Workshop week 8: Visualising Networks

MXB262, 2020

27/4/2020 – PST due Sunday 3rd May 2020

Questions for PS4 are embedded below – Due Sunday 3rd May 2020

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 than building networks in `ggplot2` – sometimes it's better to use the right tool for a job, even when it is a little different to what you're used to. But take care, the way the graphs are built don't follow the same grammatical structure as in `ggplot2`.

The graphs are supressed here to save space, but you'll see them all as you run through the code.

Workshop Set-up

Setup

Install and load the following packages.

```r
library(igraph)
library(network)
library(sna)
library(ndtv)
library(tidyverse)
library(EpiContactTrace)
library(RColorBrewer)
library(viridis)
library(circlize)
library(alluvial)
```

Part 1

We will use data about media organisations in this workshop. The dataset contains information about the hyperlinks and mentions between online media outlets.

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.

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

As always, start by having a quick look at the data

```
head(media_vertices)
head(media_edges)

#or

glimpse(media_vertices)
glimpse(media_edges)
```

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 vertex for each edge. The following columns are edge attributes (weight, type, label, or anything else).
- `vertices` - starts with a column of vertex IDs. Any following columns are interpreted as vertex attributes.

```
net <- graph_from_data_frame(d = media_edges, vertices = media_vertices, directed= T)
net
```

```
## IGRAPH 14f5d48 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 14f5d48 (vertex names):
##  [1] s01->s02 s01->s03 s01->s04 s01->s15 s02->s01 s02->s03 s02->s09 s02->s10
##  [9] s03->s01 s03->s04 s03->s05 s03->s08 s03->s10 s03->s11 s03->s12 s04->s03
## [17] s04->s06 s04->s11 s04->s12 s04->s17 s05->s01 s05->s02 s05->s09 s05->s15
## [25] s06->s06 s06->s16 s06->s17 s07->s03 s07->s08 s07->s10 s07->s14 s08->s03
## [33] s08->s07 s08->s09 s09->s10 s10->s03 s12->s06 s12->s13 s12->s14 s13->s12
## [41] s13->s17 s14->s11 s14->s13 s15->s01 s15->s04 s15->s06 s16->s06 s16->s17
## [49] s17->s04
```

The 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 (vertices need a name attribute)
3. W for a weighted graph (edges need a weight attribute)
4. B for a bipartite graph (vertices need a type attribute)

The two numbers that follow (here, 17 49) refer to the number of vertices and edges in the graph. The description also lists vertex & edge attributes, which you can explore by browsing the help documentation for igraph.

You can also access the vertices, 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 vertices and edges by attribute:
# (that returns oblects of type vertex sequence/edge sequence)
```

```r
V(net)[media == "BBC"]
E(net)[type == "mention"]

# You can also examine the network matrix directly:
net[1,]
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 vertices 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.igraph(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 to create a simple, or acyclic network.

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

2. Also reduce the arrow size and remove the uninformative and cluttering labels by setting them to NA (which in `R` basically means 'this is empty'):

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

Plotting parameters

These network plots have a wide range of parameters you can adjust. They include vertex options or parameters (such as `vertex`) and edge options (such as `edge`). A list of options can be viewed by using `?igraph.plotting`. Explore that list of options now, including color, sizes, widths etc. This is where most of your aesthetic changes will be made, which is critical for this unit (and for any data communication!). We will explore some of these here, but not all of them.

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

First, let's just remember what the original ineffective version looked like

```r
plot.igraph(net)
```

Gross. Let's explore some more ways to change its appearance.

Using `plot.igraph()` to make changes

Changes through `plot.igraph()` are a blunt instrument, and can really only apply to the whole network (Q: what is the analogy in `ggplot.igraph()`?).

```r
# 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 edges between two vertices
# (e.g. edges going in either direction, or multiplex edges)
plot.igraph(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 vertex & border color to orange.
# Replace the vertex label with the vertex names stored in "media".
plot.igraph(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()` to make changes

Let's say we want to color our network vertices based on type of media, and size them based on degree centrality (more edges -> larger vertex). This requires us to be able to make changes to parts of the data, but not others. We can make those more fine-scale changes by altering the actual graph object, and saving these new plotting elements as parameters in the dataset (Q: what is the`ggplot2` analogy?). We will also change the width of the edges based on their weight.

