that have weight higher than the mean for the network.

# We can delete edges using delete_edges(net, edges)
cut.off <- mean(links$weight)
net.sp <- delete_edges(net, E(net)[weight<cut.off])
plot(net.sp)
```

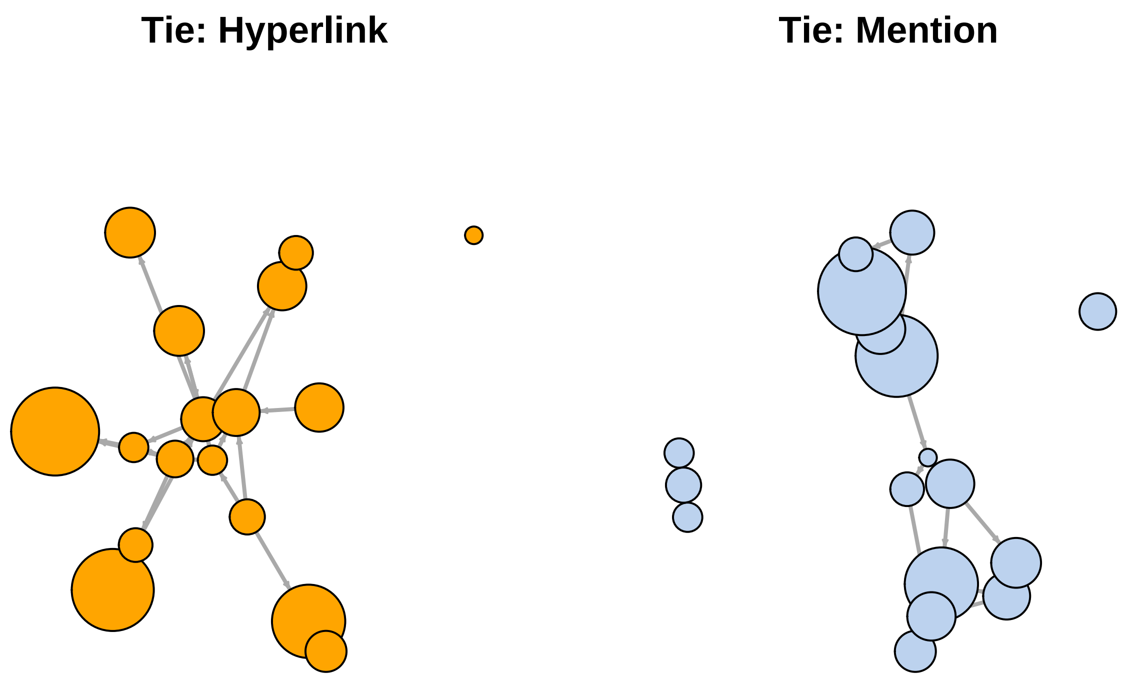



```r
# Another way to think about this is to plot the two tie types
# (hyperlik & mention) separately:

E(net)$width <- 2
plot(net, edge.color=c("dark red", "slategrey")[(E(net)$type=="hyperlink")+1],
     vertex.color="gray40", layout=layout_in_circle)

# Another way to delete edges:
net.m <- net - E(net)[E(net)$type=="hyperlink"]
```

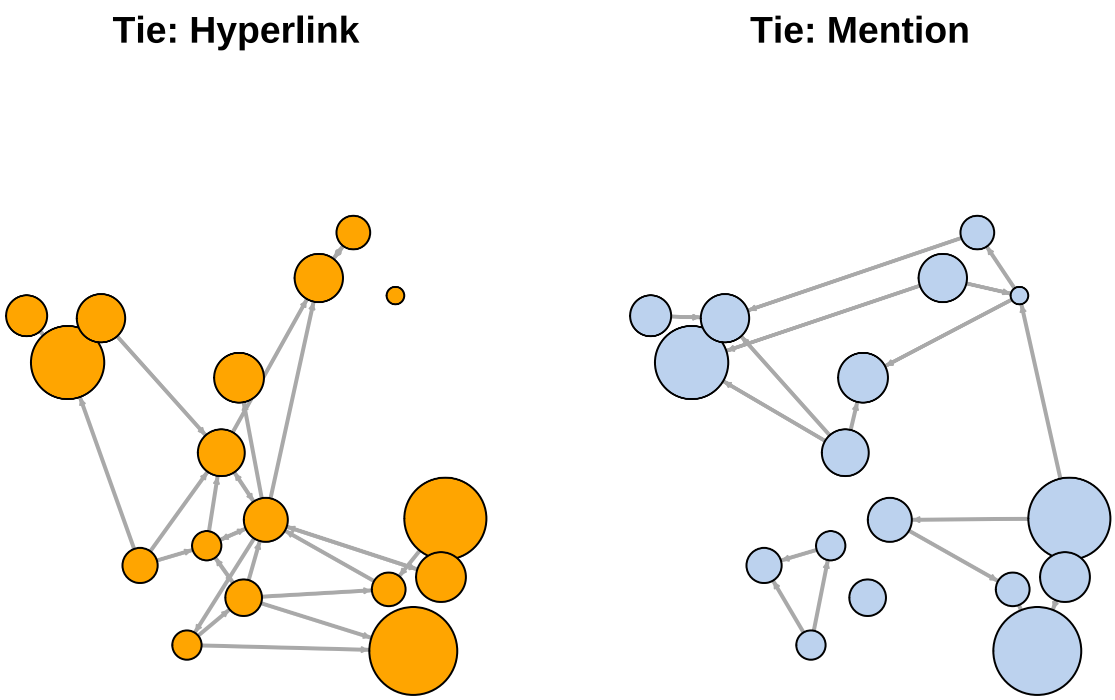
```
net.h <- net - E(net)[E(net)$type=="mention"]

