Gov 2018: Social Networks

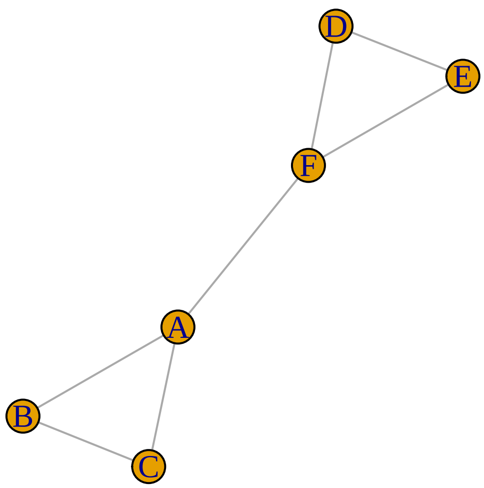
Your name:

Apr 12, 2022

There are a ton of R packages available for network analysis, however, here you're going to focus on using the `igraph` package throughout (Gabor & Nepusz 2006, http://igraph.org). It combines ease of use and high-level computation (it is also available for Python and C/C++). This package has proven to be very useful, and it covers a lot of basic network analysis methods as well as plotting capabilities. This exercise uses materials from Dai Shizuka and Pablo Barbera.

To see how to manually make a network and take a look at it before you dive in:

```
g=make_graph(~A-B-C-A, D-E-F-D, A-F)
plot(g)
```




```
#look at vertices
V(g)
```

```
## + 6/6 vertices, named, from 3e9b050:
## [1] A B C D E F
```

```
#look at edges
E(g)
```

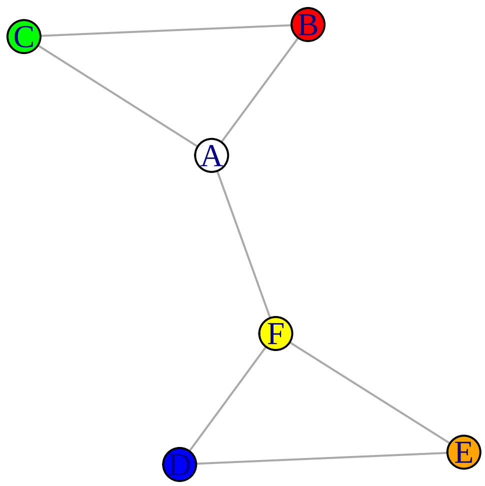
```
## + 7/7 edges from 3e9b050 (vertex names):
## [1] A--B A--C A--F B--C D--E D--F E--F
```

```
#look at vertex attributes (currently just name)
V(g)$name
```

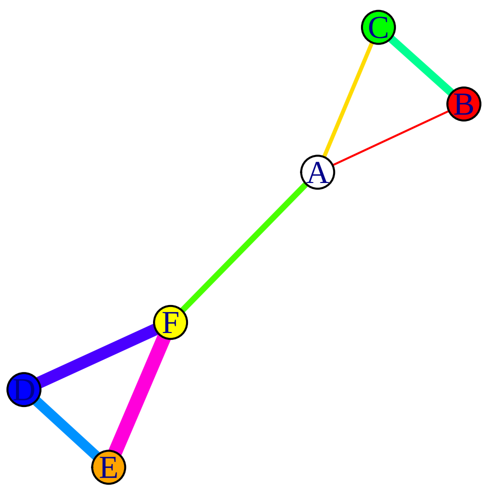
```
## [1] "A" "B" "C" "D" "E" "F"
```

```
#Create new vertex attributes -- some can actually be directly interpreted by igraph, such as color
V(g)$color=c("white", "red", "green", "blue", "orange", "yellow") #a random set of colors
```

```r
plot(g)
```




```r
#Add edge attributes
E(g)$width=1:7
E(g)$color=rainbow(7) #rainbow() function chooses a specified number of colors
plot(g)
```



1 Exploring network data formats

1.1 Adjacency matrix, Edge list, Affiliation

Extract the adjacency matrix of `g` using `as_adjacency_matrix`. Then pull out the edgelist using `as_edgelist`.

```r
# extract the adjacency matrix
as_adjacency_matrix(g)
```

```
## 6 x 6 sparse Matrix of class "dgCMatrix"
##   A B C D E F
## A . 1 1 . . 1
## B 1 . 1 . . .
## C 1 1 . . . .
```

```
## D . . . . 1 1
## E . . . 1 . 1
## F 1 . . 1 1 .
```

```r
# pull out edge list
as_edgelist(g)
```

```
##      [,1] [,2]
## [1,] "A"  "B"
## [2,] "A"  "C"
## [3,] "A"  "F"
## [4,] "B"  "C"
## [5,] "D"  "E"
## [6,] "D"  "F"
## [7,] "E"  "F"
```

1.2 Affiliation matrix

In many cases, we will construct social networks from co-membership in groups. For example, we would draw edges between individuals based on their patterns of co-occurrence in a flock. Similarly, we could construct networks of species co-occurrences in populations, etc.

To do this, we would first need data in a matrix in which rows represent individuals (or species) and columns represent groups (or populations). Note that you could flip the columns and rows–either way is fine. You just need to be aware of how you arranged it.

Below is a toy example in which individuals A through E occur in different combinations in 4 groups.

```r
A=c(1,1,0,0)
B=c(1,0,1,0)
C=c(1,0,1,0)
D=c(0,1,0,1)
E=c(0,0,1,1)
aff=matrix(c(A,B,C,D,E),nrow=5,byrow=TRUE)
dimnames(aff)=list(c("A","B","C","D","E"),c("Group1","Group2","Group3","Group4"))
aff #The individual-by-group matrix
```

```
##   Group1 Group2 Group3 Group4
## A      1      1      0      0
## B      1      0      1      0
## C      1      0      1      0
## D      0      1      0      1
## E      0      0      1      1
```

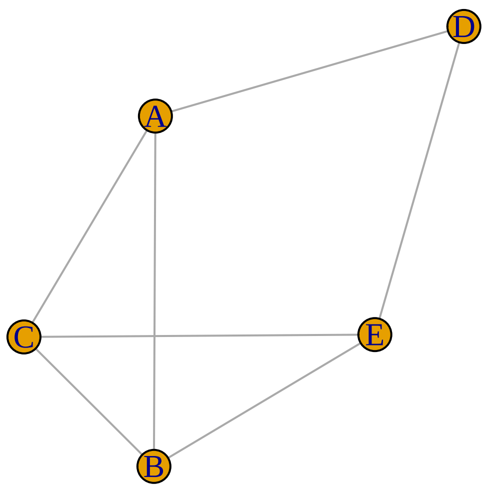
There are different ways to convert this data into a social network–i.e., a network that describes which individual co-occurs with which individual in groups. One simple way is to do what is called a one-mode projection of this data by multiplying this matrix with the transpose of itself. Do such a projection on `aff` and print it out. Then take the projection and plot the adjacency matrix from it where edges are weighted by the variable `weight`.

```r
# one-mode projection of aff: multiply aff with the transpose of itself
# print it out
aff_t <- aff %*% t(aff)

aff_t
```

```
##   A B C D E
```

```
## A 2 1 1 1 0
## B 1 2 2 0 1
## C 1 2 2 0 1
## D 1 0 0 2 1
## E 0 1 1 1 2
```

```r
# create igraph object using graph_from_adjacency_matrix() with mode = "undirected", weighted = T, diag
# plot the graph using the edge attribute named `weight` as your edge.width
igraph_obj <- graph_from_adjacency_matrix(aff_t, mode = "undirected", weighted = T, diag = F)

plot(igraph_obj)
```



2 Real data

Here you will be using a small network that indicates interactions in the movie Star Wars Episode IV. Each node is a character and each edge indicates whether they appeared together in a scene of the movie. Edges here are thus undirected and they also have weights attached, since they can appear in multiple scenes together.

