# Wk7.Networks

*4/9/2019*

Todays worksheet is in pdf format. Copy and paste each of the code chunks into an R script to run the code.

To visualise network graphs we will use the `igraph` package and not `ggplot2`. This package is more developed that building networks in `ggplot`. But take care, the way the graphs are build don't follow the same structure as in `ggplot`.

## Workshop Set-up

### Setup

Install and load the following packages.

```
#install.packages("igraph")
#install.packages("network")
#install.packages("sna")
#install.packages("visNetwork")
#install.packages("threejs")
#install.packages("networkD3")
#install.packages("RColorBrewer")
#install.packages("EpiContactTrace")
#install.packages("reshape")
```

## Part 1

We will use two small datasets about media organisations in this workshop. The first contains information about hyperlinks to, and mentions about, a range of media sources. The second dataset contains information about links between media sources and media consumers.

### Dataset 1: Hyperlinks & Mentions

#### Load data

Download and save the `Dataset1-Media-Example-NODES.csv` and `Dataset1-Media-Example-EDGES.csv`to your working directory. Note: you may need to delete `data/` from the file path in the code below depending on where you save the csv files.

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

Have a quick look at the data

```
head(nodes)
head(links)
#or
glimpse(nodes)
glimpse(links)
```

### Create an igraph object

To use the igraph package, to generate a network visualisatiom, first you must create an igraph network object using the `graph_from_data_frame()` function. This function takes two arguments `d` and `vertices`:

- `d` - describes the edges of the network. Its first two columns are the IDs of the source and the target node for each edge. The following columns are edge attributes (weight, type, label, or anything else).

- `vertices` - starts with a column of node IDs. Any following columns are interpreted as node attributes.

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

```
## IGRAPH 8a82ae9 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 8a82ae9 (vertex names):
##  [1] s01->s02 s01->s03 s01->s04 s01->s15 s02->s01 s02->s03 s02->s09
##  [8] s02->s10 s03->s01 s03->s04 s03->s05 s03->s08 s03->s10 s03->s11
## [15] s03->s12 s04->s03 s04->s06 s04->s11 s04->s12 s04->s17 s05->s01
## [22] s05->s02 s05->s09 s05->s15 s06->s06 s06->s16 s06->s17 s07->s03
## [29] s07->s08 s07->s10 s07->s14 s08->s03 s08->s07 s08->s09 s09->s10
## [36] s10->s03 s12->s06 s12->s13 s12->s14 s13->s12 s13->s17 s14->s11
## [43] s14->s13 s15->s01 s15->s04 s15->s06 s16->s06 s16->s17 s17->s04
```

The first line in the description above has four letters:

1. D or U, for a directed or undirected graph

2. N for a named graph (where nodes have a name attribute)

3. W for a weighted graph (where edges have a weight attribute)

4. B for a bipartite (two-mode) graph (where nodes have a type attribute)

The two numbers that follow (17 49) refer to the number of nodes and edges in the graph. The description also lists node & edge attributes, for example:

- (g/c) - graph-level character attribute

- (v/c) - vertex-level character attribute

- (e/n) - edge-level numeric attribute

You can also access the nodes, edges, and their attributes with the following functions:

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

# Find nodes and edges by attribute:
# (that returns oblects of type vertex sequence/edge sequence)
V(net)[media == "BBC"]
E(net)[type == "mention"]

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

```r
net[5,7]
```

You can easily extract an edge list or matrix from the igraph network like this:

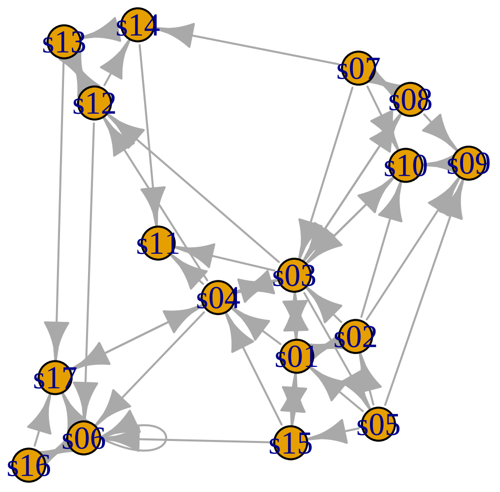
```r
# Get an edge list or a matrix:
as_edgelist(net, names = T)
as_adjacency_matrix(net, attr = "weight")
```

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

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

Now we have the igraph object, lets see what the plot looks like

```r
plot(net)
```



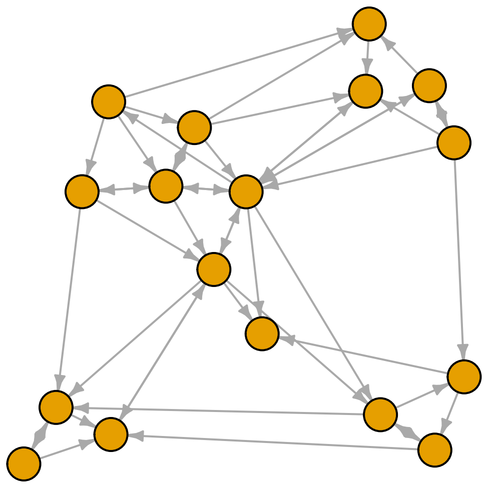
This graph violates a few of the visual communication design principles we've learnt about in this unit. Lets improve it!

1. Remove the loops.

```r
net <- igraph::simplify(net, remove.multiple = F, remove.loops = T)
```

2. Also reduce the arrow size and remove the labels by setting them to NA:

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



Note: We could also combine the edges by summing their weights with a command like `simplify(net, edge.attr.comb=list(Weight="sum","ignore"))`. However, in our data this would combine multiple edge types (i.e "hyperlinks" and "mentions").

## Dataset 2: Links between souces and consumers - Matrix format

Our second dataset is a network of links between news outlets and media consumers. Save `Dataset2-Media-Example-NODES.csv` and `Dataset2-Media-Example-EDGES.csv`.

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

examine these datasets using `head()` or `glimpse`

### bipartite networks in graphs

We can see that `links2` is an adjacency matrix for a two-mode network. Two-mode or bipartite graphs have two different types of actors and links between them, but not within each type. This data is a bipartite network where we can examine links between news sources and their consumers.

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

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

```
dim(nodes2)
```

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

Let's convert this second dataset into an igraph object. The edges of our second network are in a matrix format. We can read these into a graph object using `graph_from_incidence_matrix()`. An igraph, bipartite network has a node attribute called `type` that is `FALSE` (or 0) for vertices in one mode and `TRUE` (or 1) for those in the other mode.

```
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
## s01   1   1   1   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
## s03   0   0   0   0   0   1   1   1   1   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
## s05   0   0   0   0   0   0   0   0   0   0   1   1   1   0   0   0   0
## s06   0   0   0   0   0   0   0   0   0   0   0   0   1   1   0   0   1
##     U18 U19 U20
## s01   0   0   0
## s02   0   0   1
## s03   0   0   0
## s04   0   0   0
## s05   0   0   0
## s06   0   0   0
```

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

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

NOTE: If your data is a one-mode network matrix, it can be transformed into an `igraph` object, using `graph_from_adjacency_matrix()` instead of `graph_from_incidence_matrix()`

### Plotting parameters

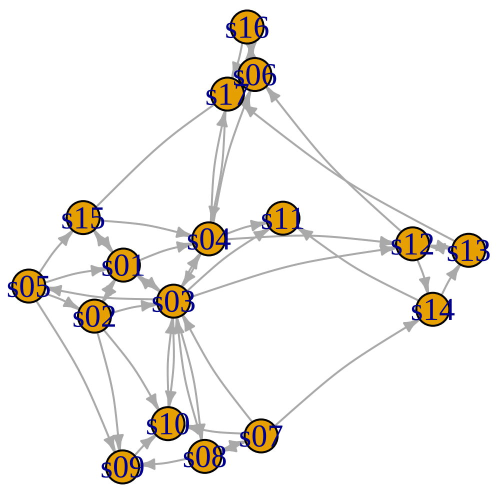
These network plots have a wide range of parameters you can adjust. They include node options or parameters (such as `vertex`) and edge options (such as `edge`). A list of options can be viewed by using `?igraph.plotting`.

```
#TODO: insert table of parameters.
```

We can set the node & edge options in two ways - the first one is to specify them in the `plot()` function, the second is to add parameters to the `igraph()` function.

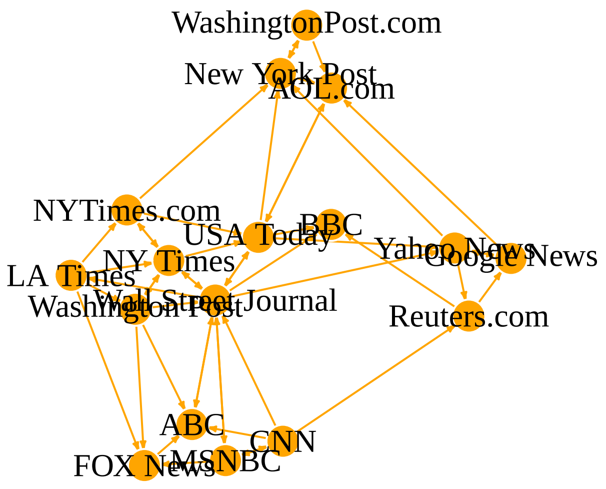
#### using `plot()`

```
# Use curved edges (edge.curved=.1) and reduce
#arrow size (edge.arrow.size= .4):
# Note that using curved edges will allow
# you to see multiple links between two nodes
# (e.g. links going in either direction, or multiplex links)
plot(net, edge.arrow.size = .4, edge.curved = .1)
```



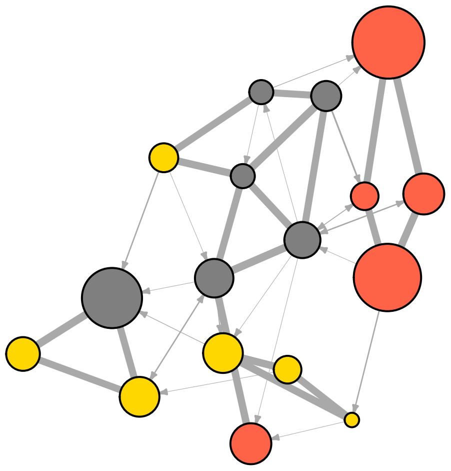
Let's modify some of the design features to improve the graph.

```
# Set edge color to light gray, the node & border color to orange.
# Replace the vertex label with the node names stored in "media".
plot(net, edge.arrow.size = .2, edge.color = "orange", vertex.color = "orange",
    vertex.frame.color = "#ffffff", vertex.label = V(net)$media,
    vertex.label.color = "black")
```