# Plot the two links separately:
par(mfrow=c(1,2))
```



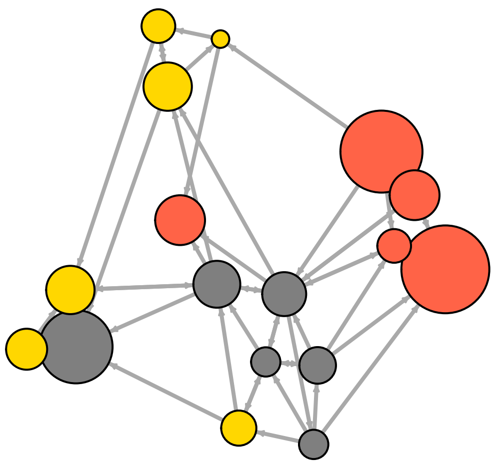
```
plot(net.h, vertex.color="orange", main="Tie: Hyperlink")
plot(net.m, vertex.color="lightsteelblue2", main="Tie: Mention")
```

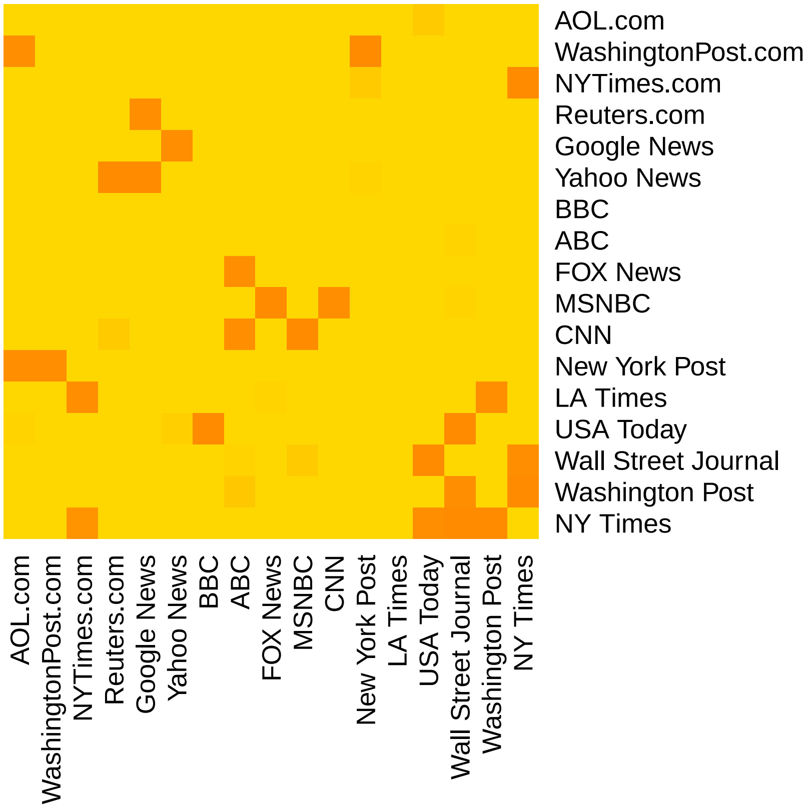



```
# dev.off()

# Make sure the nodes stay in place in both plots:
par(mfrow=c(1,2),mar=c(1,1,4,1))

l <- layout_with_fr(net)
plot(net.h, vertex.color="orange", layout=l, main="Tie: Hyperlink")
plot(net.m, vertex.color="lightsteelblue2", layout=l, main="Tie: Mention")
```




```
# dev.off()


# ------->> Interactive plotting with tkplot --------

# R and igraph offer interactive plotting capabilities
# (mostly helpful for small networks)

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


# ------->> Heatmaps as a way to represent networks --------

# A quick reminder that there are other ways to represent a network:

# Heatmap of the network matrix:
netm <- get.adjacency(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(20),
        scale="none", margins=c(10,10) )
```






```
# -------->> Plotting two-mode networks with igraph --------

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)
```

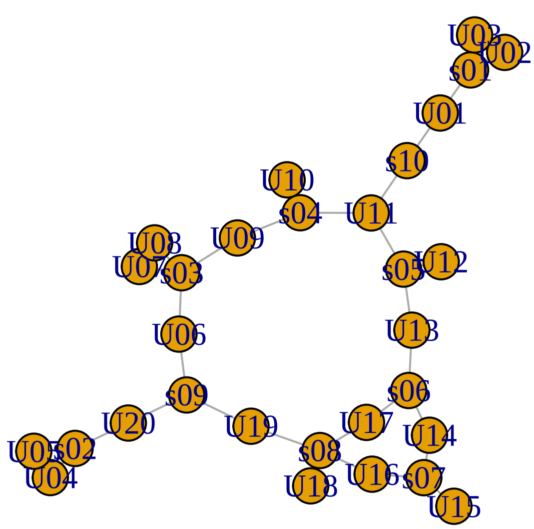
```
##     U01 U02 U03 U04 U05 U06 U07 U08 U09 U10 U11 U12 U13 U14 U15 U16 U17 U18 U19
## s01   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## s02   0   0   0   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## s03   0   0   0   0   0   1   1   1   1   0 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