The first step is to read the list of edges and nodes in this network:

```r
edges <- read.csv("star-wars-network-edges.csv")
head(edges)
```

```
##      source target weight
## 1     C-3PO  R2-D2     17
## 2      LUKE  R2-D2     13
## 3   OBI-WAN  R2-D2      6
## 4      LEIA  R2-D2      5
## 5       HAN  R2-D2      5
## 6 CHEWBACCA  R2-D2      3
```

```r
nodes <- read.csv("star-wars-network-nodes.csv")
head(nodes)
```

```
##          name id
## 1       R2-D2  0
## 2   CHEWBACCA  1
## 3       C-3PO  2
## 4        LUKE  3
```

```
## 5 DARTH VADER  4
## 6       CAMIE  5
```

For example, we learn that C-3PO and R2-D2 appeared in 17 scenes together.

How do we convert these two datasets into a network object in R? There are multiple packages to work with networks, but the most popular is `igraph` because it's very flexible and easy to do. Other packages that you may want to explore are `sna` and `networks`.

Now, how do we create the igraph object? We can use the `graph_from_data_frame` function, which takes two arguments: `d`, the data frame with the edge list in the first two columns; and `vertices`, a data frame with node data with the node label in the first column. (Note that igraph calls the nodes `vertices`, but it's exactly the same thing.)

```
g <- graph_from_data_frame(d=edges, vertices=nodes, directed=FALSE)
g
```

```
## IGRAPH 0a0033b UNW- 22 60 --
## + attr: name (v/c), id (v/n), weight (e/n)
## + edges from 0a0033b (vertex names):
##  [1] R2-D2      --C-3PO       R2-D2      --LUKE        R2-D2      --OBI-WAN
##  [4] R2-D2      --LEIA        R2-D2      --HAN         R2-D2      --CHEWBACCA
##  [7] R2-D2      --DODONNA     CHEWBACCA  --OBI-WAN     CHEWBACCA  --C-3PO
## [10] CHEWBACCA  --LUKE        CHEWBACCA  --HAN         CHEWBACCA  --LEIA
## [13] CHEWBACCA  --DARTH VADER CHEWBACCA  --DODONNA     LUKE       --CAMIE
## [16] CAMIE      --BIGGS       LUKE       --BIGGS       DARTH VADER--LEIA
## [19] LUKE       --BERU        BERU       --OWEN        C-3PO      --BERU
## [22] LUKE       --OWEN        C-3PO      --LUKE        C-3PO      --OWEN
## + ... omitted several edges
```

What does it mean? - `U` means undirected
- `N` means named graph
- `W` means weighted graph
- `22` is the number of nodes
- `60` is the number of edges
- `name (v/c)` means *name* is a node attribute and it's a character
- `weight (e/n)` means *weight* is an edge attribute and it's numeric

2.1 Practice accessing elements of the network

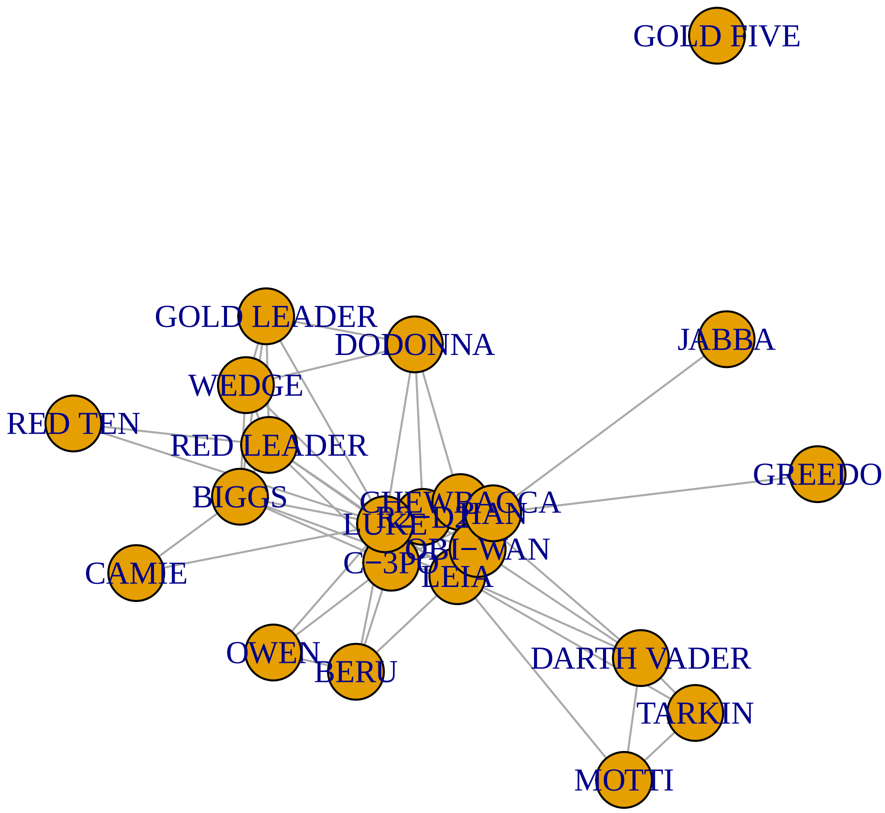
Practice accessing the following elements of the network: nodes, names of the nodes, attributes of the nodes, edges, weights for each edge, all attributes of the edges, the adjacency matrix, and just the first row of the adjacency matrix.

```
# nodes
# names of each node
# all attributes of the nodes
# edges
# weights for each edge
# all attributes of the edges
# adjacency matrix
# first row of adjacency matrix
```