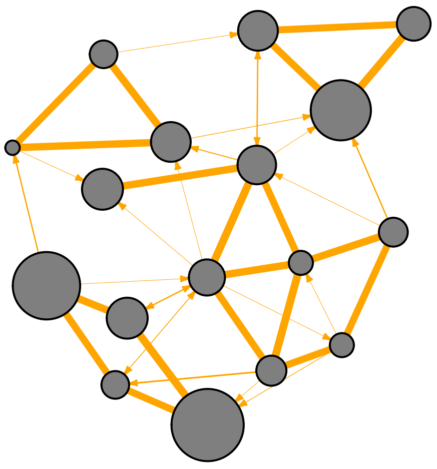
**Using `igraph()`** Let's say we want to color our network nodes based on type of media, and size them based on degree centrality (more links -> larger node) We will also change the width of the edges based on their weight.

```
# Generate colors based on media type:
colrs <- c("gray50", "tomato", "gold")
V(net)$color <- colrs[V(net)$media.type]

# Compute node degrees (#links) and use that to set node size:
deg <- igraph::degree(net, mode="total")
V(net)$size <- deg*3

# We could also use the audience size value:
V(net)$size <- V(net)$audience.size*0.6

# The labels are currently node IDs.
# Setting them to NA will render no labels:
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"

# We can even set the network layout:
graph_attr(net, "layout") <- layout_with_lgl
plot(net)
```



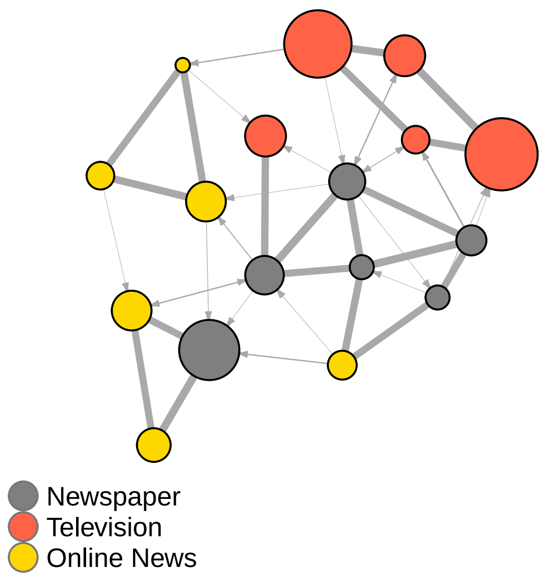
We can also override the attributes explicitly in the `plot()` after they have been set in ‘igraph():

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



It helps to add a legend explaining the meaning of the colors we used:

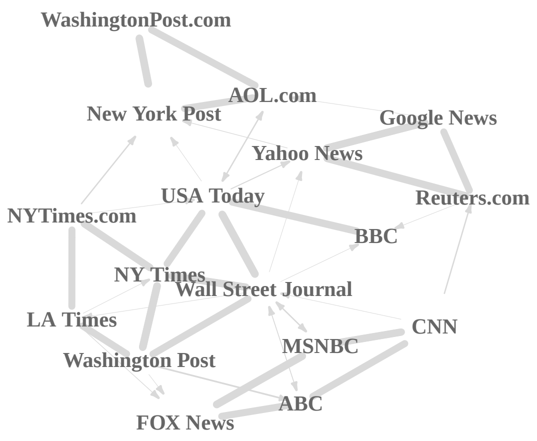
```
plot(net)
legend(x = -1.5, y = -1.1, c( "Newspaper", "Television", "Online News"), pch = 21, col = "#777777", pt.
```



NOTE: the legend function will not work in the console, but will when you render the markdown document.

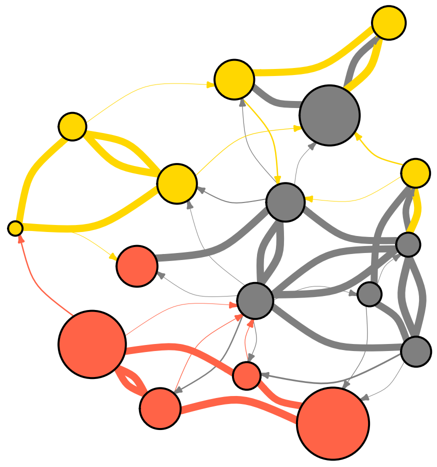
Sometimes, especially with semantic networks, we may be interested in plotting only the labels of the nodes:

```
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 can get the starting node for each edge with the ends() igraph function. It returns the start and end vertex for edges listed in the es parameter. The names parameter control whether the function returns edge names or IDs.

```
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=.4)
```



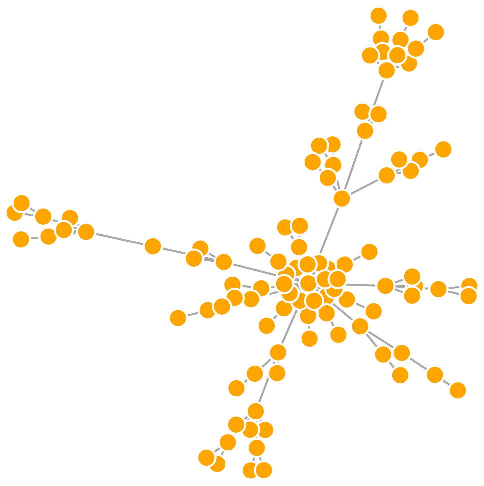
NOTE: this graph renders differently in the console compared to in a rendered markdown file.

### Network layouts

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

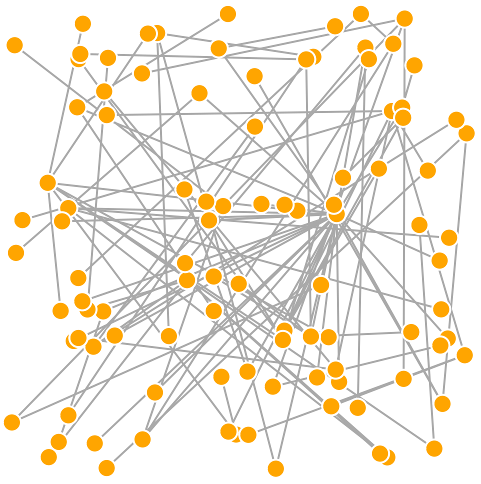
For the purposes of exploring layouts, we will generate a slightly larger 100-node graph. We use the sample_pa() function which generates a simple graph starting from one node and adding more nodes and links based on a preset level of preferential attachment (Barabasi-Albert model).