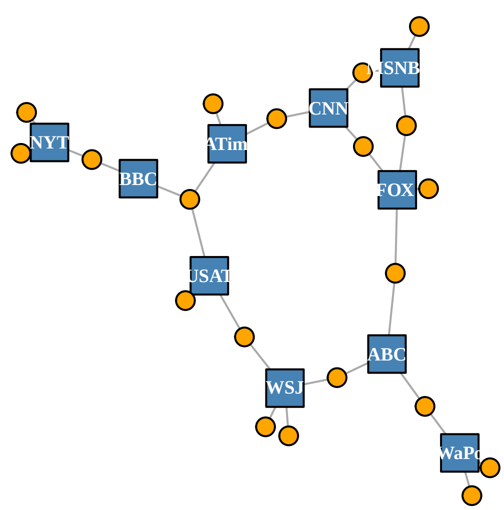
```
## s04   0
## s05   0
## s06   0
```

```r
net2
```

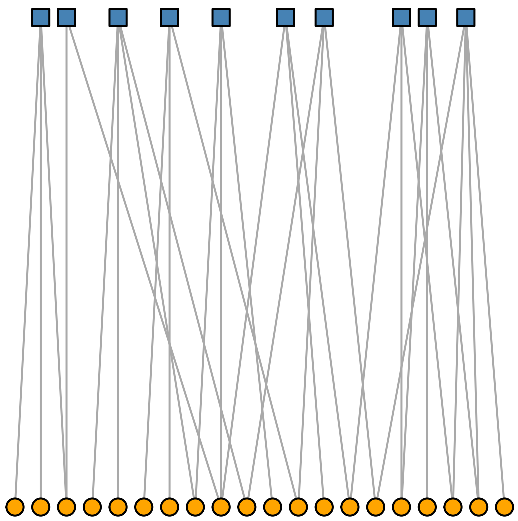
```
## IGRAPH 1bb2f9a UN-B 30 31 --
## + attr: type (v/l), name (v/c)
## + edges from 1bb2f9a (vertex names):
##  [1] s01--U01 s01--U02 s01--U03 s02--U04 s02--U05 s02--U20 s03--U06 s03--U07
##  [9] s03--U08 s03--U09 s04--U09 s04--U10 s04--U11 s05--U11 s05--U12 s05--U13
## [17] s06--U13 s06--U14 s06--U17 s07--U14 s07--U15 s07--U16 s08--U16 s08--U17
## [25] s08--U18 s08--U19 s09--U06 s09--U19 s09--U20 s10--U01 s10--U11
```

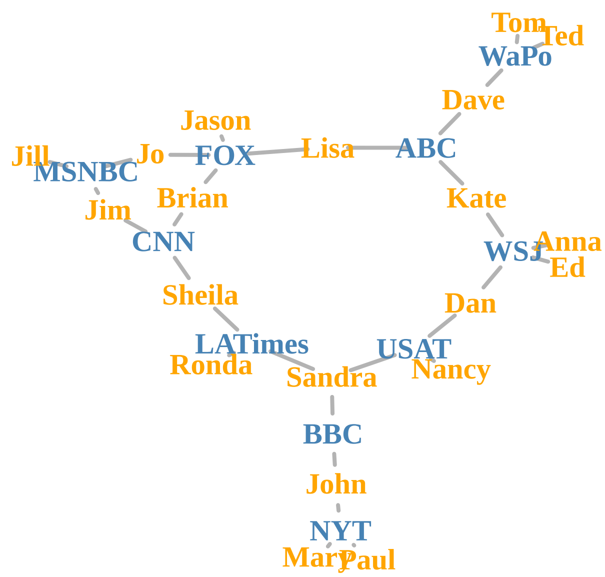
```r
plot(net2)

# This time we will make nodes look different based on their type.
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=.6
V(net2)$label.font=2

plot(net2, vertex.label.color="white", vertex.size=(2-V(net2)$type)*8)
```





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


# Using text as nodes:
par(mar=c(0,0,0,0))
plot(net2, vertex.shape="none", vertex.label=nodes2$media,
     vertex.label.color=V(net2)$color, vertex.label.font=2,
     vertex.label.cex=.95, edge.color="gray70",  edge.width=2)
```






```r
# dev.off()
```

```r
# Density
# The proportion of present edges from all possible ties.
edge_density(net, loops=F)
```

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

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

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

```r
# Reciprocity
# The proportion of reciprocated ties (for a directed network).
reciprocity(net)
```

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

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

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

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

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

```r
# Transitivity
# global - ratio of triangles (direction disregarded) to connected triples
# local - ratio of triangles to connected triples each vertex is part of
transitivity(net, type="global")  # net is treated as an undirected network
```

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

```r
transitivity(as.undirected(net, mode="collapse")) # same as above
```

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

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

```
##  [1] 0.2142857 0.4000000 0.1153846 0.1944444 0.5000000 0.2666667 0.2000000
##  [8] 0.1000000 0.3333333 0.3000000 0.3333333 0.2000000 0.1666667 0.1666667
## [15] 0.3000000 0.3333333 0.2000000
```

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

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

```r
# Triad types (per Davis & Leinhardt):
#
# 003  A, B, C, empty triad.
# 012  A->B, C
# 102  A<->B, C
# 021D A<-B->C
```

```r
# 021U A->B<-C
# 021C A->B->C
# 111D A<->B<-C
# 111U A<->B->C
# 030T A->B<-C, A->C
# 030C A<-B<-C, A->C.
# 201  A<->B<->C.
# 120D A<-B->C, A<->C.
# 120U A->B<-C, A<->C.
# 120C A->B->C, A<->C.
# 210  A->B<->C, A<->C.
# 300  A<->B<->C, A<->C, completely connected.