2.2 Network visualization

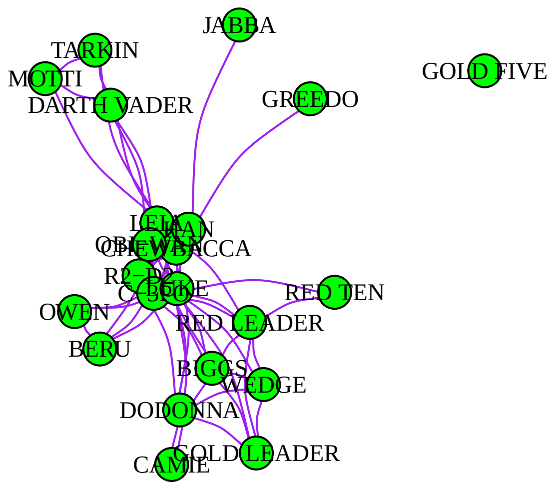
How can we visualize this network? The `plot()` function works out of the box, but the default options are often not ideal:

```
par(mar=c(0,0,0,0))
plot(g)
```



Improve this figure. To see all the available plotting options, you can check `?igraph.plotting`. Set the vertex color, label colors, the size of the labels, curvature to the edge and edge color to ones different from the default settings and in a way that is visually appealing to you.

```
# change color of nodes
# change color of labels
# change size of labels to 75% of original size
# add a 25% curve to the edges
# change edge color to grey

plot.igraph(g,
            vertex.color = "green",
            vertex.label.color = "black",
            vertex.label.cex = .75,
            edge.curved=.25,
            edge.color = "purple")
```



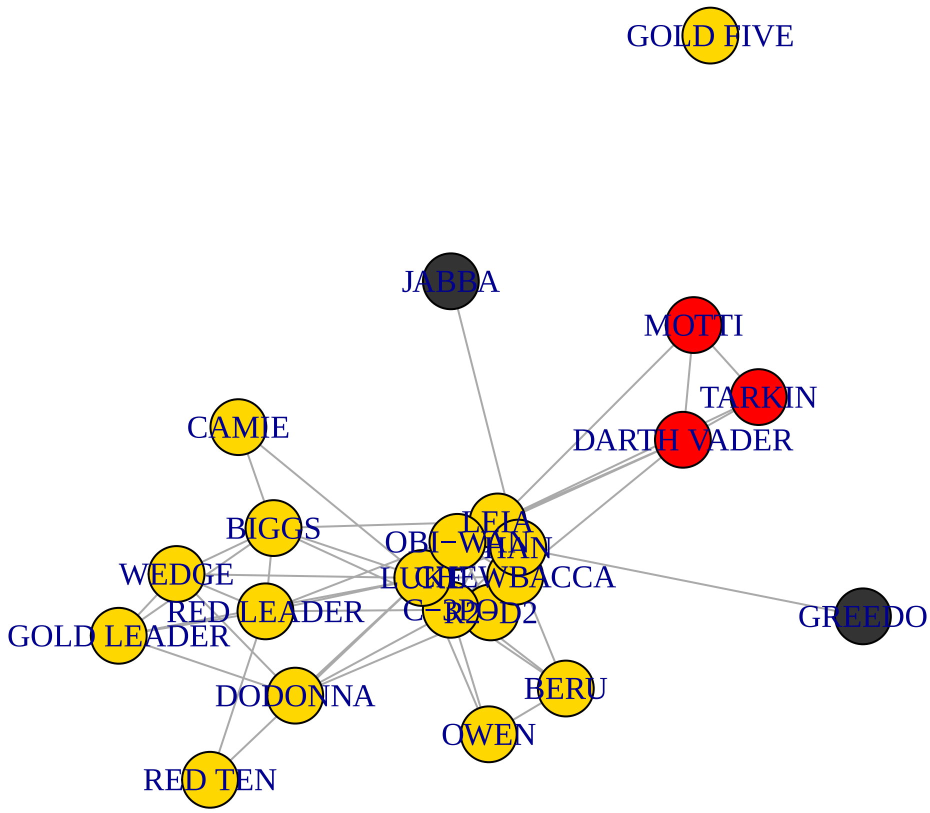
Now modify some of these plotting attributes so that they are function of network properties. For example, a common adjustment is to change the size of the nodes and node labels so that they match their `importance`.

Here, `strength` will correspond to the number of scenes they appear in. Let the size of the node be determined by the `strength`, and only show the labels of character that appear in 10 or more scenes. Finally change the colors of the node based on what side they're in (dark side or light side) and add an informative legend.

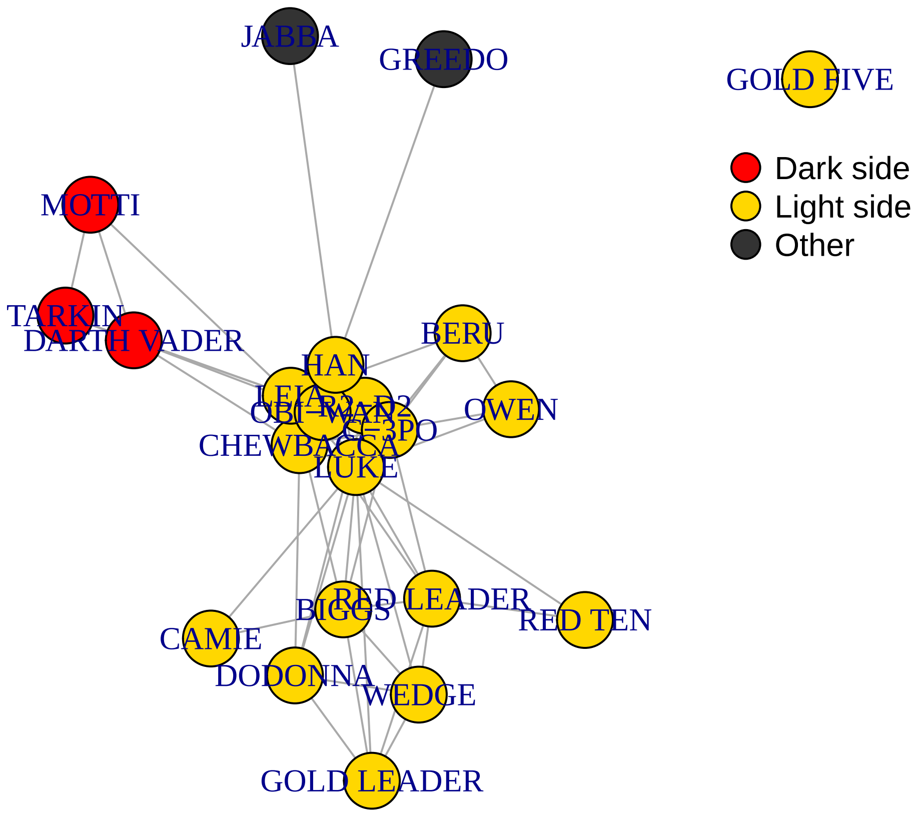
```
# let the size of the node be determined by the `strength`:
# - you may need to take log to improve the visualization
# - only show the labels of character that appear in 10 or more scenes

# Change the colors of each node based on what side they're in (dark side or light side):
# - create vectors with characters in each side
dark_side <- c("DARTH VADER", "MOTTI", "TARKIN")
light_side <- c("R2-D2", "CHEWBACCA", "C-3PO", "LUKE", "CAMIE", "BIGGS",
                "LEIA", "BERU", "OWEN", "OBI-WAN", "HAN", "DODONNA",
                "GOLD LEADER", "WEDGE", "RED LEADER", "RED TEN", "GOLD FIVE")
other <- c("GREEDO", "JABBA")
# - create a new color variable as a node property
V(g)$color <- NA
V(g)$color[V(g)$name %in% dark_side] <- "red"
V(g)$color[V(g)$name %in% light_side] <- "gold"
V(g)$color[V(g)$name %in% other] <- "grey20"
vertex_attr(g)
```

```
## $name
##  [1] "R2-D2"       "CHEWBACCA"   "C-3PO"       "LUKE"        "DARTH VADER"
##  [6] "CAMIE"       "BIGGS"       "LEIA"        "BERU"        "OWEN"
## [11] "OBI-WAN"     "MOTTI"       "TARKIN"      "HAN"         "GREEDO"
## [16] "JABBA"       "DODONNA"     "GOLD LEADER" "WEDGE"       "RED LEADER"
## [21] "RED TEN"     "GOLD FIVE"
##
## $id
##  [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
##
## $color
##  [1] "gold"   "gold"   "gold"   "gold"   "red"    "gold"   "gold"   "gold"
##  [9] "gold"   "gold"   "gold"   "red"    "red"    "gold"   "grey20" "grey20"
## [17] "gold"   "gold"   "gold"   "gold"   "gold"   "gold"
```

```
par(mar=c(0,0,0,0)); plot(g)
```




```
# - assign the color based on its side

# Plot the result with legend
par(mar=c(0,0,0,0)); plot(g)
legend(x=.75, y=.75, legend=c("Dark side", "Light side", "Other"),
       pch=21, pt.bg=c("red", "gold", "grey20"), pt.cex=2, bty="n")
```