```r
net.bg <- sample_pa(100)
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)
```



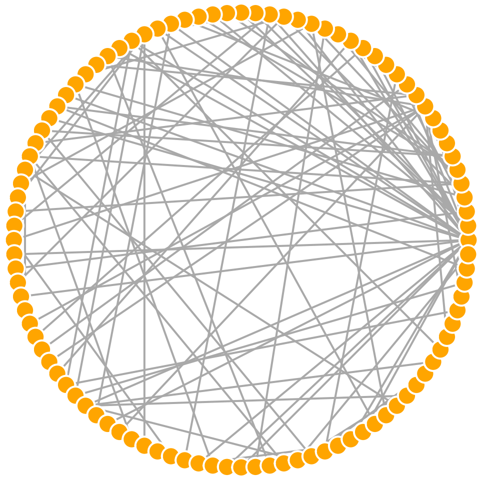
You can set the layout in the plot function:

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



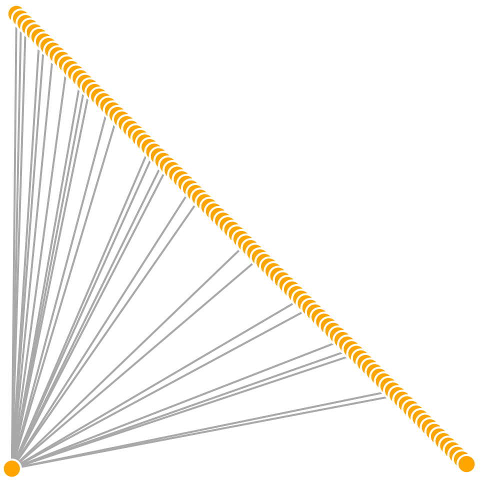
Or you can calculate the vertex coordinates in advance:

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



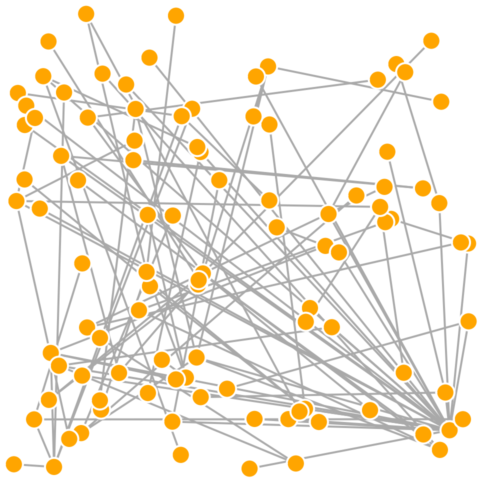
`l` is simply a matrix of x, y coordinates (N x 2) for the N nodes in the graph. You can easily generate your own:

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

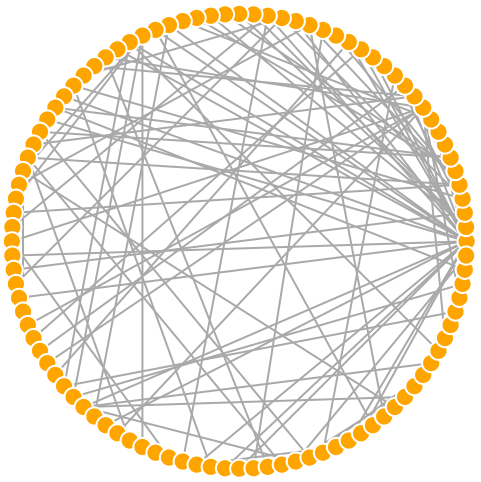


This layout is just an example and not very helpful - thankfully igraph has a number of built-in layouts, including:

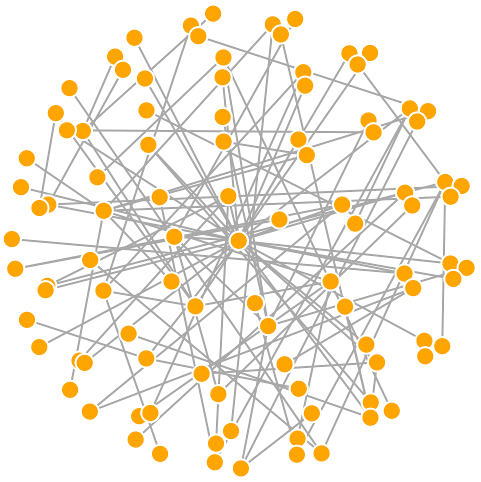
```r
# Randomly placed vertices
l <- layout_randomly(net.bg)
plot(net.bg, layout=l)
```



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



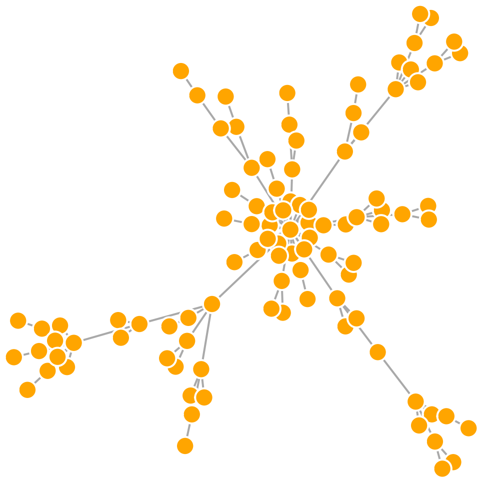
```r
# 3D sphere layout
l <- layout_on_sphere(net.bg)
plot(net.bg, layout=l)
```



Fruchterman-Reingold is one of the most used force-directed layout algorithms out there.

Force-directed layouts try to get a nice-looking graph where edges are similar in length and cross each other as little as possible. They simulate the graph as a physical system. Nodes are electrically charged particles that repulse each other when they get too close. The edges act as springs that attract connected nodes closer together. As a result, nodes are evenly distributed through the chart area, and the layout is intuitive in that nodes which share more connections are closer to each other. The disadvantage of these algorithms is that they are rather slow and therefore less often used in graphs larger than ~1000 vertices. You can set the "weight" parameter which increases the attraction forces among nodes connected by heavier edges.

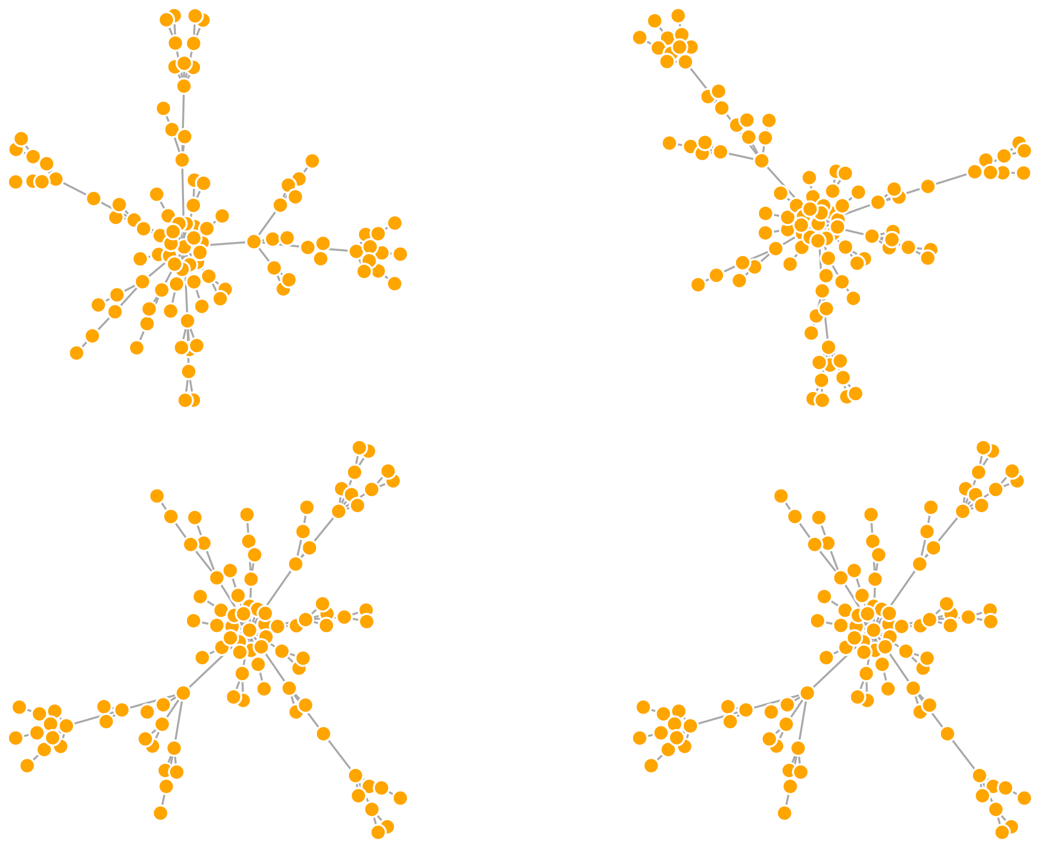
```r
l <- layout_with_fr(net.bg)
plot(net.bg, layout=l)
```



You will notice that this 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, which can be helpful if you want to plot the time evolution of a graph, or different relationships – and want nodes to stay in the same place in multiple plots.

TODO: add in description of the par() function.

```r
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)
```

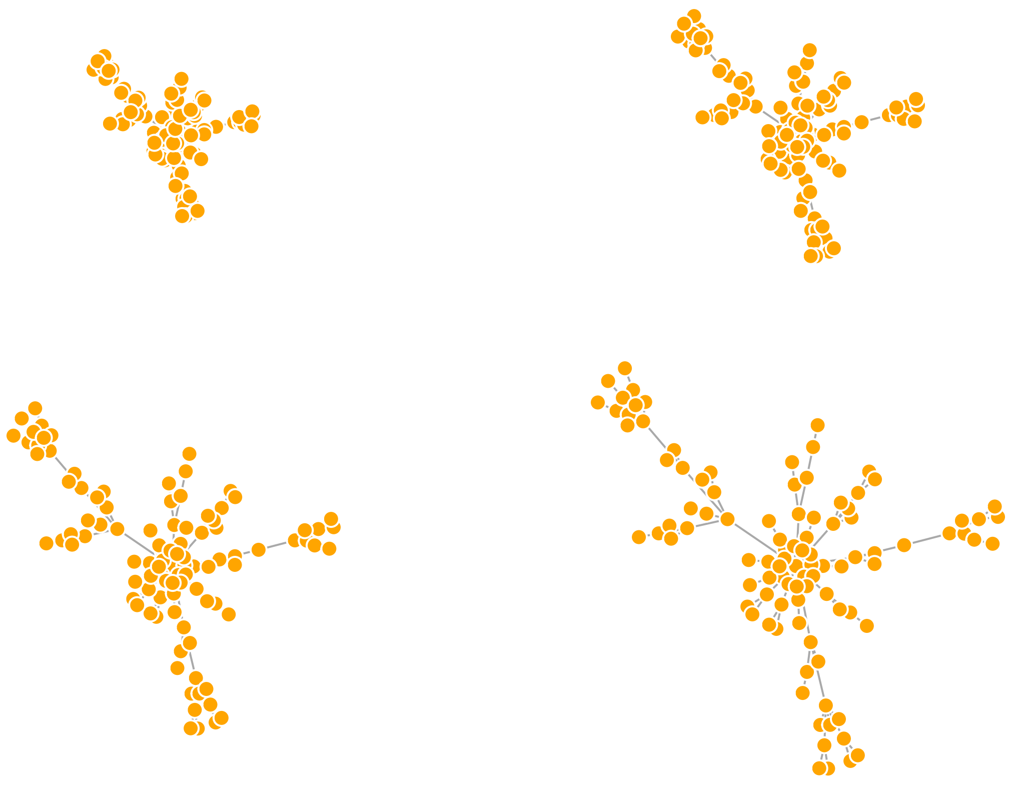


```
dev.off()
```

```
## null device
##           1
```

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. This way you can create more compact or spread out layout versions.

```
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)
```

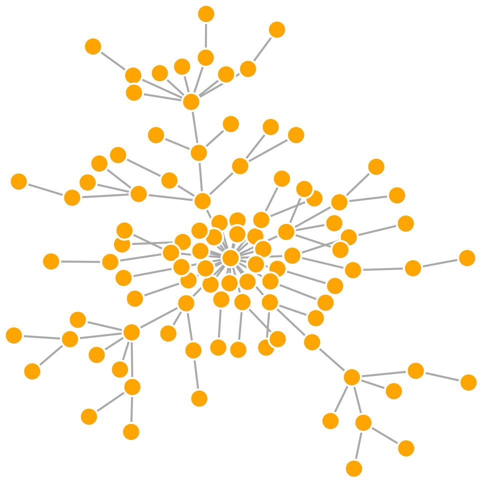


```
dev.off()
```

```
## null device
##           1
```

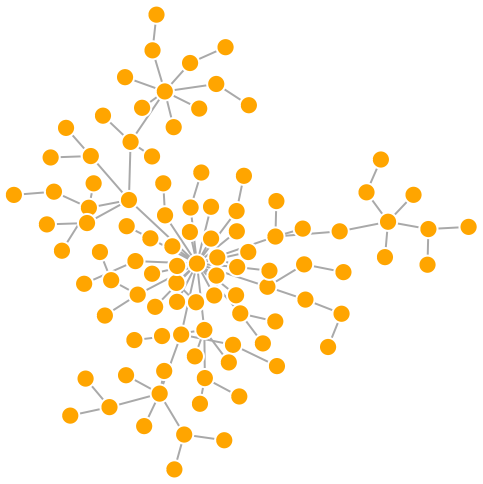
Another popular force-directed algorithm that produces nice results for connected graphs is Kamada Kawai. Like Fruchterman Reingold, it attempts to minimize the energy in a spring system.

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



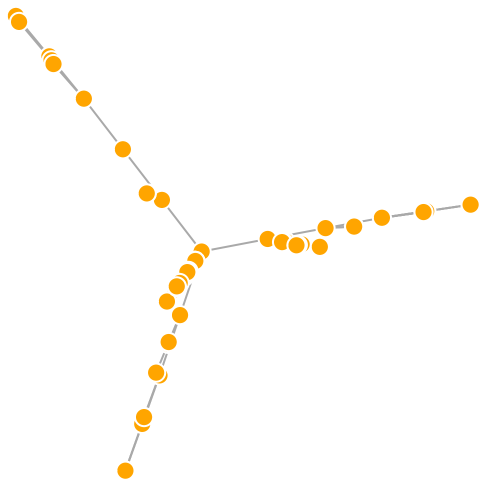
The LGL algorithm is meant for large, connected graphs. Here you can also specify a root: a node that will be placed in the middle of the layout.