# Diameter (longest geodesic distance)
# Note that edge weights are used by default, unless set to NA.
diameter(net, directed=F, weights=NA)
```

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

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

```
## [1] 28
```

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

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

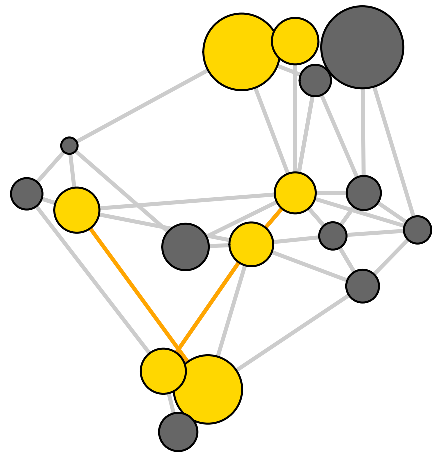
```r
# Note: vertex sequences asked to behave as a vector produce numeric index of nodes
class(diam)
```

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

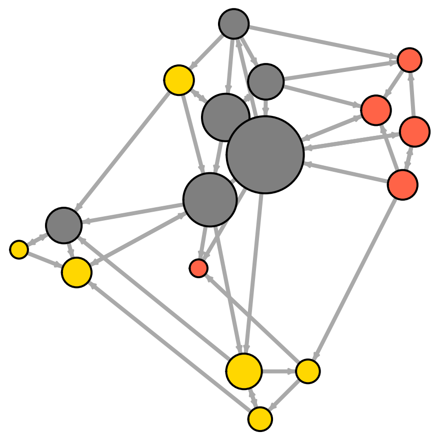
```r
as.vector(diam)
```

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

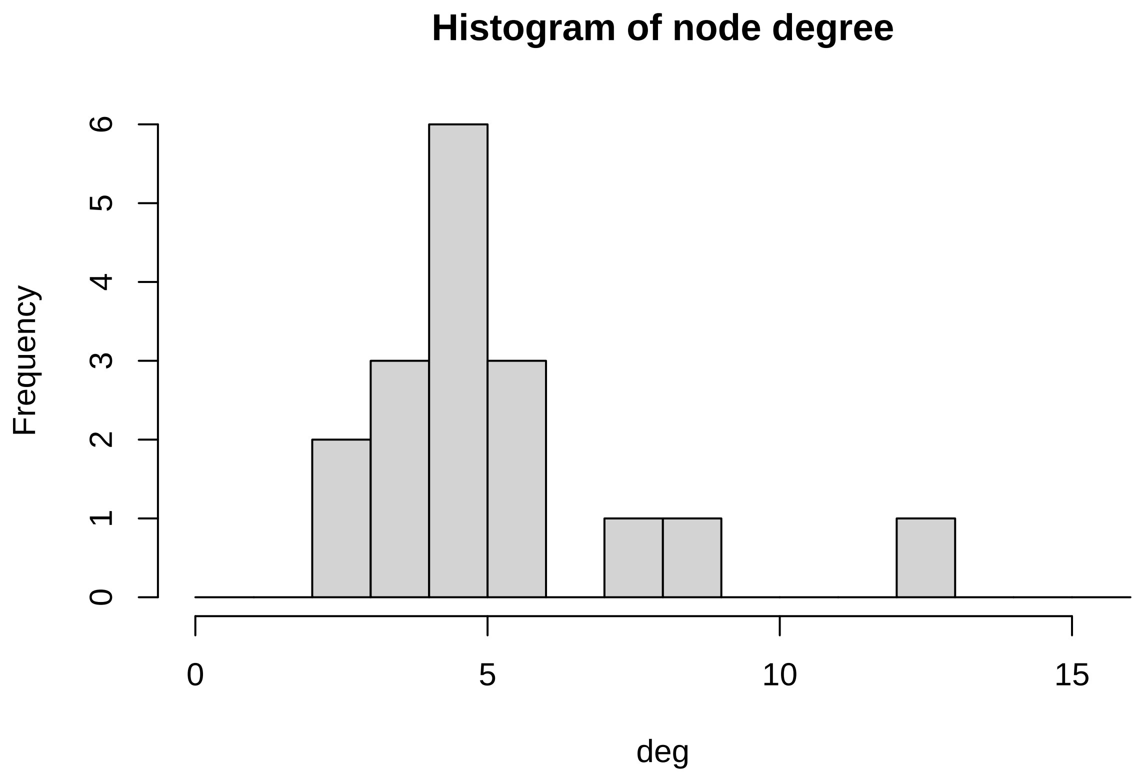
```r
# Color nodes along the diameter:
vcol <- rep("gray40", vcount(net))
vcol[diam] <- "gold"
ecol <- rep("gray80", ecount(net))
ecol[E(net, path=diam)] <- "orange"
# E(net, path=diam) finds edges along a path, here 'diam'
plot(net, vertex.color=vcol, edge.color=ecol, edge.arrow.mode=0)
```



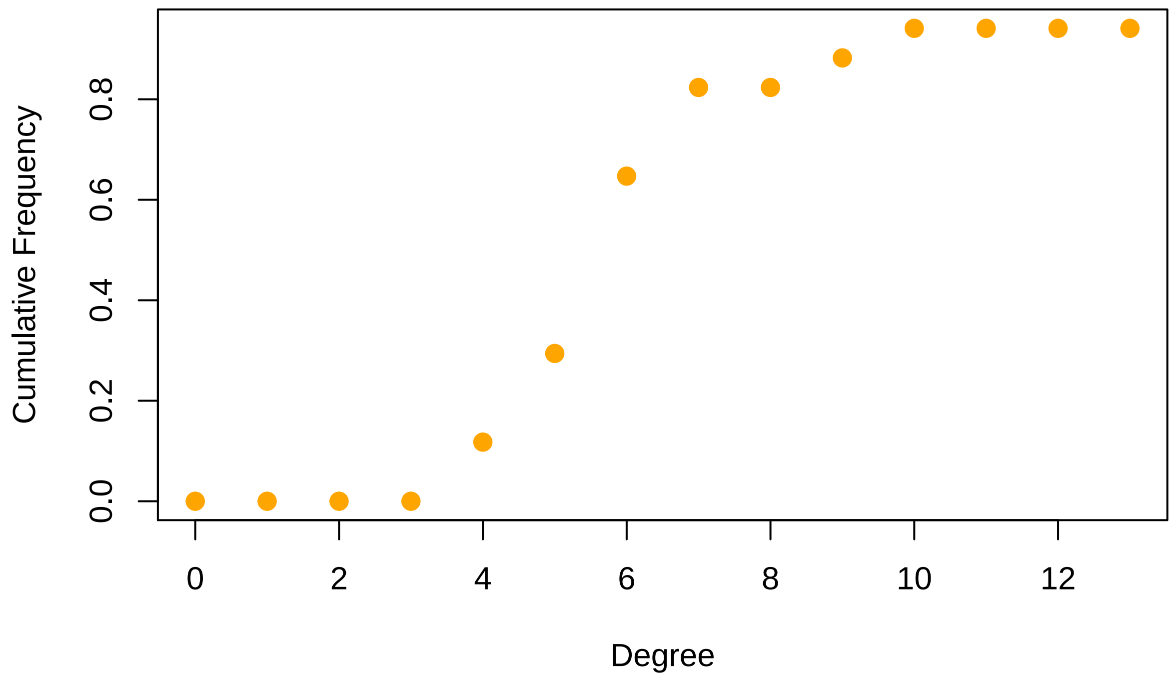
```r
# Node degrees
# 'degree' has a mode of 'in' for in-degree, 'out' for out-degree,
# and 'all' or 'total' for total degree.
deg <- degree(net, mode="all")
plot(net, vertex.size=deg*3)
```



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