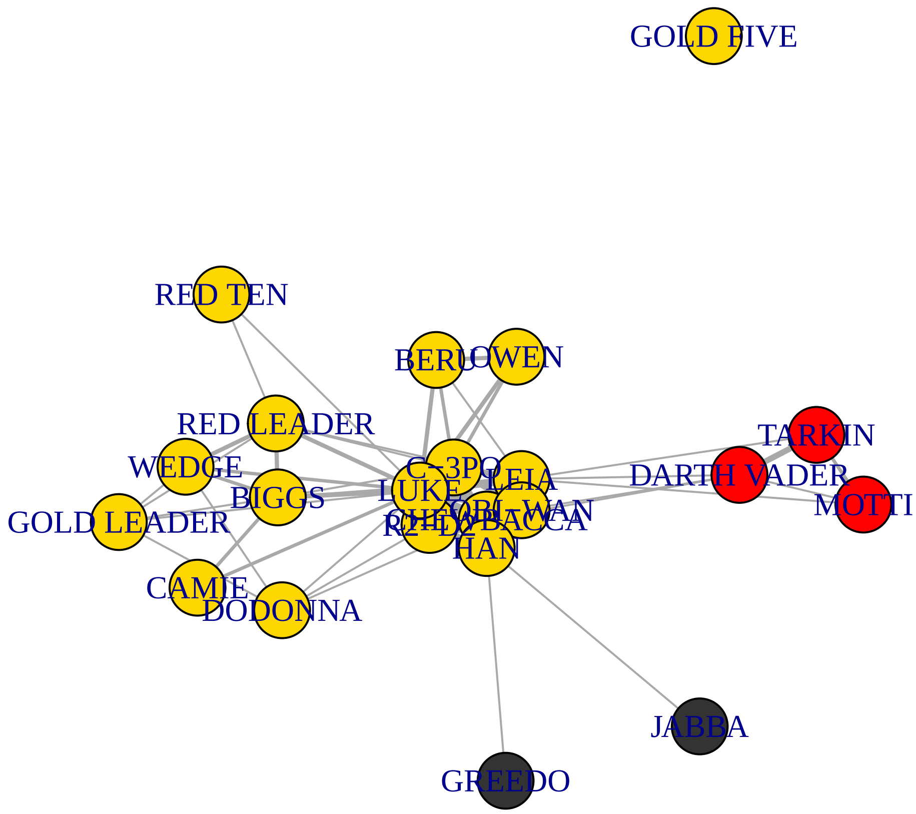


Edge properties can also be modified. Set the width of each edge as a function of the log number of scenes two characters appear together. Plot it.

```r
# Hint: Use E(g)$width
E(g)$width <- log(E(g)$weight) + 1
edge_attr(g)
```

```
## $weight
##  [1] 17 13  6  5  5  3  1  7  5 16 19 11  1  1  2  2  4  1  3  3  2  3 18  2  6
## [26] 17  1 19  6  1  2  1  7  9 26  1  1  6  1  1 13  1  1  1  1  1  1  2  1  1
## [51]  3  3  1  1  3  1  2  1  1  1
##
## $width
##  [1] 3.833213 3.564949 2.791759 2.609438 2.609438 2.098612 1.000000 2.945910
##  [9] 2.609438 3.772589 3.944439 3.397895 1.000000 1.000000 1.693147 1.693147
## [17] 2.386294 1.000000 2.098612 2.098612 1.693147 2.098612 3.890372 1.693147
## [25] 2.791759 3.833213 1.000000 3.944439 2.791759 1.000000 1.693147 1.000000
## [33] 2.945910 3.197225 4.258097 1.000000 1.000000 2.791759 1.000000 1.000000
## [41] 3.564949 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.693147
## [49] 1.000000 1.000000 2.098612 2.098612 1.000000 1.000000 2.098612 1.000000
## [57] 1.693147 1.000000 1.000000 1.000000
```

```r
par(mar=c(0,0,0,0)); plot(g)
```