```r
plot(net.bg, layout=layout_with_lgl)
```



The MDS (multidimensional scaling) algorithm tries to place nodes based on some measure of similarity or distance between them. More similar nodes are plotted closer to each other. By default, the measure used is based on the shortest paths between nodes in the network. We can change that by using our own distance matrix (however defined) with the parameter dist. MDS layouts are nice because positions and distances have a clear interpretation. The problem with them is visual clarity: nodes often overlap, or are placed on top of each other.

```r
plot(net.bg, layout=layout_with_mds)
```



Let's take a look at all available layouts in igraph:

```
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"
```

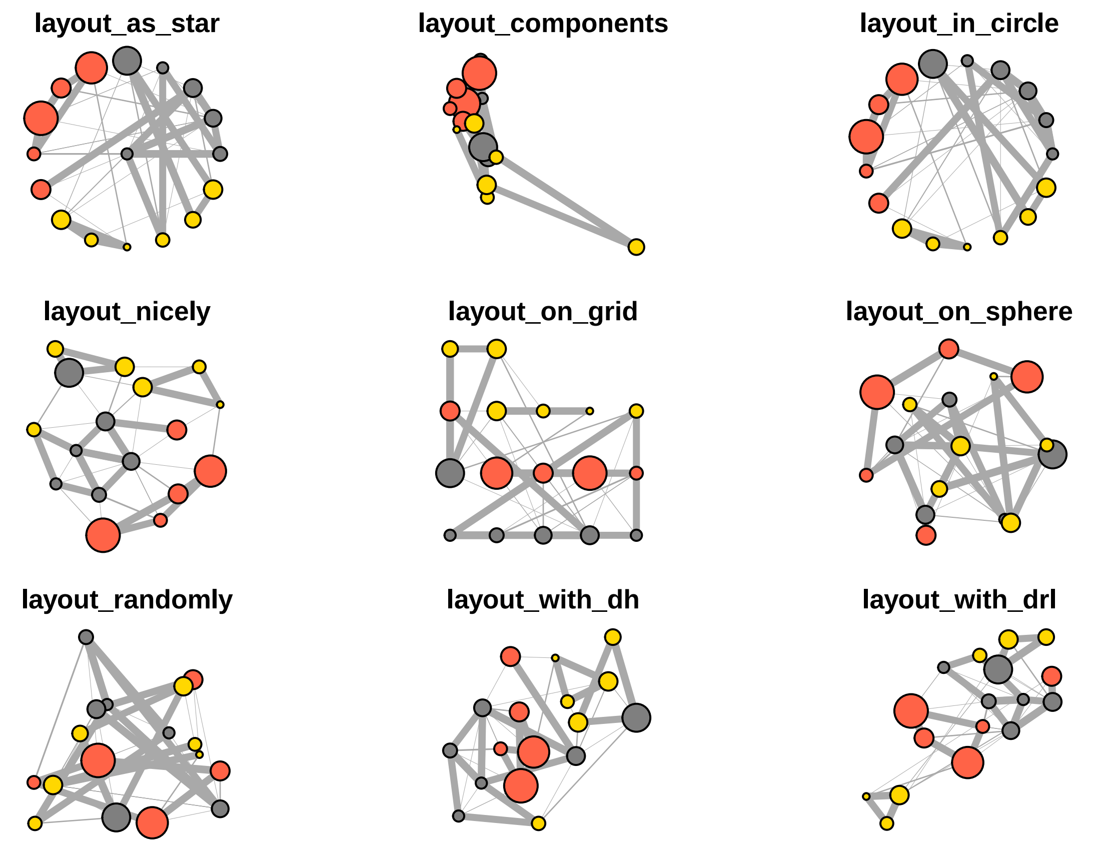
```
## [1] "layout_on_grid"
```

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

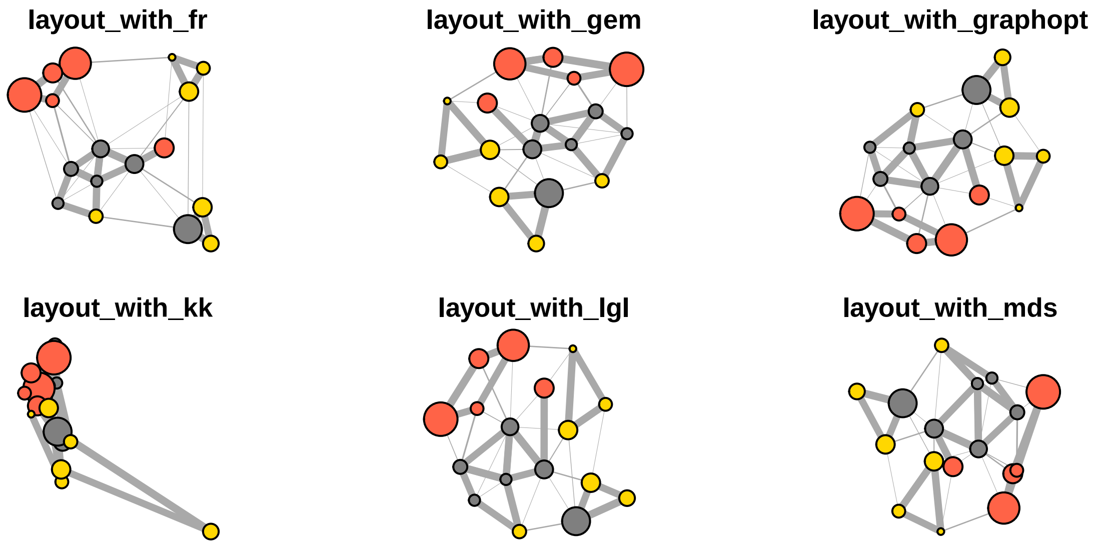
```
## [1] "layout_randomly"
```

```
## [1] "layout_with_dh"
```

```
## [1] "layout_with_drl"
```



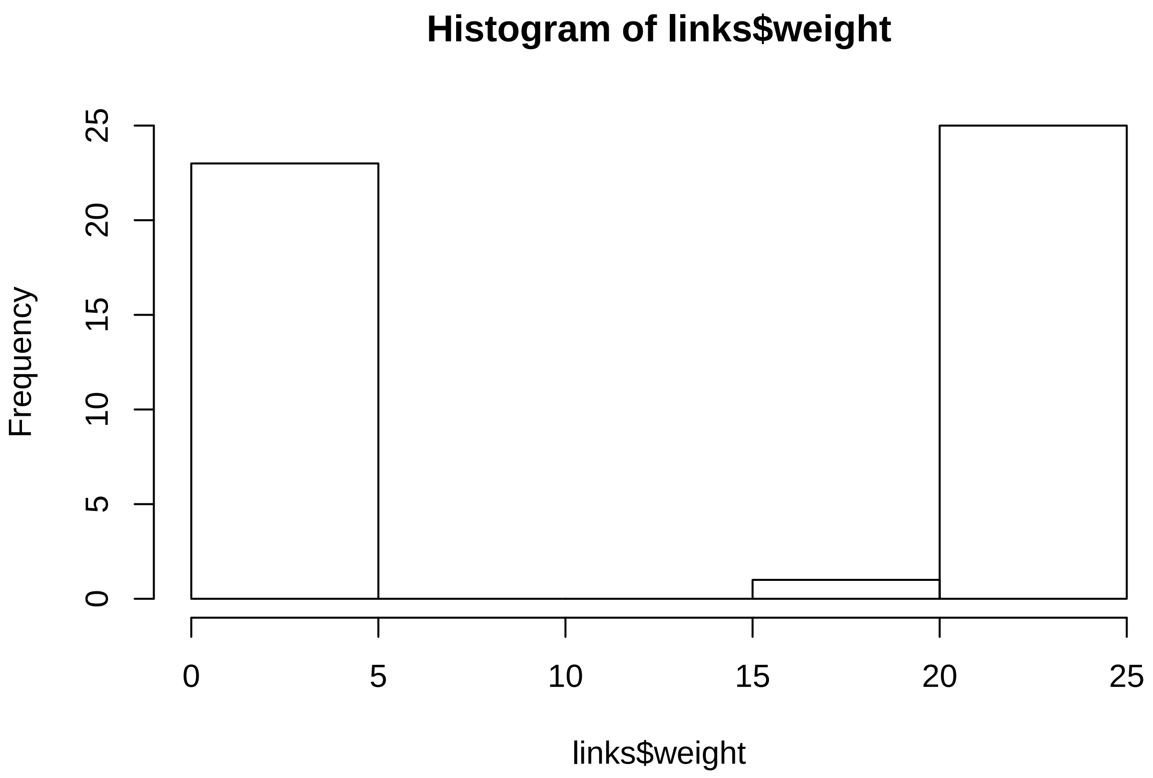
```
## [1] "layout_with_fr"
## [1] "layout_with_gem"
## [1] "layout_with_graphopt"
## [1] "layout_with_kk"
## [1] "layout_with_lgl"
## [1] "layout_with_mds"
```



### Highlight aspects of the network

Notice that our 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, keeping only the most important ties and discarding the rest.

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



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

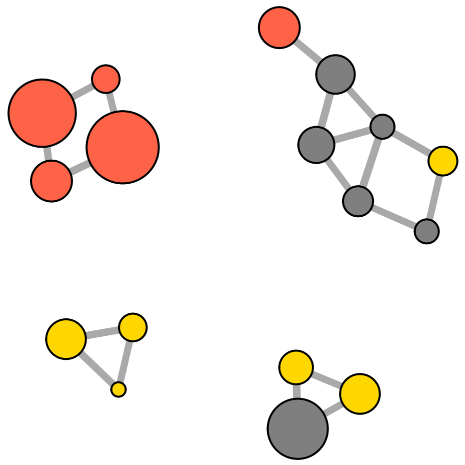
```
## [1] 12.40816
```

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

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

There are more sophisticated ways to extract the key edges, but for the purposes of this exercise we'll only keep ones that have weight higher than the mean for the network. In igraph, 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, layout=layout_with_kk)
```