```r
# 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")
```




```r
# Centrality & centralization

# Centrality functions (vertex level) and centralization functions (graph level).
# The centralization functions return "res" - vertex centrality, "centralization",
# and "theoretical_max" - maximum centralization score for a graph of that size.
# The centrality functions can run on a subset of nodes (set with the "vids" parameter)

# Degree (number of ties)
degree(net, mode="in")
```

```
## 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
```

```r
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
```

```r
# Closeness (centrality based on distance to others in the graph)
# Inverse of the node's average geodesic distance to others in the network
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
```

```r
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
```

```r
# Eigenvector (centrality proportional to the sum of connection centralities)
# Values of the first eigenvector of the graph adjacency matrix
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] 0
##
## $options$ldv
## [1] 0
##
## $options$ishift
## [1] 1
##
## $options$maxiter
## [1] 1000
##
## $options$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] 7
##
## $options$nconv
## [1] 1
##
## $options$numop
## [1] 31
##
## $options$numopb
## [1] 0
##
## $options$numreo
## [1] 18
```

```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] 0
##
## $options$ldv
## [1] 0
##
## $options$ishift
## [1] 1
##
## $options$maxiter
## [1] 1000
##
## $options$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] 7
##
## $options$nconv
```

```
## [1] 1
##
## $options$numop
## [1] 31
##
## $options$numopb
## [1] 0
##
## $options$numreo
## [1] 18
##
##
## $centralization
## [1] 0.5071775
##
## $theoretical_max
## [1] 16
```

```r
# Betweenness (centrality based on a broker position connecting others)
# (Number of geodesics that pass through the node or the edge)
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
```

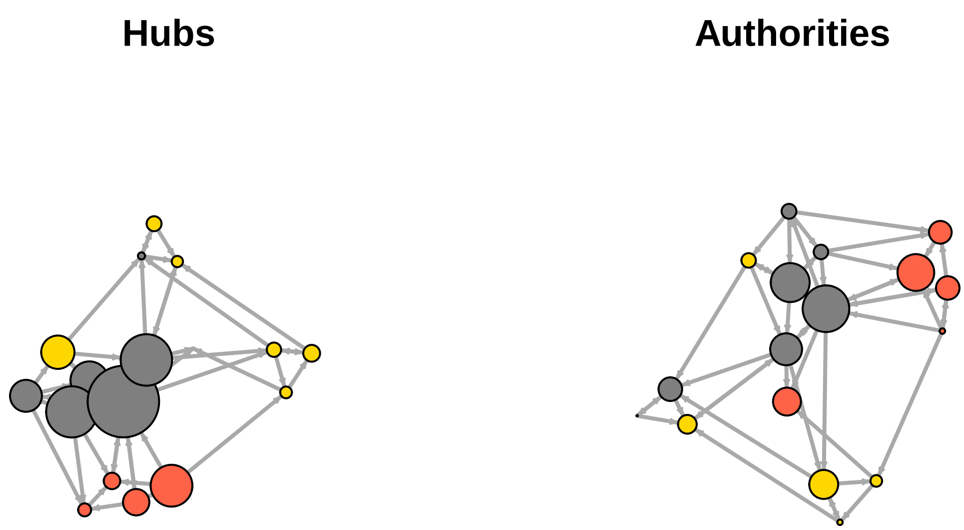
```r
edge_betweenness(net, directed=T, weights=NA)
```

```
##  [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
```

```r
# Hubs and authorities

# The hubs and authorities algorithm developed by Jon Kleinberg was initially used
# to examine web pages. Hubs were expected to contain catalogues with a large number
# of outgoing links; while authorities would get many incoming links from hubs,
# presumably because of their high-quality relevant information.