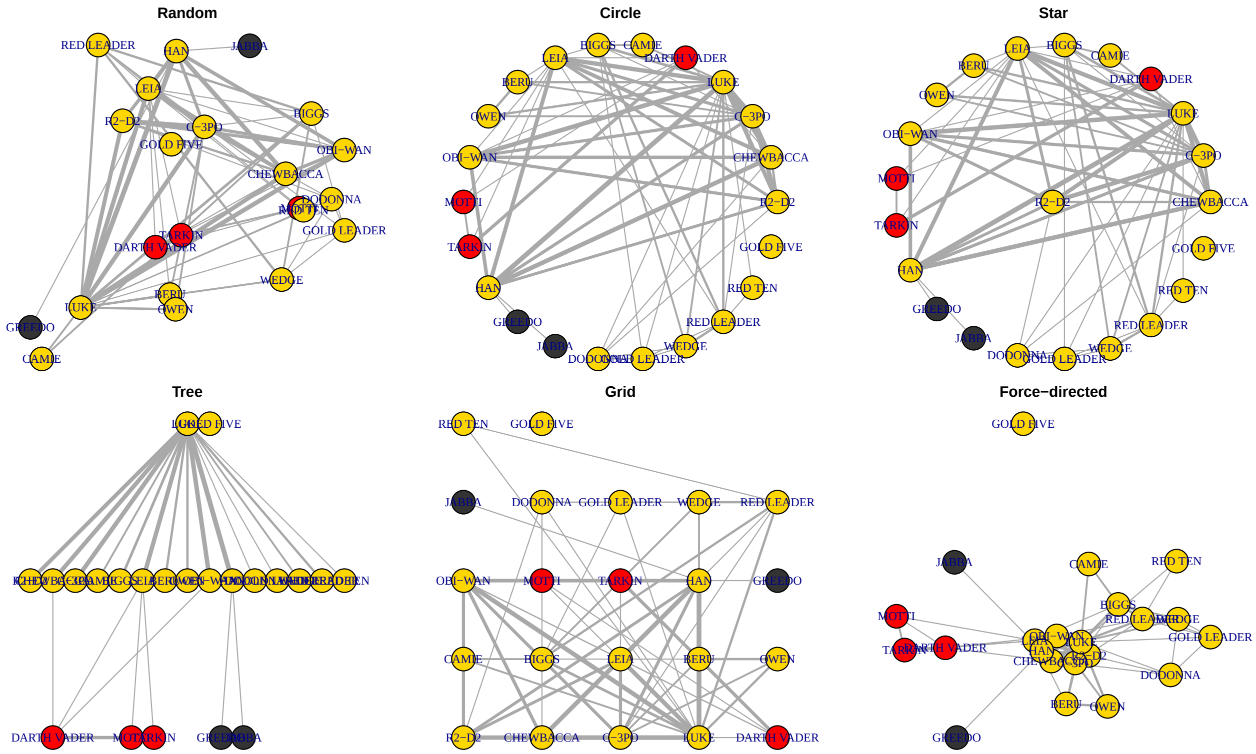


Extra: layouts

Up to now, each time we run the `plot` function, the nodes appear to be in a different location. Why? Because it's running a probabilistic function trying to locate them in the optimal way possible.

However, we can also specify the **layout** for the plot; that is, the (x,y) coordinates where each node will be placed. `igraph` has a few different layouts built-in, that will use different algorithms to find an `optimal` distribution of nodes. The following code illustrates some of these:

```
par(mfrow=c(2, 3), mar=c(0,0,1,0))
plot(g, layout=layout_randomly, main="Random")
plot(g, layout=layout_in_circle, main="Circle")
plot(g, layout=layout_as_star, main="Star")
plot(g, layout=layout_as_tree, main="Tree")
plot(g, layout=layout_on_grid, main="Grid")
plot(g, layout=layout_with_fr, main="Force-directed")
```



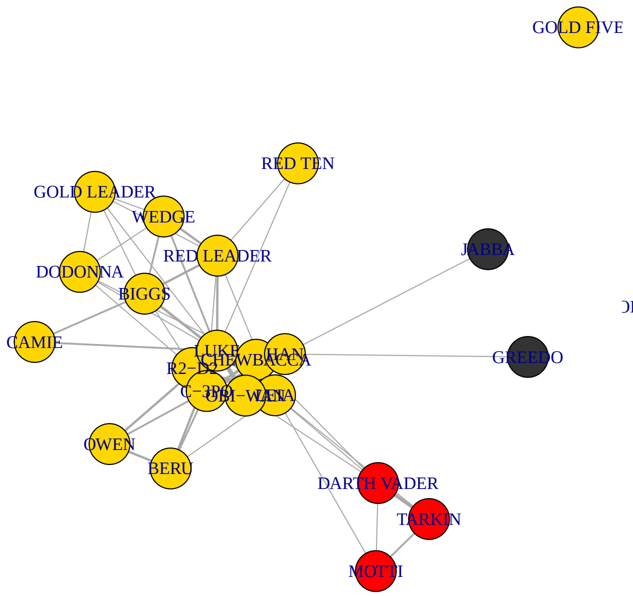
Note that each of these is actually just a matrix of (x,y) locations for each node.

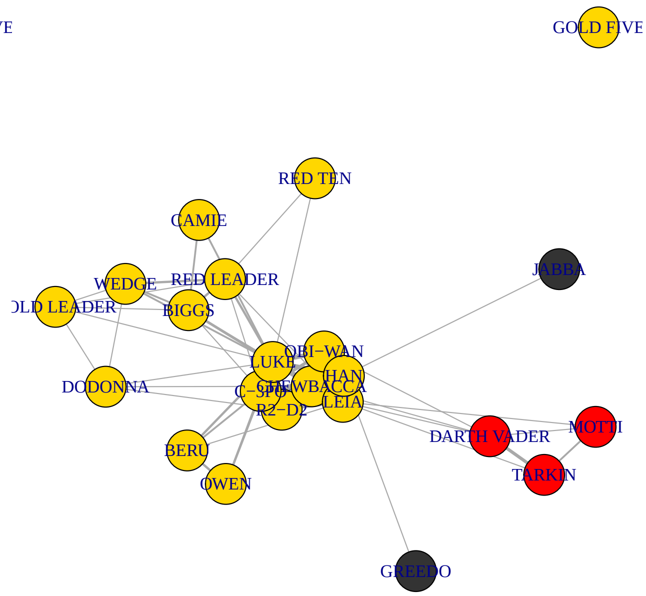
```
l <- layout_randomly(g)
str(l)
```

```
##  num [1:22, 1:2] -0.1635 0.4798 0.5684 -0.0622 -0.6679 ...
```

The most popular layouts are force-directed. These algorithms, such as Fruchterman-Reingold, try to position the nodes so that the edges have similar length and there are as few crossing edges as possible. The idea is to generate "clean" layouts, where nodes that are closer to each other share more connections in common that those that are located further apart. Note that this is a non-deterministic algorithm: choosing a different seed will generate different layouts.

```
par(mfrow=c(1,2))
set.seed(777)
fr <- layout_with_fr(g, niter=1000)
par(mar=c(0,0,0,0)); plot(g, layout=fr)
set.seed(666)
fr <- layout_with_fr(g, niter=1000)
par(mar=c(0,0,0,0)); plot(g, layout=fr)
```





3 Node properties

Let's look at descriptive statistics at the node level. All of these are in some way measures of importance or **centrality**.

The most basic measure is **degree**, the number of adjacent edges to each node. It is often considered a measure of direct influence. In the Star Wars network, it will be the unique number of characters that each character is interacting with. Sort the `degree` of the network and print it out.

```r
# sort the `degree` of the network and print it out

sort(degree(g))
```

```
##   GOLD FIVE      GREEDO       JABBA       CAMIE     RED TEN        OWEN
##           0           1           1           2           2           3
##       MOTTI      TARKIN        BERU DARTH VADER     DODONNA GOLD LEADER
##           3           3           4           5           5           5
##       WEDGE       R2-D2       BIGGS     OBI-WAN  RED LEADER   CHEWBACCA
##           5           7           7           7           7           8
##         HAN       C-3PO        LEIA        LUKE
##           8          10          12          15
```

In directed graphs, there are three types of degree: indegree (incoming edges), outdegree (outgoing edges), and total degree. You can find these using `mode="in"` or `mode="out"` or `mode="total"`.