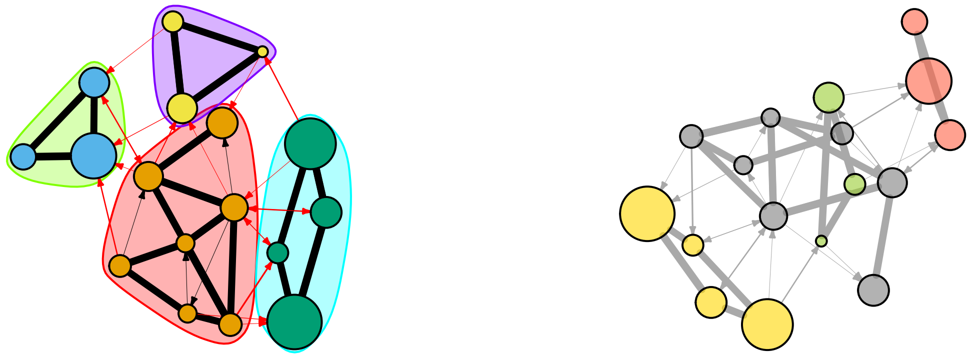
nother way to think about this is to plot the two tie types (hyperlink & mention) separately. We will do that in section 5 of this tutorial: Plotting multiplex networks.

We can also try to make the network map more useful by showing the communities within it:

```r
par(mfrow=c(1,2))

# Community detection (by optimizing modularity over partitions):
clp <- cluster_optimal(net)
class(clp)
```

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

```r
# Community detection returns an object of class "communities"
# which igraph knows how to plot:
plot(clp, net)

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



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

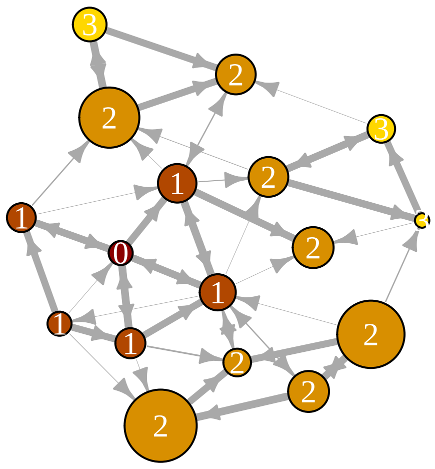
```
## null device
##           1
```

### Highlight specific nodes or links

Sometimes we want to focus the visualization on a particular node or a group of nodes. In our example media network, we can examine the spread of information from focal actors. For instance, let's represent distance from the NYT.

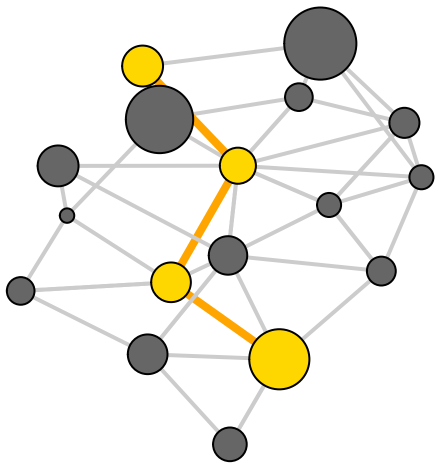
The `distances` function returns a matrix of shortest paths from nodes listed in the v parameter to ones included in the to parameter.

```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")
```



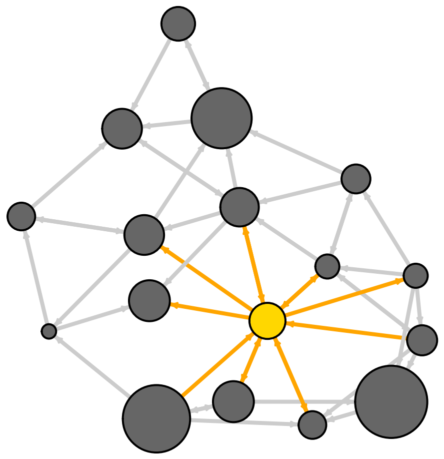
We can also highlight a path in the network:

```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$epath)] <- "orange"
# Generate edge width variable to plot the path:
ew <- rep(2, ecount(net))
ew[unlist(news.path$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)
```



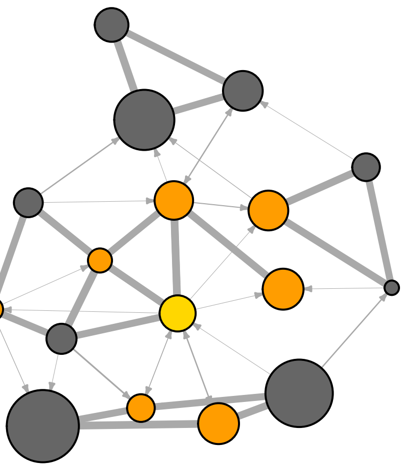
We can highlight the edges going into or out of a vertex, for instance the WSJ. For a single node, use `incident()`, for multiple nodes use `incident_edges()`

```
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, edge.width=2)
```



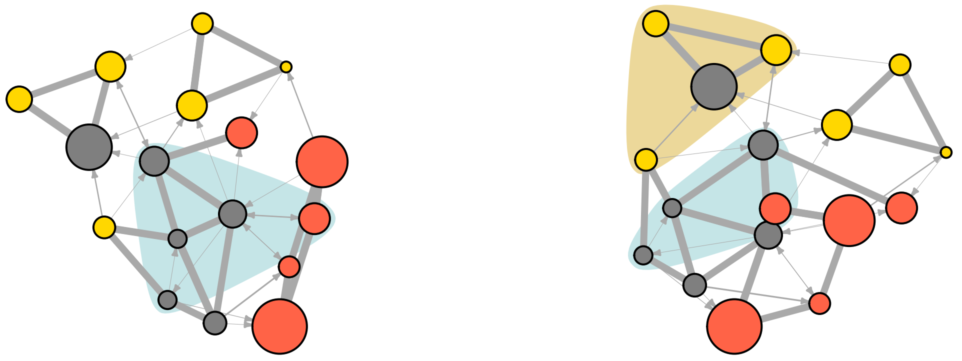
We can also point to the immediate neighbors of a vertex, say WSJ. The neighbors function finds all nodes one step out from the focal actor.To find the neighbors for multiple nodes, use adjacent_vertices() instead of neighbors(). To find node neighborhoods going more than one step out, use function ego() 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)
```



A way to draw attention to a group of nodes (we saw this before with communities) is to “mark” them. Note that this is generally not recommended since, depending on layout, nodes that have not been marked can accidentally get placed on top of the colored area.

```
par(mfrow=c(1,2))
plot(net, mark.groups=c(1,4,5,8), mark.col="#C5E5E7", mark.border=NA)

# Mark multiple groups:
plot(net, mark.groups=list(c(1,4,5,8), c(15:17)),
          mark.col=c("#C5E5E7","#ECD89A"), mark.border=NA)
```



```
dev.off()
```

```
## null device
##           1
```

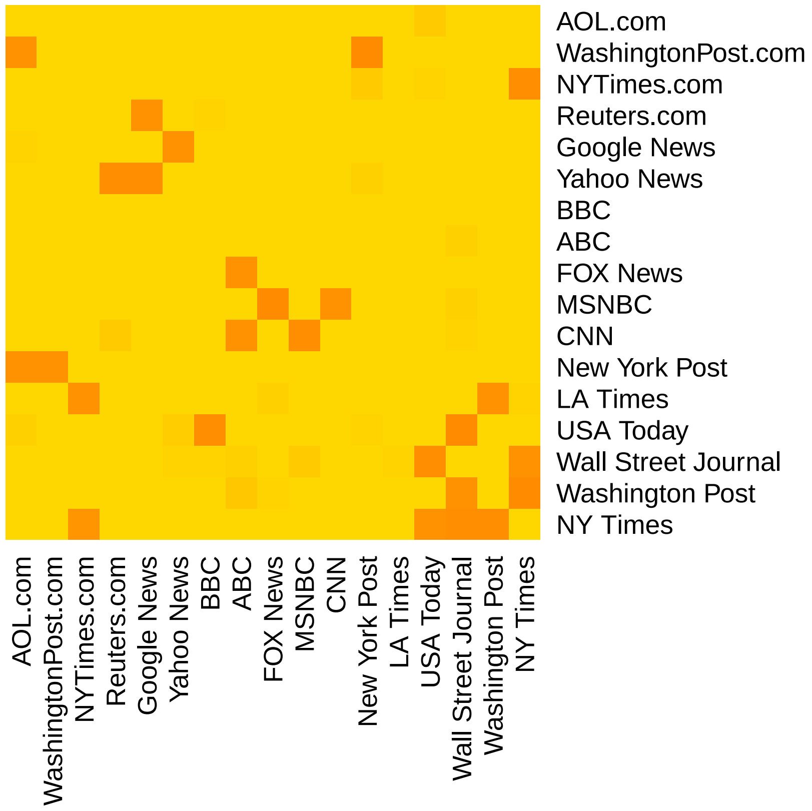
### Other ways to represent a network

At this point it might be useful to provide a quick reminder that there are many ways to represent a network not limited to a hairball plot.