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

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




```r
# dev.off()


# ================ 7. Distances and paths ================


# Average path length
# The mean of the shortest distance between each pair of nodes in the network
# (in both directions for directed graphs).
mean_distance(net, directed=F)
```

```
## [1] 2.058824
```

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

```
## [1] 2.742188
```

```r
# We can also find the length of all shortest paths in the graph:
distances(net) # with edge weights
```

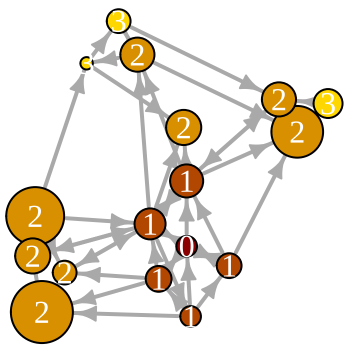
```
##     s01 s02 s03 s04 s05 s06 s07 s08 s09 s10 s11 s12 s13 s14 s15 s16 s17
## s01   0   4   2   6   1   5   3   4   3   4   3   3   9   4   7  26   8
## s02   4   0   4   8   3   7   5   6   1   5   5   5  11   6   9  28  10
## s03   2   4   0   4   1   3   1   2   3   2   1   1   7   2   5  24 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

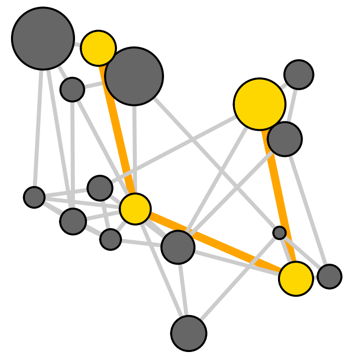
```r
distances(net, weights=NA) # ignore weights
```

```
##     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
```

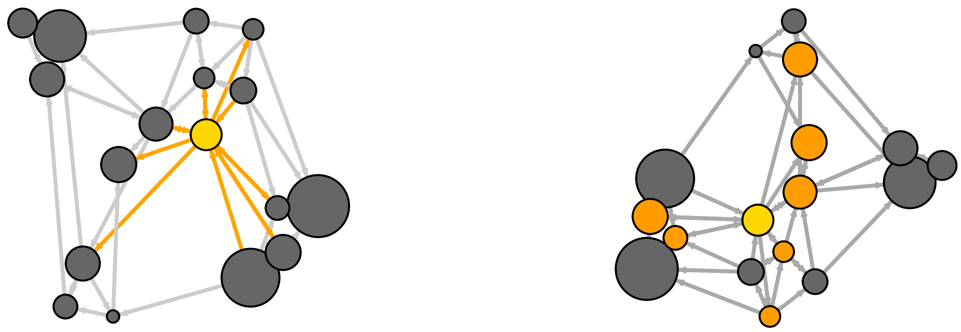
```r
# We can extract the distances to a node or set of nodes we are interested in.
# Here we will get the distance of every media from the New York Times.
```

```r
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")

# We can also find the shortest path between specific nodes.
# Say here between MSNBC and the New York Post:
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$epath)] <- "orange"
# Generate edge width variable to plot the path:
ew <- rep(2, ecount(net))
ew[unlist(news.path$epath)] <- 4
# Generate node color variable to plot the path:
vcol <- rep("gray40", vcount(net))
vcol[unlist(news.path$vpath)] <- "gold"

plot(net, vertex.color=vcol, edge.color=ecol,
     edge.width=ew, edge.arrow.mode=0)
```





```
# Identify the edges going into or out of a vertex, for instance the WSJ.
# For a single node, use 'incident()', for multiple nodes use 'incident_edges()'
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)


# We can also easily identify the immediate neighbors of a vertex, say WSJ.
# The 'neighbors' function finds all nodes one step out from the focal actor.
# To find the neighbors for multiple nodes, use 'adjacent_vertices()'.
# To find node neighborhoods going more than one step out, use function 'ego()'
# with parameter 'order' set to the number of steps out to go from the focal node(s).

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)
```



```r
# Special operators for the indexing of edge sequences: %--%, %->%, %<-%
# E(network)[X %--% Y] selects edges between vertex sets X and Y, ignoring direction
# E(network)[X %->% Y] selects edges from vertex sets X to vertex set Y
# E(network)[X %->% Y] selects edges from vertex sets Y to vertex set X

# For example, select edges from newspapers to online sources:
E(net)[ V(net)[type.label=="Newspaper"] %->% V(net)[type.label=="Online"] ]
```

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

```r
# Cocitation (for a couple of nodes, how many shared nominations they have)
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
```

```r
# ================ 8. Subgroups and communities ================

# Converting 'net' to an undirected network.
# There are several ways to do that: we can create an undirected link between any pair
# of connected nodes (mode="collapse"), or create an undirected link for each directed
# one (mode="each"), or create an undirected link for each symmetric link (mode="mutual").
# In cases when A -> B and B -> A are collapsed into a single undirected link, we
# need to specify what to do with the edge attributes. Here we have said that
# the 'weight' of links should be summed, and all other edge attributes ignored.

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


#  ------->> Cliques --------

# Find cliques (complete subgraphs of an undirected graph)
cliques(net.sym) # list of cliques
```

```
## [[1]]
## + 1/17 vertex, named, from 1fba0f0:
## [1] s03
##
## [[2]]
## + 1/17 vertex, named, from 1fba0f0:
## [1] s06
##
## [[3]]
## + 1/17 vertex, named, from 1fba0f0:
## [1] s14
##
## [[4]]
## + 1/17 vertex, named, from 1fba0f0:
## [1] s09
##
## [[5]]
## + 1/17 vertex, named, from 1fba0f0:
## [1] s04
##
## [[6]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s04 s06
##
## [[7]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s03 s04
##
```