Strength is a weighted measure of degree that takes into account the number of edges that go from one node to another. In this network, it will be the total number of interactions of each character with anybody else. Sort the `strength` of the network and print it out.

```r
# sort the `strength` of the network and print it out

sort(strength(g))
```

```
##   GOLD FIVE      GREEDO       JABBA     RED TEN       CAMIE       MOTTI
##           0           1           1           2           4           4
##     DODONNA GOLD LEADER        OWEN        BERU       WEDGE      TARKIN
##           5           5           8           9           9          10
```

```
## DARTH VADER  RED LEADER       BIGGS     OBI-WAN       R2-D2        LEIA
##          11          13          14          49          50          59
##   CHEWBACCA       C-3PO         HAN        LUKE
##          63          64          80         129
```

Closeness measures how many steps are required to access every other node from a given node. It's a measure of how long information takes to arrive (who hears news first?). Higher values mean less centrality. Sort the `closeness` of the network (normalize it) and print it out.

```r
# sort the `closeness` of the network and print it out

sort(closeness(g, normalized=TRUE))
```

```
## Warning in closeness(g, normalized = TRUE): At centrality.c:2617 :closeness
## centrality is not well-defined for disconnected graphs
```

```
##   GOLD FIVE      GREEDO       JABBA         HAN        OWEN       CAMIE
##  0.04545455  0.11666667  0.11666667  0.13043478  0.17647059  0.18584071
##      TARKIN       R2-D2     OBI-WAN       MOTTI DARTH VADER        BERU
##  0.20000000  0.20388350  0.20792079  0.21000000  0.21649485  0.21649485
##   CHEWBACCA       WEDGE     RED TEN       C-3PO        LUKE        LEIA
##  0.21875000  0.21875000  0.22105263  0.23595506  0.23863636  0.24418605
## GOLD LEADER  RED LEADER     DODONNA       BIGGS
##  0.24418605  0.25000000  0.25301205  0.25925926
```

Betweenness measures brokerage or gatekeeping potential. It is (approximately) the number of shortest paths between nodes that pass through a particular node. Sort the `betweenness` of the network and print it out.

```r
# sort the `betweenness` of the network and print it out

sort(betweenness(g))
```

```
##       CAMIE        OWEN     OBI-WAN       MOTTI      TARKIN      GREEDO
##    0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
##       JABBA       WEDGE   GOLD FIVE        BERU     RED TEN DARTH VADER
##    0.000000    0.000000    0.000000    1.666667    2.200000   15.583333
##   CHEWBACCA        LUKE       R2-D2 GOLD LEADER  RED LEADER       BIGGS
##   15.916667   18.333333   22.750000   23.800000   31.416667   31.916667
##       C-3PO         HAN     DODONNA        LEIA
##   32.783333   37.000000   47.533333   59.950000
```

Eigenvector centrality is a measure of being well-connected connected to the well-connected. First eigenvector of the graph adjacency matrix. Only works with undirected networks. Sort the returned *vector* from the `eigen_centrality` of the network and print it out.

```r
# sort the `vector` from the `eigen_centrality` of the network and print it out

sort(eigen_centrality(g)$vector)
```

```
##       MOTTI      TARKIN       JABBA      GREEDO     RED TEN GOLD LEADER
## 0.009298153 0.011493184 0.011602602 0.011602604 0.015241796 0.017475057
## DARTH VADER       CAMIE     DODONNA       WEDGE        OWEN  RED LEADER
## 0.027009389 0.030744983 0.031723524 0.034374377 0.062695673 0.065141246
##        BERU       BIGGS   GOLD FIVE       R2-D2     OBI-WAN        LEIA
## 0.070824006 0.078921422 0.121485774 0.503053912 0.541729368 0.592624857
##       C-3PO   CHEWBACCA         HAN        LUKE
## 0.595864470 0.657663375 0.812242325 1.000000000
```

Page rank approximates probability that any message will arrive to a particular node. This algorithm was developed by Google founders, and originally applied to website links. Sort the returned *vector* from the `page_rank` of the network and print it out.

```
# sort the `vector` from the `page_rank` of the network and print it out

sort(page_rank(g)$vector)
```

```
##   GOLD FIVE       JABBA      GREEDO     RED TEN       CAMIE     DODONNA
## 0.007092199 0.008310156 0.008310156 0.010573836 0.013792262 0.016185680
##       MOTTI GOLD LEADER        OWEN        BERU       WEDGE      TARKIN
## 0.016813964 0.017945853 0.018881975 0.020368818 0.026377242 0.034180007
## DARTH VADER  RED LEADER       BIGGS     OBI-WAN       R2-D2        LEIA
## 0.034576040 0.034578060 0.035070288 0.067378471 0.068538690 0.086027500
##   CHEWBACCA       C-3PO         HAN        LUKE
## 0.086390090 0.088708430 0.114631333 0.185268949
```

Authority score is another measure of centrality initially applied to the Web. A node has high authority when it is linked by many other nodes that are linking many other nodes. Sort the returned *vector* from the `authority_score` of the network and print it out.

```
# sort the `vector` from the `authority_score` of the network and print it out

sort(authority_score(g)$vector)
```

```
##    GOLD FIVE        MOTTI       TARKIN       GREEDO        JABBA      RED TEN
## 1.273708e-17 9.118469e-03 1.133319e-02 1.154515e-02 1.154515e-02 1.512880e-02
##  GOLD LEADER  DARTH VADER        CAMIE      DODONNA        WEDGE         OWEN
## 1.717615e-02 2.671707e-02 3.064953e-02 3.143121e-02 3.410098e-02 6.256707e-02
##   RED LEADER         BERU        BIGGS        R2-D2      OBI-WAN         LEIA
## 6.476889e-02 7.063977e-02 7.856101e-02 5.030995e-01 5.417666e-01 5.923767e-01
##        C-3PO    CHEWBACCA          HAN         LUKE
## 5.957835e-01 6.577603e-01 8.125507e-01 1.000000e+00
```

Finally, not exactly a measure of centrality, but we can learn more about who each node is connected to by using the following functions: `neighbors` (for direct neighbors) and `ego` (for neighbors up to `n` neighbors away). Find the neighbors of "DARTH VADER". Find his neighbors up to order 2 away.

```
# find the neighbors of "DARTH VADER" using `neighbors` (for direct neighbors)
neighbors(g, v=which(V(g)$name=="DARTH VADER"))
```

```
## + 5/22 vertices, named, from 0a0033b:
## [1] CHEWBACCA LEIA      OBI-WAN   MOTTI     TARKIN
```

```
# find his neighbors up to order 2 away using `ego`
ego(g, order=2, nodes=which(V(g)$name=="DARTH VADER"))
```

```
## [[1]]
## + 14/22 vertices, named, from 0a0033b:
##  [1] DARTH VADER CHEWBACCA   LEIA        OBI-WAN     MOTTI       TARKIN
##  [7] R2-D2       C-3PO       LUKE        HAN         DODONNA     BIGGS
## [13] BERU        RED LEADER
```