```
# Generate colors based on media type:
# Let's take a look at the variable we want to use to assign colours: media type.
V(net)$media.type
```

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

```
# They are just numbers -- 1, 2, 3. So, we can easily use them as indices in a vector (remember that ve

# First, save a vector of colour names that you want to use:
colrs <- c("gray50", "tomato", "gold") # I haven't grabbed these from ColorBrewer2, but I should have!
# Then, make a new variable for the vertices of `net` that points to one of those colours depending on
V(net)$color <- colrs[V(net)$media.type]

# Compute vertex degrees (#edges) and use that to set vertex 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 vertex 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

# Now that we have made all those changes to the data structure `net`, when we plot it, we will see the
```

```
plot.igraph(net)
```

We can also override the attributes explicitly in the `plot.igraph()` after they have been set in `igraph()` – this is not necessarily a good idea, but is something to be aware of (Q: have you seen this issue cropping up in `ggplot2` as well?):

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

It helps to add a legend explaining the meaning of the colors we used. Look up the help file for `?legend` to explore any legend parameters you haven't seen before.

```r
plot.igraph(net)
legend(x = -1.5, y = -1.1, # this sets the position of the legend
       c( "Newspaper", "Television", "Online News"), # we need to know about the data to know what labe
       pch = 21,
       col = "#777777",
       pt.bg = colrs, # note that this is the vector of colours that we defined earlier
       pt.cex = 2,
       cex = .8,
       bty = "n",
       ncol =  1)
```

Sometimes we may be interested in plotting only the labels of the vertices, without any dots at all:

```r
plot.igraph(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 origin vertex. The `ends()` igraph function creates a matrix with a row for each edge, and two columns: the origin vertex, and the destination vertex. The edges you want to explore need to be listed in the `es` parameter – you can just do a subset of the edges, but here we will use all of them, `E(net)`. The `names` parameter controls whether the function returns edge names (`names=TRUE`) or a numerical ID (`names=FALSE`).

```r
edge.start <- ends(net, es=E(net), names=FALSE)[,1] # the number in the square brackets here is saying
edge.col <- V(net)$color[edge.start] # remember that above, we saved the colours of the vertices in the

plot.igraph(net, edge.color=edge.col, edge.curved=.4)
```

Task: make the same plot using the colours of the destination nodes instead. It is a very simple change.

Network layouts

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

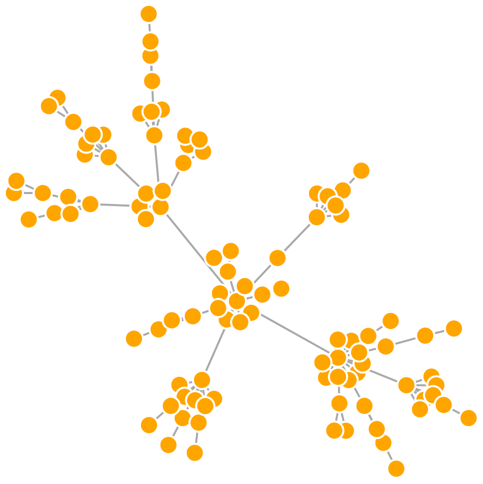
For the purposes of exploring layouts, we will generate a slightly larger 100-vertex graph using the sample_pa() function.

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

```r
plot.igraph(random.net)
```

Now that we know about effective visualisation theory, it is painful to look at ineffective visualisations. So let's make some initial changes here by altering the igraph object `random.net`. This illustrates how useful it is to change graphical parameters of the actual network object – now every time we plot it, it will use the same parameters.

```r
V(random.net)$size <- 8
V(random.net)$frame.color <- "white"
V(random.net)$color <- "orange"
V(random.net)$label <- ""
E(random.net)$arrow.mode <- 0
plot.igraph(random.net)
```



You can set the layout in the plot function:

```
plot.igraph(random.net, layout=layout_randomly)
```

Or you can calculate the vertex coordinates in advance, which is useful especially if the placement of the vertices means something in real life (like, if it is a spatial network)