```
## [[8]]
## + 1/17 vertex, named, from 1fba0f0:
## [1] s05
##
## [[9]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s05 s09
##
## [[10]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s03 s05
##
## [[11]]
## + 1/17 vertex, named, from 1fba0f0:
## [1] s13
##
## [[12]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s13 s14
##
## [[13]]
## + 1/17 vertex, named, from 1fba0f0:
## [1] s10
##
## [[14]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s09 s10
##
## [[15]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s03 s10
##
## [[16]]
## + 1/17 vertex, named, from 1fba0f0:
## [1] s16
##
## [[17]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s06 s16
##
## [[18]]
## + 1/17 vertex, named, from 1fba0f0:
## [1] s08
##
## [[19]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s08 s09
##
## [[20]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s03 s08
##
## [[21]]
## + 1/17 vertex, named, from 1fba0f0:
```

```
## [1] s01
##
## [[22]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s01 s05
##
## [[23]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s01 s03 s05
##
## [[24]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s01 s04
##
## [[25]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s01 s03 s04
##
## [[26]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s01 s03
##
## [[27]]
## + 1/17 vertex, named, from 1fba0f0:
## [1] s17
##
## [[28]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s16 s17
##
## [[29]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s06 s16 s17
##
## [[30]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s13 s17
##
## [[31]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s04 s17
##
## [[32]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s04 s06 s17
##
## [[33]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s06 s17
##
## [[34]]
## + 1/17 vertex, named, from 1fba0f0:
## [1] s12
##
```

```
## [[35]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s12 s13
##
## [[36]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s12 s13 s14
##
## [[37]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s04 s12
##
## [[38]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s04 s06 s12
##
## [[39]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s03 s04 s12
##
## [[40]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s12 s14
##
## [[41]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s06 s12
##
## [[42]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s03 s12
##
## [[43]]
## + 1/17 vertex, named, from 1fba0f0:
## [1] s11
##
## [[44]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s04 s11
##
## [[45]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s03 s04 s11
##
## [[46]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s11 s14
##
## [[47]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s03 s11
##
## [[48]]
## + 1/17 vertex, named, from 1fba0f0:
```

```
## [1] s07
##
## [[49]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s07 s08
##
## [[50]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s03 s07 s08
##
## [[51]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s07 s10
##
## [[52]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s03 s07 s10
##
## [[53]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s07 s14
##
## [[54]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s03 s07
##
## [[55]]
## + 1/17 vertex, named, from 1fba0f0:
## [1] s15
##
## [[56]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s01 s15
##
## [[57]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s01 s05 s15
##
## [[58]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s01 s04 s15
##
## [[59]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s05 s15
##
## [[60]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s04 s15
##
## [[61]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s04 s06 s15
##
```

```
## [[62]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s06 s15
##
## [[63]]
## + 1/17 vertex, named, from 1fba0f0:
## [1] s02
##
## [[64]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s01 s02
##
## [[65]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s01 s02 s05
##
## [[66]]
## + 4/17 vertices, named, from 1fba0f0:
## [1] s01 s02 s03 s05
##
## [[67]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s01 s02 s03
##
## [[68]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s02 s10
##
## [[69]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s02 s09 s10
##
## [[70]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s02 s03 s10
##
## [[71]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s02 s05
##
## [[72]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s02 s05 s09
##
## [[73]]
## + 3/17 vertices, named, from 1fba0f0:
## [1] s02 s03 s05
##
## [[74]]
## + 2/17 vertices, named, from 1fba0f0:
## [1] s02 s09
##
## [[75]]
## + 2/17 vertices, named, from 1fba0f0:
```

```
## [1] s02 s03
```

```r
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
```

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

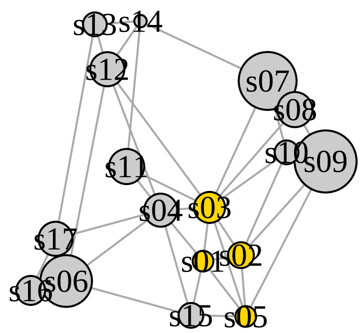
```
## [[1]]
## + 4/17 vertices, named, from 1fba0f0:
## [1] s03 s01 s02 s05
```

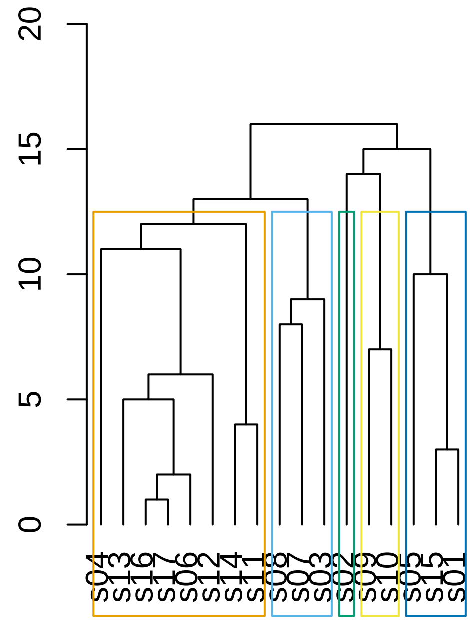
```r
vcol <- rep("grey80", vcount(net.sym))
vcol[unlist(largest_cliques(net.sym))] <- "gold"
plot(net.sym, vertex.label=V(net.sym)$name, vertex.color=vcol)



#  ------->> Communities --------

# A number of algorithms aim to detect groups that consist of densely connected nodes
# with fewer connections across groups.