4 Network properties

Let's now try to describe what a network looks like as a whole. We can start with measures of the **size** of a network. `diameter` is the length of the longest path (in number of edges) between two nodes. We can use get_diameter to identify this path. mean_distance is the average number of edges between any two nodes in the network. We can find each of these paths between pairs of edges with distances. Find the diameter and mean distances of the network.

```
# find the length of the longest path (`diameter`)
diameter(g, directed=FALSE, weights=NA)
```

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

```
get_diameter(g, directed=FALSE, weights=NA)
```

```
## + 4/22 vertices, named, from 0a0033b:
## [1] DARTH VADER CHEWBACCA   C-3PO       OWEN
```

```
# find the mean distances of the network (`mean_distance`)
mean_distance(g, directed=FALSE)
```

```
## [1] 1.909524
```

```
dist <- distances(g, weights=NA)
dist[1:5, 1:5]
```

```
##             R2-D2 CHEWBACCA C-3PO LUKE DARTH VADER
## R2-D2           0         1     1    1           2
## CHEWBACCA       1         0     1    1           1
## C-3PO           1         1     0    1           2
## LUKE            1         1     1    0           2
## DARTH VADER     2         1     2    2           0
```

edge_density is the proportion of edges in the network over all possible edges that could exist. Find the edge_density of the network.

```
# find the `edge_density`
edge_density(g)
```

```
## [1] 0.2597403
```

```
# (How can we calculate this manually?)
60/((22*21)/2)
```

```
## [1] 0.2597403
```

reciprocity measures the propensity of each edge to be a mutual edge; that is, the probability that if i is connected to j, j is also connected to i. Find the reciprocity of the network – you should find that it is 1. Explain why you think reciprocity=1 in this case.

```
# find the reciprocity of the network
reciprocity(g)
```

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

```
# Why is it 1?

# because there is one person at the center of the whole network - it all revolves around Luke.
```

transitivity, also known as clustering coefficient, measures that probability that adjacent nodes of a network are connected. In other words, if i is connected to j, and j is connected to k, what is the probability that i is also connected to k? Find the transitivity of the network.

```
# find the transitivity of the network

transitivity(g)
```

```
## [1] 0.5375303
```

Extra: Network communities

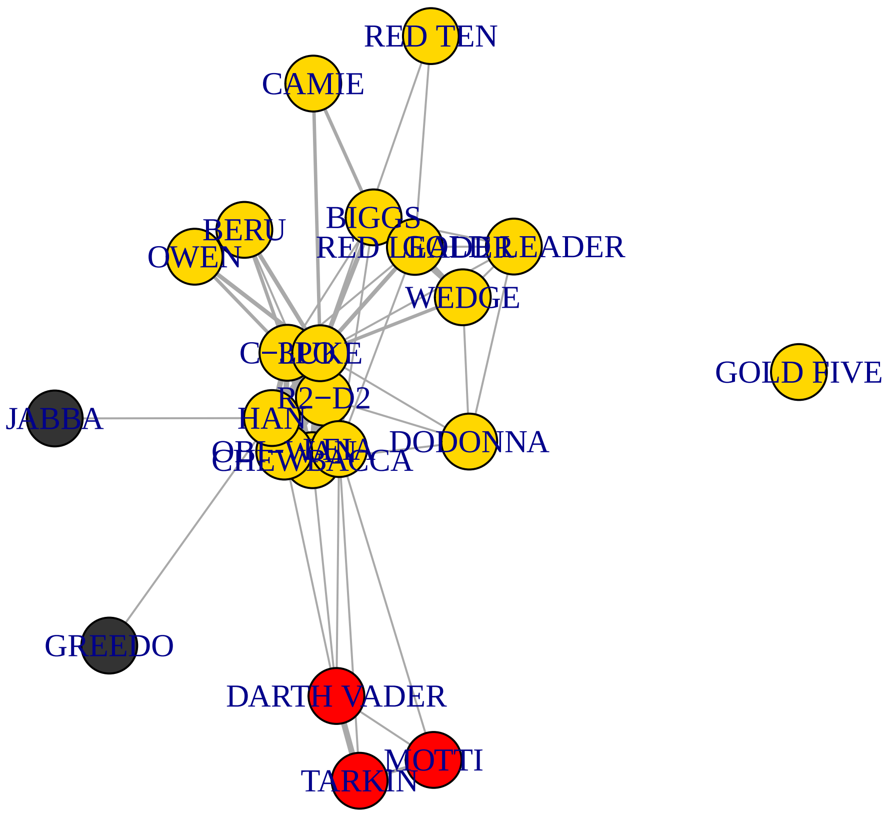
Networks often have different clusters or communities of nodes that are more densely connected to each other than to the rest of the network. Let's cover some of the different existing methods to identify these communities.

The most straightforward way to partition a network is into **connected components**. Each component is a group of nodes that are connected to each other, but *not* to the rest of the nodes. For example, this network has two components.

```
components(g)
```

```
## $membership
##       R2-D2   CHEWBACCA       C-3PO        LUKE DARTH VADER       CAMIE
##           1           1           1           1           1           1
##       BIGGS        LEIA        BERU        OWEN     OBI-WAN       MOTTI
##           1           1           1           1           1           1
##      TARKIN         HAN      GREEDO       JABBA     DODONNA GOLD LEADER
##           1           1           1           1           1           1
##       WEDGE  RED LEADER     RED TEN   GOLD FIVE
##           1           1           1           2
##
## $csize
## [1] 21  1
##
## $no
## [1] 2
```

```
par(mar=c(0,0,0,0)); plot(g)
```



Most networks have a single **giant connected component** that includes most nodes. Most studies of networks actually focus on the giant component (e.g. the shortest path between nodes in a network with two or more component is Inf!).

```
giant <- decompose(g)[[1]]
```

Components can be **weakly connected** (in undirected networks) or ___strongly connected (in directed networks, where there is an edge that ends in every single node of that component).

Even within a giant component, there can be different subsets of the network that are more connected to each other than to the rest of the network. The goal of **community detection algorithms** is to identify these subsets.

There are a few different algorithms, each following a different logic.