For example, here is a quick heatmap of the network matrix:

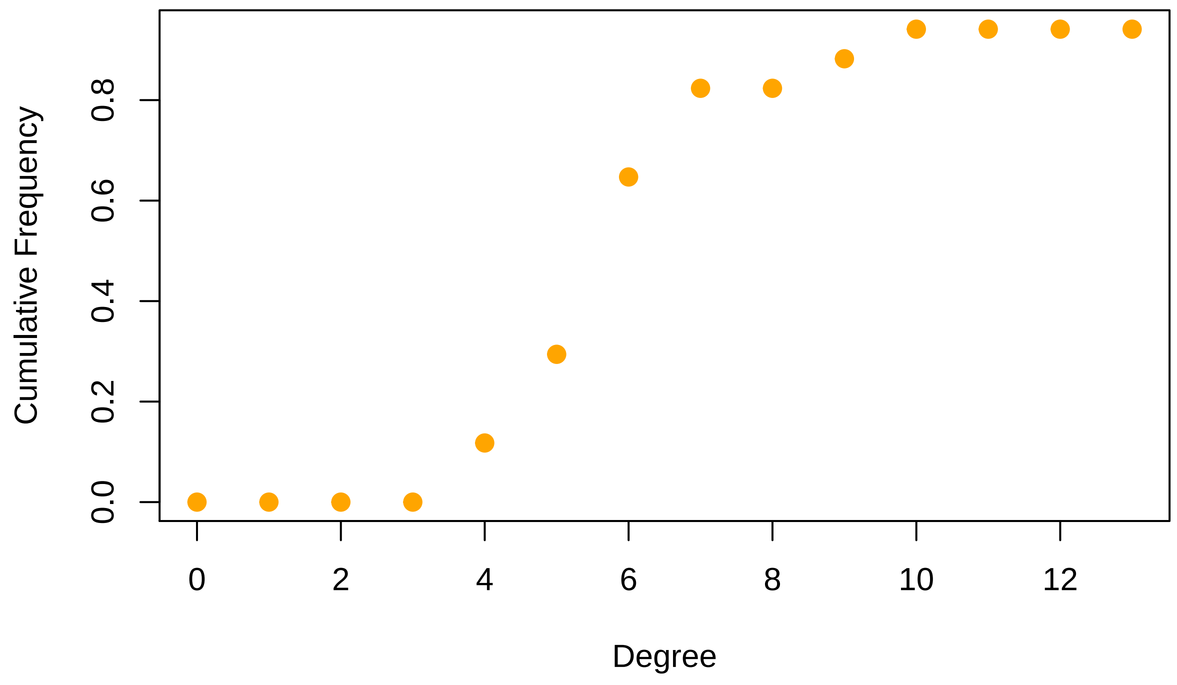
```
netm <- as_adjacency_matrix(net, attr="weight", sparse=F)
colnames(netm) <- V(net)$media
rownames(netm) <- V(net)$media

palf <- colorRampPalette(c("gold", "dark orange"))
```

```
heatmap(netm[,17:1], Rowv = NA, Colv = NA, col = palf(100),
        scale="none", margins=c(10,10) )
```



Depending on what properties of the network or its nodes and edges are most important to you, simple graphs can often be more informative than network maps.

```
# Plot the degree distribution for our network:
deg.dist <- igraph::degree_distribution(net, cumulative=T, mode="total")
plot( x=0:max(igraph::degree(net)), y=1-deg.dist, pch=19, cex=1.2, col="orange",
  xlab="Degree", ylab="Cumulative Frequency")
```



### Plotting two mode networks

As you might remember, our second media example is a two-mode network examining links between news sources and their consumers.

```
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
```

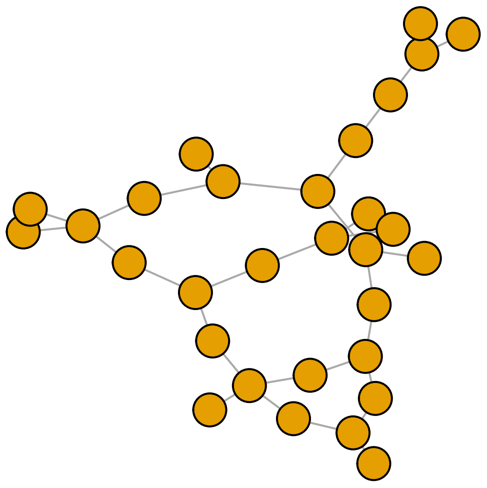
```
head(links2)
```

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

```r
net2
```

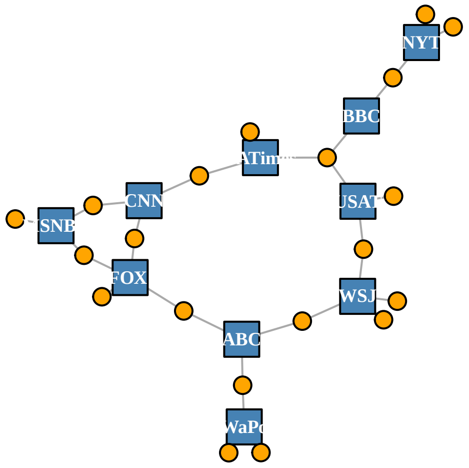
```
## IGRAPH 3afede6 UN-B 30 31 --
## + attr: type (v/l), name (v/c)
## + edges from 3afede6 (vertex names):
##  [1] s01--U01 s01--U02 s01--U03 s02--U04 s02--U05 s02--U20 s03--U06
##  [8] s03--U07 s03--U08 s03--U09 s04--U09 s04--U10 s04--U11 s05--U11
## [15] s05--U12 s05--U13 s06--U13 s06--U14 s06--U17 s07--U14 s07--U15
## [22] s07--U16 s08--U16 s08--U17 s08--U18 s08--U19 s09--U06 s09--U19
## [29] s09--U20 s10--U01 s10--U11
```

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



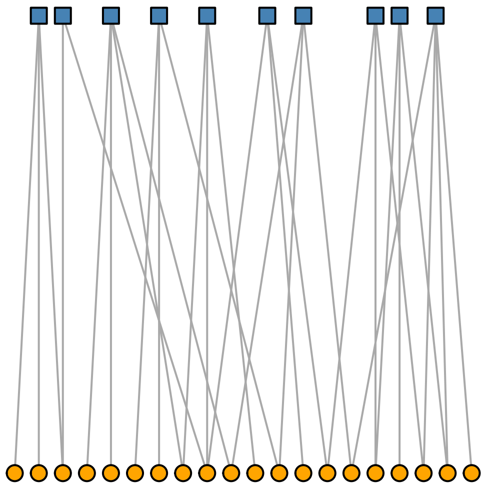
As with one-mode networks, we can modify the network object to include the visual properties that will be used by default when plotting the network. Notice that this time we will also change the shape of the nodes - media outlets will be squares, and their users will be circles.

```r
# Media outlets are blue squares, audience nodes are orange circles:
V(net2)$color <- c("steel blue", "orange")[V(net2)$type+1]
V(net2)$shape <- c("square", "circle")[V(net2)$type+1]

# Media outlets will have name labels, audience members will not:
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)
```



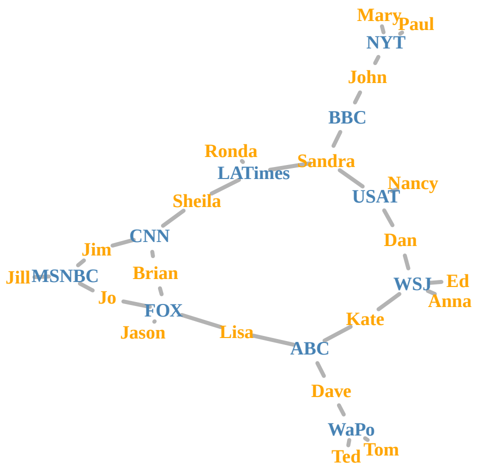
In `igraph`, there is also a special layout for bipartite networks (though it doesn't always work great, and you might be better off generating your own two-mode layout).

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



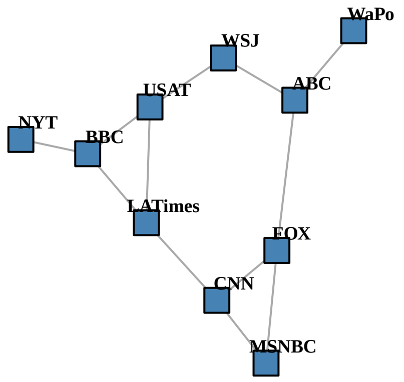
Using text as nodes may be helpful at times:

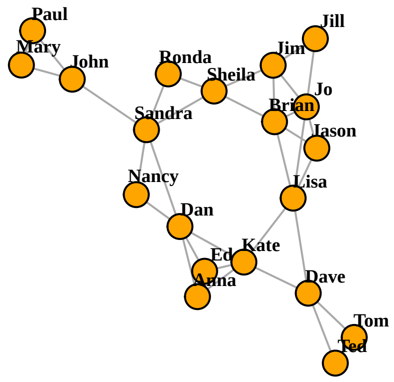
```r
plot(net2, vertex.shape="none", vertex.label=nodes2$media,
     vertex.label.color=V(net2)$color, vertex.label.font=2,
     vertex.label.cex=.6, edge.color="gray70",  edge.width=2)
```



We can also generate and plot 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.

```
par(mfrow=c(1,2))

net2.bp <- bipartite.projection(net2)

plot(net2.bp$proj1, vertex.label.color="black", vertex.label.dist=2,
     vertex.label=nodes2$media[!is.na(nodes2$media.type)])

plot(net2.bp$proj2, vertex.label.color="black", vertex.label.dist=2,
     vertex.label=nodes2$media[ is.na(nodes2$media.type)])
```



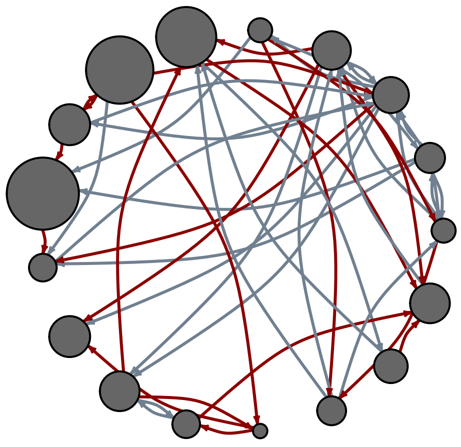


```
dev.off()
```

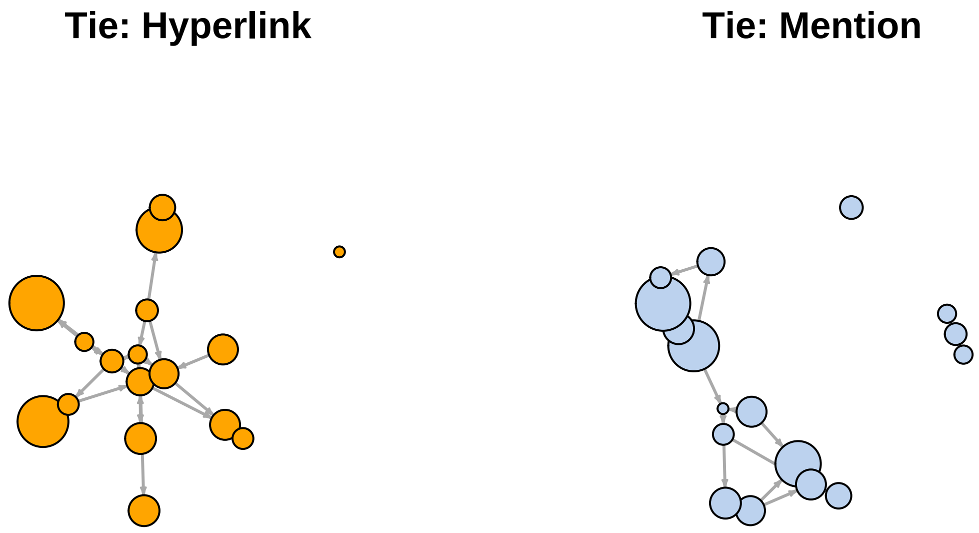
```
## null device
##           1
```

### Plotting multiple networks

```
E(net)$width <- 1.5
plot(net, edge.color=c("dark red", "slategrey")[(E(net)$type=="hyperlink")+1],
      vertex.color="gray40", layout=layout_in_circle, edge.curved=.3)
```