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

`l` is just a matrix of x, y coordinates for the vertices in the graph. You can easily generate your own:

```
bad.layout <- cbind(1:100, c(1, 100:2))
plot.igraph(random.net, layout=bad.layout)
```

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(random.net)
plot.igraph(random.net, layout=l)
```

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

```
# 3D sphere layout
l <- layout_on_sphere(random.net)
plot.igraph(random.net, 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. vertices are electrically charged particles that repulse each other when they get too close. The edges act as springs that attract connected vertices closer together. As a result, vertices are evenly distributed through the chart area, and the layout is intuitive in that vertices which share more connections are closer to each other.

```
l <- layout_with_fr(random.net)
plot.igraph(random.net, 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 vertices to stay in the same place in multiple plots.

```
# compare these plots
plot.igraph(random.net, layout=layout_with_fr)
plot.igraph(random.net, layout=layout_with_fr)
plot.igraph(random.net, layout=l)
plot.igraph(random.net, layout=l)
```

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(random.net)
plot.igraph(random.net, layout=l)
```

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

```
plot.igraph(random.net, layout=layout_with_lgl)
```

The MDS (multidimensional scaling) algorithm tries to place vertices based on some measure of similarity or distance between them. More similar vertices are plotted closer to each other. By default, the measure used is based on the shortest paths between vertices 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: vertices often overlap, or are placed on top of each other.

```
plot.igraph(random.net, layout=layout_with_mds)
```

Bipartite networks layout

Bipartite networks need their own special layout. I'll have to introduce a new dataset here that happens to be a bipartite network for you to test this. Our second dataset is a network of edges between news outlets and media consumers. Save `Dataset2-Media-Example-NODES.csv` and `Dataset2-Media-Example-EDGES.csv` into your working directory.

```
media_vertices2 <- read.csv("data/Dataset2-Media-User-Example-NODES.csv", header = T, as.is = T)
media_edges2 <- read.csv("data/Dataset2-Media-User-Example-EDGES.csv", header = T, row.names = 1)
```

As always, examine these datasets using `head()` or `glimpse()`. When you view `media_edges2`, you can see that it's an adjacency matrix for a bipartite network – none of the rows or columns refer to the same nodes (see lecture). One group of vertices are news sources and, the other is consumers. Edges represent which news sources each customer consumes.

Let's convert this second dataset into an igraph object. The edges of our second network are in a matrix format. We can read bipartite matrices into a graph object using `graph_from_incidence_matrix()`. The structure of igraph represents bipartite networks by assigning all vertices attributes called `type`: it has a value of `FALSE` (or 0) for vertices in one group and `TRUE` (or 1) for those in the other group

```
net2 <- graph_from_incidence_matrix(media_edges2) # build the igraph object
V(net2) # take a look at just the vertices using the V() function
```

```
## + 30/30 vertices, named, from baf45f9:
##  [1] s01 s02 s03 s04 s05 s06 s07 s08 s09 s10 U01 U02 U03 U04 U05 U06 U07 U08 U09
## [20] U10 U11 U12 U13 U14 U15 U16 U17 U18 U19 U20
```

```
head(V(net2)$type) # take a look at the `type` attributes of the vertices
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE
```

```
table(V(net2)$type) # summarise the numbers of each type
```

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

> NOTE: If your data is a non-bipartite network matrix (i.e. just normal, with edges between any vertices), it can be transformed into an `igraph` object, using `graph_from_adjacency_matrix()` instead of `graph_from_incidence_matrix()`

Try plotting it as a normal network:

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

In `igraph`, there is also a special layout for bipartite networks, that makes it more clear what is going on with these special types of networks.