# Community detection based on edge betweenness (Newman-Girvan)
# High-betweenness edges are removed sequentially (recalculating at each step)
# and the best partitioning of the network is selected.
ceb <- cluster_edge_betweenness(net)
```

```
## Warning in cluster_edge_betweenness(net): At community.c:460 :Membership vector
## will be selected based on the lowest modularity score.
```

```
## Warning in cluster_edge_betweenness(net): At community.c:467 :Modularity
## calculation with weighted edge betweenness community detection might not make
## sense -- modularity treats edge weights as similarities while edge betwenness
## treats them as distances
```

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





```r
plot(ceb, net)

# Let's examine the community detection igraph object:
class(ceb)
```

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

```r
length(ceb)     # number of communities
```

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

```r
membership(ceb) # community membership for each node
```

```
## 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
crossing(ceb, net)   # boolean vector: TRUE for edges across communities
```

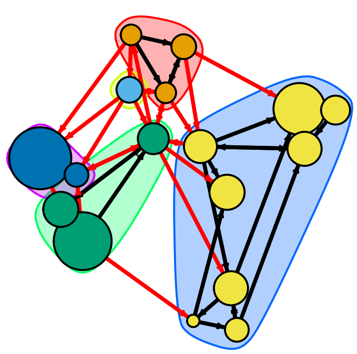
```
## s01|s02 s01|s03 s01|s04 s01|s15 s02|s01 s02|s03 s02|s09 s02|s10 s03|s01 s03|s04
##    TRUE    TRUE    TRUE   FALSE    TRUE    TRUE    TRUE    TRUE    TRUE    TRUE
## s03|s05 s03|s08 s03|s10 s03|s11 s03|s12 s04|s03 s04|s06 s04|s11 s04|s12 s04|s17
##    TRUE   FALSE    TRUE    TRUE    TRUE    TRUE   FALSE   FALSE   FALSE   FALSE
## s05|s01 s05|s02 s05|s09 s05|s15 s06|s16 s06|s17 s07|s03 s07|s08 s07|s10 s07|s14
```

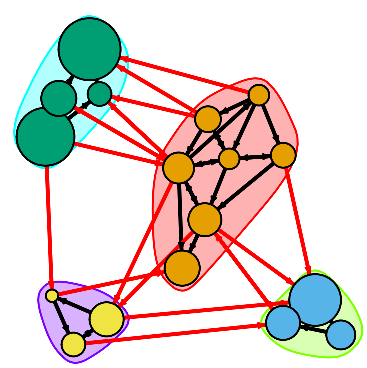
```
##   FALSE    TRUE    TRUE   FALSE   FALSE   FALSE   FALSE   FALSE    TRUE    TRUE
## s08|s03 s08|s07 s08|s09 s09|s10 s10|s03 s12|s06 s12|s13 s12|s14 s13|s12 s13|s17
##   FALSE   FALSE    TRUE   FALSE    TRUE   FALSE   FALSE   FALSE   FALSE   FALSE
## s14|s11 s14|s13 s15|s01 s15|s04 s15|s06 s16|s06 s16|s17 s17|s04
##   FALSE   FALSE   FALSE    TRUE    TRUE   FALSE   FALSE   FALSE
```

```
modularity(ceb) # how modular the graph partitioning is
```

```
## [1] 0.292476
```

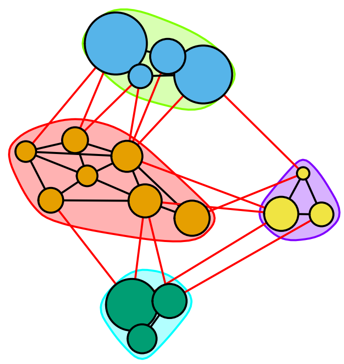
```
# High modularity for a partitioning reflects dense connections within communities
# and sparse connections across communities.


# Community detection based on propagating labels
# Assigns node labels, randomizes, and replaces each vertex's label with
# the label that appears most frequently among neighbors. Repeated until
# each vertex has the most common label of its neighbors.
clp <- cluster_label_prop(net)
plot(clp, net)
```

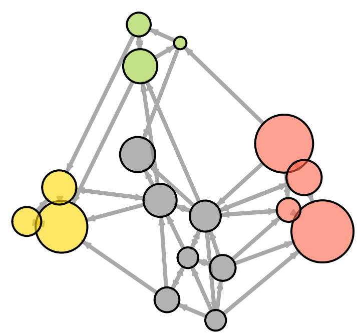




```
# Community detection based on greedy optimization of modularity
cfg <- cluster_fast_greedy(as.undirected(net))
plot(cfg, as.undirected(net))
```

```
# We can also plot the communities without relying on their built-in plot:
V(net)$community <- cfg$membership
colrs <- adjustcolor( c("gray50", "tomato", "gold", "yellowgreen"), alpha=.6)
plot(net, vertex.color=colrs[V(net)$community])
```





```
# K-core decomposition
# The k-core is the maximal subgraph in which every node has degree of at least k
# This also means that the (k+1)-core will be a subgraph of the k-core.
# The result here gives the coreness of each vertex in the network.
kc <- coreness(net, mode="all")
plot(net, vertex.size=kc*6, vertex.label=kc, vertex.color=colrs[kc])


# ================ 9. Assortativity and Homophily ================

# Assortativity (homophily)
# The tendency of nodes to connect to others who are similar on some variable.
# assortativity_nominal() is for categorical variables (labels)
# assortativity() is for ordinal and above variables
# assortativity_degree() checks assortativity in node degrees

V(net)$type.label
```

```
##  [1] "Newspaper" "Newspaper" "Newspaper" "Newspaper" "Newspaper" "Newspaper"
```

```
##  [7] "TV"       "TV"       "TV"       "TV"       "TV"       "Online"
## [13] "Online"   "Online"   "Online"   "Online"   "Online"
```

```r
V(net)$media.type
```

```
##  [1] 1 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 3
```

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

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

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

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

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

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

```r
# ================ The End ================
```