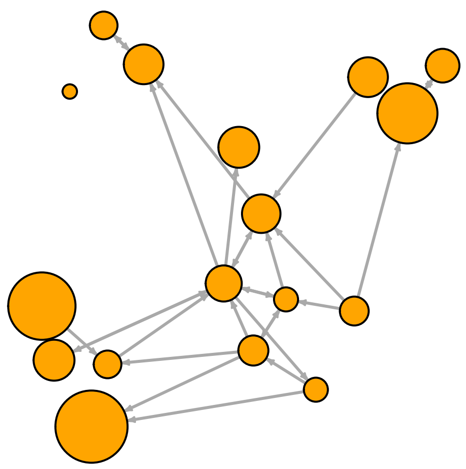


```
net.m <- net - E(net)[E(net)$type=="hyperlink"] # another way to delete edges:
net.h <- net - E(net)[E(net)$type=="mention"]   # using the minus operator

# Plot the two links separately:
par(mfrow=c(1,2))
plot(net.h, vertex.color="orange", layout=layout_with_fr, main="Tie: Hyperlink")
plot(net.m, vertex.color="lightsteelblue2", layout=layout_with_fr, main="Tie: Mention")
```



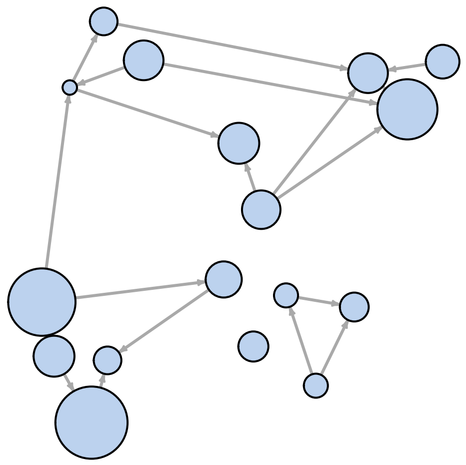
```
# Make sure the nodes stay in place in both plots:
l <- layout_with_fr(net)
plot(net.h, vertex.color="orange", layout=l, main="Tie: Hyperlink")
```

Tie: Hyperlink



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

Tie: Mention



```
dev.off()
```

```
## null device
##           1
```

In our example network, it so happens that we do not have node dyads connected by multiple types of connections. That is to say, we never have both a 'hyperlink' and a 'mention' tie between the same two news outlets. However, this could easily happen in a multiplex network.

## Network Example 2.

The following section outlines an dadditional example of `igraph` but using a different dataset. This dataset contains cattle movement data.

Download and save the R.data files into the project directory. Then load both files using the code below.

```
attr1 <- read.csv("data/attr_farms.csv", stringsAsFactors = F) #load the edgelist_farms.Rdata
edges <- read.csv("data/Edgelist_farms.csv", stringsAsFactors = F) #load attr.farms.Rdata
```

### Create an igraph object.

Inspect the `edges` dataset

```
head(edges)
```

```
##   Origin Dest       Date breeding.cows steers heifers calves batch.size
## 1      1   25 2008-12-10             0     43       0      0         43
## 2      1   25 2008-12-10             0     41       0      0         41
## 3      3   88 2009-07-30             0      0       0     24         24
## 4      2   25 2008-12-10             0     42       0      0         42
## 5      5   13 2008-12-20             0      0      12      0         12
## 6      5    7 2008-12-20             0      0       8      0          8
```

Create an igraph object using the `edges` data that you downloaded.

```
net_edges <- graph.data.frame(edges,directed=T)
net_edges
```

```
## IGRAPH 4f3b91b DN-- 120 356 --
## + attr: name (v/c), Date (e/c), breeding.cows (e/n), steers (e/n),
## | heifers (e/n), calves (e/n), batch.size (e/n)
## + edges from 4f3b91b (vertex names):
##  [1] 1 ->25  1 ->25  3 ->88  2 ->25  5 ->13  5 ->7   15->18  15->18
##  [9] 15->18  15->19  15->19  15->7   15->7   15->88  15->88  15->106
## [17] 15->26  13->4   13->14  13->14  13->17  13->20  13->8   13->10
## [25] 13->7   13->88  13->93  23->22  23->21  18->19  18->7   18->7
## [33] 18->88  18->88  18->106 18->106 9 ->13  9 ->10  9 ->10  9 ->10
## [41] 9 ->10  9 ->10  9 ->10  9 ->10  9 ->10  9 ->10  9 ->10  9 ->10
## [49] 4 ->29  4 ->29  4 ->29  11->13  11->13  35->116 12->13  12->13
## + ... omitted several edges
```

The next key thing to know how to do in `igraph` is how add node-level attributes from another file. We didn't do this in the last example. The attr1 file is a table of attributes

```
head(attr1)
```

```
##   X farm.id      type size    long      lat state
## 1 1       1 Fattening  354 6194009 567599.8     0
## 2 2       3     Dairy  203 6181087 559118.7     0
## 3 3       2 Fattening  250 6196090 562336.4     0
## 4 4       5     Dairy  994 6207398 474646.4     0
## 5 5      15     Dairy  544 6221094 413707.2     0
## 6 6      13     Dairy   87 6205884 421178.5     0
```

next we will add `production` to each node.

The V(net) notation indicates a vertex attribute of the specified network. V(net)$name indicates what we would like to name the attribute. The "V" stands for vertex, which is another name for node. The match command ensures that the production type added to each node is the correct one for that node by matching the name of the node to the corresponding farm.id in the attribute file.

```r
V(net_edges)$type <- as.character(attr1$type[match(V(net_edges)$name,attr1$farm.id)])
net_edges
```

```
## IGRAPH 4f3b91b DN-B 120 356 --
## + attr: name (v/c), type (v/c), Date (e/c), breeding.cows (e/n),
## | steers (e/n), heifers (e/n), calves (e/n), batch.size (e/n)
## + edges from 4f3b91b (vertex names):
##  [1] 1 ->25  1 ->25  3 ->88  2 ->25  5 ->13  5 ->7   15->18  15->18
##  [9] 15->18  15->19  15->19  15->7   15->7   15->88  15->88  15->106
## [17] 15->26  13->4   13->14  13->14  13->17  13->20  13->8   13->10
## [25] 13->7   13->88  13->93  23->22  23->21  18->19  18->7   18->7
## [33] 18->88  18->88  18->106 18->106 9 ->13  9 ->10  9 ->10  9 ->10
## [41] 9 ->10  9 ->10  9 ->10  9 ->10  9 ->10  9 ->10  9 ->10  9 ->10
## [49] 4 ->29  4 ->29  4 ->29  11->13  11->13  35->116 12->13  12->13
## + ... omitted several edges
```

Note: “type” now appears as a node-level (v/c) attribute. Edge-level attributes are denoted as (e/n)

Now add herd size

```r
V(net_edges)$farm.size <- attr1$size[match(V(net_edges)$name,attr1$farm.id)]
net_edges
```

```
## IGRAPH 4f3b91b DN-B 120 356 --
## + attr: name (v/c), type (v/c), farm.size (v/n), Date (e/c),
## | breeding.cows (e/n), steers (e/n), heifers (e/n), calves (e/n),
## | batch.size (e/n)
## + edges from 4f3b91b (vertex names):
##  [1] 1 ->25  1 ->25  3 ->88  2 ->25  5 ->13  5 ->7   15->18  15->18
##  [9] 15->18  15->19  15->19  15->7   15->7   15->88  15->88  15->106
## [17] 15->26  13->4   13->14  13->14  13->17  13->20  13->8   13->10
## [25] 13->7   13->88  13->93  23->22  23->21  18->19  18->7   18->7
## [33] 18->88  18->88  18->106 18->106 9 ->13  9 ->10  9 ->10  9 ->10
## [41] 9 ->10  9 ->10  9 ->10  9 ->10  9 ->10  9 ->10  9 ->10  9 ->10
## + ... omitted several edges
```

## Plot the network

Prepare custom colors for plotting by creating an attribute for vertex (node) color

```r
V(net_edges)$color <- V(net_edges)$type
V(net_edges)$color <- gsub("Breeding","navy",V(net_edges)$color)
V(net_edges)$color <- gsub("Complete cycle","magenta2",V(net_edges)$color)
V(net_edges)$color <- gsub("Growing","green4",V(net_edges)$color)
V(net_edges)$color <- gsub("Fattening","steelblue1",V(net_edges)$color)
V(net_edges)$color <- gsub("Dairy","darkgoldenrod1",V(net_edges)$color)
V(net_edges)$color <- gsub("Small farm","mediumpurple",V(net_edges)$color)

net_edges
```