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

Using layout to zoom

You can make a more spreadout plot by manipulating the layout argument. Try these plots:

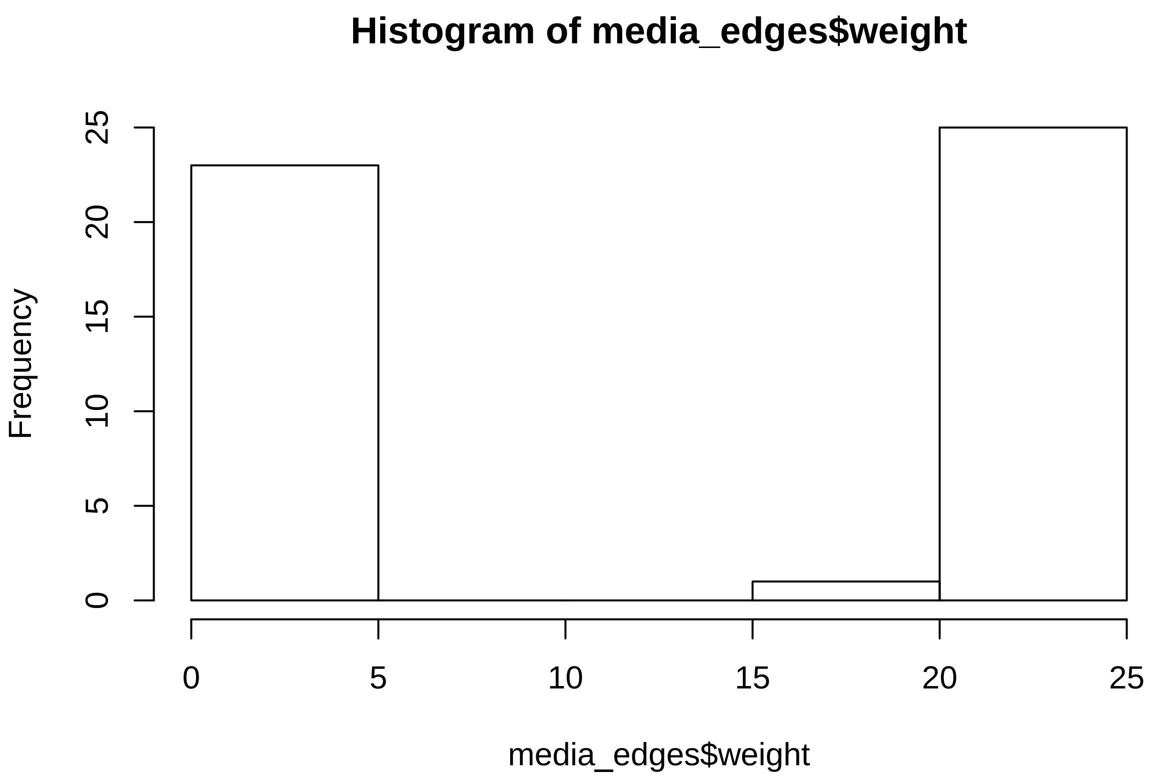
```
l <- layout_on_sphere(random.net)

plot.igraph(random.net, rescale=F, layout=l*0.4)
plot.igraph(random.net, rescale=F, layout=l*0.6)
plot.igraph(random.net, rescale=F, layout=l*0.8)
plot.igraph(random.net, rescale=F, layout=l*1.0)
plot.igraph(random.net, rescale=F, layout=l*3.0)
```

Highlight aspects of the network

Notice that our network plot is still not too helpful. We can identify the type and size of vertices, but cannot see much about the structure since the edges we're examining are so dense. One way to approach this is to see if we can make the network more sparse, keeping only the most important edges and discarding the rest. First, let's get a look at how the weights of our edges are distributed

```
hist(media_edges$weight)
```



```r
mean(media_edges$weight)
```

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

```r
sd(media_edges$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)`:

```r
cut.off <- mean(media_edges$weight)
net.sparse <- delete_edges(net, E(net)[weight<cut.off])
plot.igraph(net, layout=layout_with_kk)
plot.igraph(net.sparse, layout=layout_with_kk)
```

Another way to think about this is to plot the two types of edges we have (hyperlinks & mentions) separately. Remind yourself of this with `head(media_edges)` We will do that in section 5 of this tutorial below: Plotting multiplex networks.

We can also try to make the network map more useful by showing the communities within it. igraph can natively do this using the `cluster_optimal()` function, which identifies communities of vertices that are more closely linked to each other than to other vertices (see lecture).

```r
# First, detect the communities using the inbuilt igraph function
net.communities <- cluster_optimal(net)
```

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

Oh, that looks….. not great, or effective. Unfortunately it's the only inbuilt way to see the communities. But now that we have calculated the communites, we can use them.

Remember that you can see the variables within net.communities by typing `net.communities$` and waiting to see the pop up menu. Let's use thay to plot the communities without relying on their built-in plot.