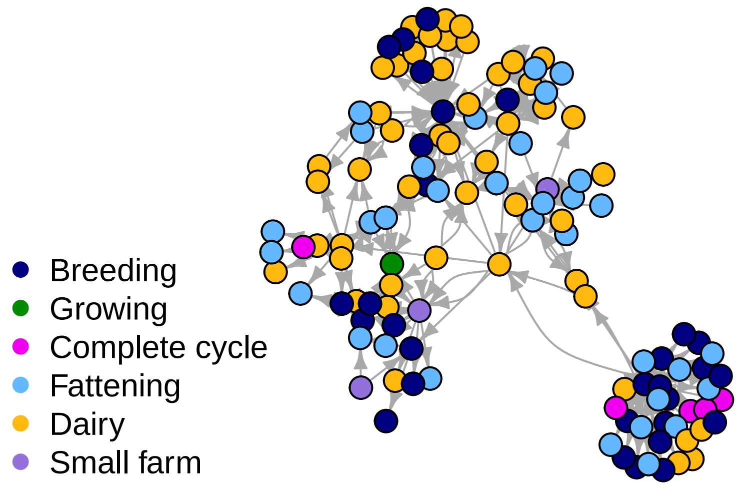
```
## IGRAPH 4f3b91b DN-B 120 356 --
## + attr: name (v/c), type (v/c), farm.size (v/n), color (v/c), Date
## | (e/c), breeding.cows (e/n), steers (e/n), heifers (e/n), calves
## | (e/n), batch.size (e/n)
## + edges from 4f3b91b (vertex names):
```

```
##  [1] 1 ->25  1 ->25  3 ->88  2 ->25  5 ->13  5 ->7   15->18  15->18
##  [9] 15->18  15->19  15->19  15->7   15->7   15->88  15->88  15->106
## [17] 15->26  13->4   13->14  13->14  13->17  13->20  13->8   13->10
## [25] 13->7   13->88  13->93  23->22  23->21  18->19  18->7   18->7
## [33] 18->88  18->88  18->106 18->106 9 ->13  9 ->10  9 ->10  9 ->10
## [41] 9 ->10  9 ->10  9 ->10  9 ->10  9 ->10  9 ->10  9 ->10  9 ->10
## + ... omitted several edges
```

```
#You could similarly specify shapes by creating a shape attribute.
```

Plot the network

```
plot.igraph(simplify(net_edges),
    layout=layout.fruchterman.reingold,
    vertex.label=NA,
    vertex.color=V(net_edges)$color,
    vertex.size=10,
    edge.arrow.size=.5)

#make a legend
legend("bottomleft",
    legend=c("Breeding","Growing","Complete cycle","Fattening","Dairy","Small farm"),
    col=c("navy","green4","magenta2","steelblue1","darkgoldenrod1","mediumpurple"),
    pch=19,cex=1,bty="n")
```



**Assignment question 1.**

Try any four different layouts, such as fruchterman.reingold, kamada.kawai, etc. These can be entered in the "layout=" as part of the plot.igraph statement. Complete this question in R Markdown. Describe in a few sentences how the different layouts convey different impressions about the structure of the network.

```
?layout.fruchterman.reingold #to see different layout options
```

On your own, try making the vertex size scale with the log of the farm size. Do you notice any patterns about where large farms are located within the network?

### Edge criteria

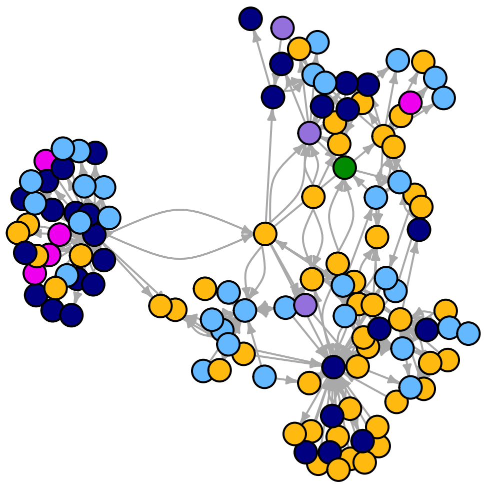
What if we want to change the way edges are determined? Right now, there is a row for every movement, but not all movements consist of the same types of animals.

```r
head(edges)
```

```
##   Origin Dest       Date breeding.cows steers heifers calves batch.size
## 1      1   25 2008-12-10             0     43       0      0         43
## 2      1   25 2008-12-10             0     41       0      0         41
## 3      3   88 2009-07-30             0      0       0     24         24
## 4      2   25 2008-12-10             0     42       0      0         42
## 5      5   13 2008-12-20             0      0      12      0         12
## 6      5    7 2008-12-20             0      0       8      0          8
```

Let's recreate our network so only movements of >50 animals are considered

```r
#Filter edges
e2 <- edges %>%
  filter('batch.size' > 50 )

net.c <- graph.data.frame(e2)

#re-add attributes and change colors
V(net.c)$farm.size <- (attr1$size[match(V(net.c)$name,attr1$farm.id)] )
V(net.c)$type <- as.character(attr1$type[match(V(net.c)$name,attr1$farm.id)] )
V(net.c)$color <- V(net.c)$type
V(net.c)$color <- gsub("Breeding","navy",V(net.c)$color)
V(net.c)$color <- gsub("Complete cycle","magenta2",V(net.c)$color)
V(net.c)$color <- gsub("Growing","green4",V(net.c)$color)
V(net.c)$color <- gsub("Fattening","steelblue1",V(net.c)$color)
V(net.c)$color <- gsub("Dairy","darkgoldenrod1",V(net.c)$color)
V(net.c)$color <- gsub("Small farm","mediumpurple",V(net.c)$color)

#You could similarly specify shapes by creating a shape attribute.
plot.igraph(simplify(net.c),
    layout=layout.fruchterman.reingold,
    vertex.label=NA,
    vertex.color=V(net.c)$color,
    vertex.size=10,
    edge.arrow.size=.3)
```



How does the network change?

Re-try this with filtering only for movements of heifers. How does the network change?

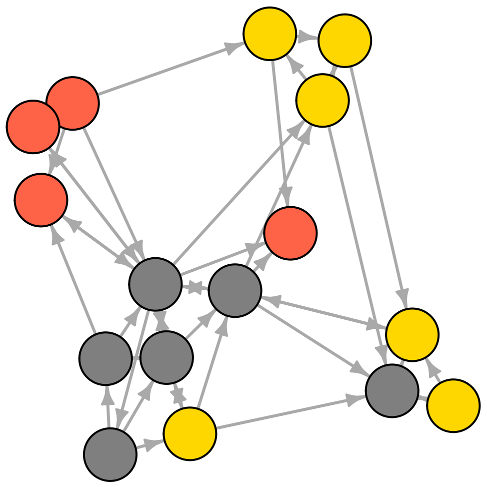
### Subgraphs

Creating a subset of the full graph can be useful for "zooming in" on areas of the network of interest. An "ego network" is a type of subgraph that is created based on all the connections of a single node that are within k steps in the network.

```
?graph.neighborhood
net3 <- graph.neighborhood(net,2,nodes=4) #find the second-order neighborhood of node 4

#Output is in list format. So extract the network from the first position in the list
net3 <- net3[[1]]
```

Plot the network

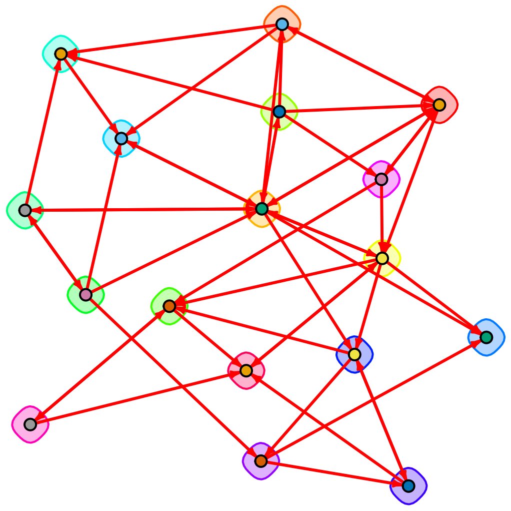
```
#highlight the focal node
V(net3)$shape <- "circle"
V(net3)$shape[V(net3)$name==4] <- "square"
plot.igraph(net3,
    vertex.color=V(net3)$color,
    vertex.label.color="white",
    vertex.shape=V(net3)$shape,
    vertex.size=25,
    vertex.shape=V(net3)$shape,
    edge.arrow.size=.5,
    layout=layout.fruchterman.reingold)
```



**Finding communities**

There are many ways to find clusters of nodes that are more interlinked with one another then with nodes outside their cluster. We will briefly explore one such method.

```r
#Create a community object.
c <- walktrap.community(net,weights=1/E(net)$batch.size,steps=7)
plot(c, net, vertex.size=5, edge.arrow.size=.25,vertex.label=NA)
```



The Modularity score listed at the top of the output is a measure ranging from 0-1 of how good the community structure (how many edges are directed within a community versus between communities). Typically, values of 0.2 - 0.7 are considered indicative of moderate to strong community structure. The Membership vector in the ouptput lists the community ID number for each node in your network.

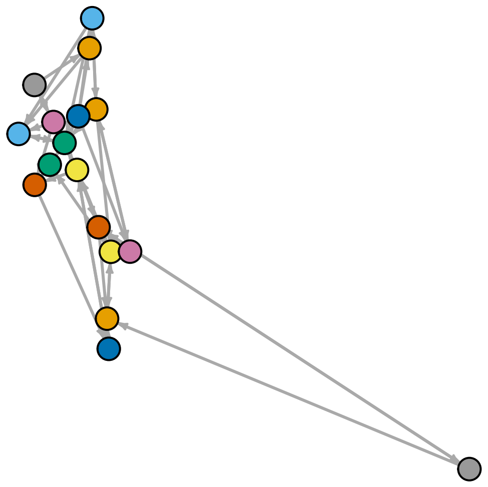
Let's re-plot this community structure in another way. Reset color palette so there are more default colors in the palette. RColorBrewer package helps select contrasting colors. You can google RColorBrewer to get a list of possible color schemes.

```r
palette(brewer.pal(length(c),"Paired"))
```

```
## Warning in brewer.pal(length(c), "Paired"): n too large, allowed maximum for palette Paired is 12
## Returning the palette you asked for with that many colors
```

Also, specifying the below code to preserve the layout each time you plot the network

```
lay.constant <- layout.kamada.kawai(net)
#add membership as a node attribute
V(net)$community <- c$membership
plot.igraph(simplify(net),
    layout=lay.constant,
    vertex.label=NA,
    vertex.color=c$membership,
    vertex.size=10,
    edge.arrow.size=.25)
```



**Assignment Question**

Change the color palette and try different layouts to see which one seems to capture the underlying community structure the best. Show at least two plots in your Rmarkdown file and discuss how the layout influences interpretation.