```
net.communities$membership
```

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

```
# We can save the membership of each vertex as a variable of the igraph object V(net)
V(net)$community <- net.communities$membership
colrs <- adjustcolor( c("gray50", "tomato", "gold", "yellowgreen"), alpha=.6)
```

```
plot.igraph(net, vertex.color=colrs[V(net)$community])
```

Highlight specific vertices or edges

Sometimes we want to focus the visualization on a particular vertex or a group of vertices. In our example media network, we can examine the spread of information from focal vertices. For instance, let's explore distance from the New York Times (i.e. the number of hyperlinks or mentions of the NYT).

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

```
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.igraph(net, vertex.color=col, vertex.label=dist.from.NYT, edge.arrow.size=.6,
     vertex.label.color="white")
```

We can also point to the immediate neighbors of a vertex, say WSJ. The `neighbors` function in `igraph` finds all vertices that are directly linked to the chosen vertex.

```
neigh.vertices <- neighbors(net, V(net)[media=="Wall Street Journal"], mode="out")

# Make the default colour for all of the vertices a grey
vertex.col <- rep("grey40", vcount(net)) # vcount() just tells us how many vertices there are

# Set colors to plot the neighbors:

vertex.col[neigh.vertices] <- "#ff9d00"

plot.igraph(net, vertex.color=vertex.col)
```

Plotting multiple networks

Recall that our media dataset has two types of edges – mentions, and hyperlinks. We can use the variable of the edges to identify which edges belong to which type.

```
E(net)[E(net)$type=="hyperlink"]
```

```
## + 28/48 edges from 807ea86 (vertex names):
##  [1] s01->s02 s01->s03 s01->s04 s02->s01 s02->s03 s02->s09 s02->s10 s03->s01
##  [9] s03->s04 s03->s05 s03->s08 s03->s11 s03->s12 s04->s03 s04->s12 s05->s02
## [17] s05->s09 s06->s16 s07->s10 s08->s03 s10->s03 s12->s13 s13->s12 s15->s01
## [25] s15->s04 s15->s06 s16->s06 s17->s04
```

```
E(net)[E(net)$type=="mention"]
```

```
## + 20/48 edges from 807ea86 (vertex names):
##  [1] s01->s15 s03->s10 s04->s06 s04->s11 s04->s17 s05->s01 s05->s15 s06->s17
##  [9] s07->s03 s07->s08 s07->s14 s08->s07 s08->s09 s09->s10 s12->s06 s12->s14
## [17] s13->s17 s14->s11 s14->s13 s16->s17
```

We can remove those edges from the igraph object we created by using the minus operator:

```
net.mentions <- net - E(net)[E(net)$type=="hyperlink"]
net.hyperlinks <- net - E(net)[E(net)$type=="mention"]
```

```
# Plot the two edges separately:
plot.igraph(net.hyperlinks, vertex.color="orange", layout=layout_with_fr, main="Tie: Hyperlink")
plot.igraph(net.mentions, vertex.color="lightsteelblue2", layout=layout_with_fr, main="Tie: Mention")
```

TASK: Between those two plots, the vertices move which is irritating. How can you apply what we have learned so far to make the vertices stay in the same place between the two plots?

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 visualising the network directly.

Networks as heatmaps

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) )
```

Networks as chord diagrams

Chord diagrams can show flows through networks (see lecture). Here we will use a dataset showing numbers of people migrating from one geographic area to each other.

```
# Load dataset from github
data <- read.table("https://raw.githubusercontent.com/holtzy/data_to_viz/master/Example_dataset/13_Adja
```

As always, take a look at the data with `head(data)`

As you'll soon see, space is at a premium when it comes to labelling chord diagrams when we rely on the base plot to to the labelling for us. It's possible to get much fancier, but this is enough for us now (online tutorials are great if you want to explore this more). For now, we will just shorten the names of the regions.

```
# short names
colnames(data) <- c("Africa", "EAsia", "Europe", "LatinAm.",   "NorthAm.",   "Oceania", "SAsia", "SEAsi
rownames(data) <- colnames(data)

# The data is in the form of an adjacency matrix, but we need it in a three column matrix instead: firs
data_long <- gather(rownames_to_column(data), key='key', value='value', -rowname)
```

```r
# color palette
mycolor <- viridis(nrow(data)) # generates 10 colours from the viridis palette, one for each node in th

# Base plot
chordDiagram(
  x = data_long,
  grid.col = mycolor,
  transparency = 0.25,
  directional = 1,
  direction.type = c("arrows", "diffHeight"),
  diffHeight  = -0.04,
#  annotationTrack = "grid",
#  annotationTrackHeight = c(0.05, 0.1),
  link.arr.type = "big.arrow",
  link.sort = TRUE)
```

Networks as alluvial plots

```r
alluvial (data_long[,1:2], freq=data_long[,3])
```

PS4 question 1.

Using the `alluvial` help menu and the very brief starter code above, create an effective alluvial plot with the same colour palette as the chord diagram example.

Part 2: Cattle Network Example

We are now going to use `igraph` to explore a different dataset about cattle movement data. Throughout this section, there are 5 questions total for PS4 (one is above in the alluvial section, and 4 are following here). You're using this new dataset here, and you can use methods either from this section or from the previous section to answer the remaining four questions.

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

```r
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

```r
head(edges)
```

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

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

```r
net_edges
```

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

```r
head(attr1)
```

We will add `production` to each vertex.

The `V()` function indicates a vertex attribute of the specified network. `V()$name` indicates what we would like to name the attribute. The match command ensures that the production type added to each vertex is the correct one for that vertex by matching the name of the vertex 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)])
```

Now add herd size

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

Plot the network

Prepare custom colors for plotting by creating an attribute for vertex color. In the previous example, we were lucky that the attributes we wanted to correspond with colours were just the numbers 1,2,3, so it was really easy to just use them as indices of a vector with colour names in it. However, we almost never see that in real data. Here, we want to use the farm size (a categorical variable) to dictate the colour. We will use the `gsub()` function to achieve that. `gsub(pattern, replacement, x)` takes three arguments: everywhere it sees the value given by the `pattern`, it replaces it with the value given by the `replacement`, within the dataset `x`. It's a quick and easy way to change words into other words. First, we will just replicate the `type` variable and save it as the variable `color`, but then one by one we will replace the types with colours. Between each line here, print your `net_edges$color` to take a look at what the function is doing.

```r
V(net_edges)$color <- V(net_edges)$type
head(V(net_edges)$color)
```

```
## [1] "Fattening" "Dairy"     "Fattening" "Dairy"     "Dairy"     "Dairy"
```

```r
V(net_edges)$color <- gsub("Fattening","steelblue1",V(net_edges)$color)
head(V(net_edges)$color)
```

```
## [1] "steelblue1" "Dairy"      "steelblue1" "Dairy"      "Dairy"
## [6] "Dairy"
```

```r
V(net_edges)$color <- gsub("Dairy","darkgoldenrod1",V(net_edges)$color)
head(V(net_edges)$color)
```

```
## [1] "steelblue1"     "darkgoldenrod1" "steelblue1"     "darkgoldenrod1"
## [5] "darkgoldenrod1" "darkgoldenrod1"
```

```r
V(net_edges)$color <- gsub("Small farm","mediumpurple",V(net_edges)$color)
V(net_edges)$color <- gsub(pattern="Breeding",replacement="navy",x=V(net_edges)$color)
V(net_edges)$color <- gsub(pattern="Complete cycle",replacement="magenta2",x=V(net_edges)$color)
V(net_edges)$color <- gsub(pattern="Growing",replacement="green4",x=V(net_edges)$color)
```

You could similarly specify shapes by creating a shape attribute!

Plot the new, improved network

```r
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")
```

PS4 question 2.

Try any four different layouts for this network. Describe in a few sentences how the different layouts convey different impressions about the structure of the network.

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

PS4 question 3.

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 (breeding, steers, calves, etc).

```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)
```

```r
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)
```

PS4 question 4.

Starting from the full network (not this new reduced network), filter for and plot only movements of heifers. Plot both the full network and this new filtered one. What observations can you make from comparing the two?

PS4 question 5.

Plot the communities of the full network.