The **walktrap** algorithm finds communities through a series of short random walks. The idea is that these random walks tend to stay within the same community. The length of these random walks is 4 edges by default, but you may want to experiment with different values. The goal of this algorithm is to identify the partition that maximizes a modularity score.

```
cluster_walktrap(giant)
```

```
## IGRAPH clustering walktrap, groups: 6, mod: 0.16
## + groups:
##   $`1`
##   [1] "CAMIE"       "BIGGS"       "DODONNA"     "GOLD LEADER" "WEDGE"
##   [6] "RED LEADER"  "RED TEN"
##
##   $`2`
##   [1] "DARTH VADER" "MOTTI"       "TARKIN"
##
```

```
##   $`3`
##   [1] "R2-D2"     "CHEWBACCA" "C-3PO"     "LUKE"      "LEIA"      "OBI-WAN"
##   [7] "HAN"
##   + ... omitted several groups/vertices
```

```r
cluster_walktrap(giant, steps=10)
```

```
## IGRAPH clustering walktrap, groups: 3, mod: 0.15
## + groups:
##   $`1`
##   [1] "DARTH VADER" "MOTTI"       "TARKIN"
##
##   $`2`
##    [1] "R2-D2"     "CHEWBACCA" "C-3PO"     "LUKE"      "LEIA"      "BERU"
##    [7] "OWEN"      "OBI-WAN"   "HAN"       "GREEDO"    "JABBA"
##
##   $`3`
##   [1] "CAMIE"       "BIGGS"       "DODONNA"     "GOLD LEADER" "WEDGE"
##   [6] "RED LEADER"  "RED TEN"
##   + ... omitted several groups/vertices
```

Other methods are:

- The **fast and greedy** method tries to directly optimize this modularity score.
- The **infomap** method attempts to map the flow of information in a network, and the different clusters in which information may get remain for longer periods. Similar to walktrap, but not necessarily maximizing modularity, but rather the so-called "map equation".
- The **edge-betweenness** method iteratively removes edges with high betweenness, with the idea that they are likely to connect different parts of the network. Here betweenness (gatekeeping potential) applies to edges, but the intuition is the same.
- The **label propagation** method labels each node with unique labels, and then updates these labels by choosing the label assigned to the majority of their neighbors, and repeat this iteratively until each node has the most common labels among its neighbors.

```r
cluster_fast_greedy(giant)
```

```
## IGRAPH clustering fast greedy, groups: 4, mod: 0.17
## + groups:
##   $`1`
##   [1] "CHEWBACCA" "LUKE"      "LEIA"      "OBI-WAN"   "HAN"       "GREEDO"
##   [7] "JABBA"
##
##   $`2`
##   [1] "R2-D2" "C-3PO" "BERU"  "OWEN"
##
##   $`3`
##   [1] "CAMIE"       "BIGGS"       "DODONNA"     "GOLD LEADER" "WEDGE"
##   [6] "RED LEADER"  "RED TEN"
##   + ... omitted several groups/vertices
```

```r
cluster_edge_betweenness(giant)
```

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

## Warning in cluster_edge_betweenness(giant): 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
```

```
## IGRAPH clustering edge betweenness, groups: 2, mod: 0.033
## + groups:
##   $`1`
##    [1] "R2-D2"       "CHEWBACCA"   "DARTH VADER" "LEIA"        "OBI-WAN"
##    [6] "MOTTI"       "TARKIN"      "HAN"         "GREEDO"      "JABBA"
##
##   $`2`
##    [1] "C-3PO"       "LUKE"        "CAMIE"       "BIGGS"       "BERU"
##    [6] "OWEN"        "DODONNA"     "GOLD LEADER" "WEDGE"       "RED LEADER"
##   [11] "RED TEN"
##
```

```
cluster_infomap(giant)
```

```
## IGRAPH clustering infomap, groups: 2, mod: 0.064
## + groups:
##   $`1`
##    [1] "R2-D2"       "CHEWBACCA"   "C-3PO"       "LUKE"        "CAMIE"
##    [6] "BIGGS"       "LEIA"        "BERU"        "OWEN"        "OBI-WAN"
##   [11] "HAN"         "GREEDO"      "JABBA"       "DODONNA"     "GOLD LEADER"
##   [16] "WEDGE"       "RED LEADER"  "RED TEN"
##
##   $`2`
##   [1] "DARTH VADER" "MOTTI"       "TARKIN"
##
```

```
cluster_label_prop(giant)
```

```
## IGRAPH clustering label propagation, groups: 2, mod: 0.064
## + groups:
##   $`1`
##    [1] "R2-D2"       "CHEWBACCA"   "C-3PO"       "LUKE"        "CAMIE"
##    [6] "BIGGS"       "LEIA"        "BERU"        "OWEN"        "OBI-WAN"
##   [11] "HAN"         "GREEDO"      "JABBA"       "DODONNA"     "GOLD LEADER"
##   [16] "WEDGE"       "RED LEADER"  "RED TEN"
##
##   $`2`
##   [1] "DARTH VADER" "MOTTI"       "TARKIN"
##
```

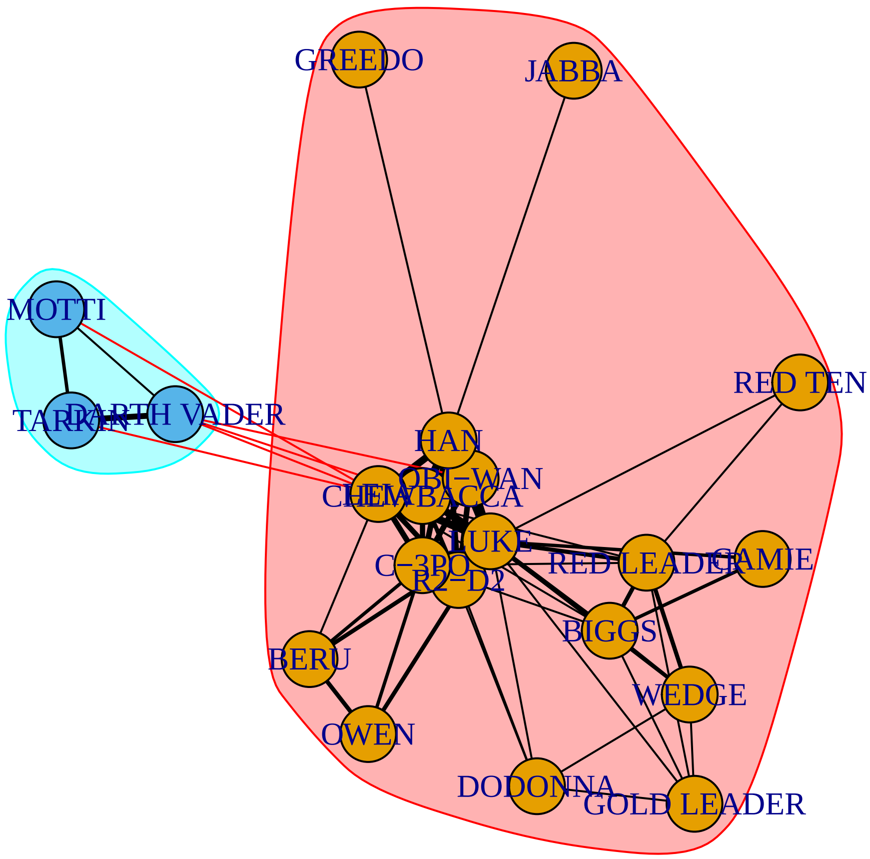
Infomap tends to work better in most social science examples (websites, social media, classrooms, etc), but fastgreedy is faster.

`igraph` also makes it very easy to plot the resulting communities:

```
comm <- cluster_infomap(giant)
modularity(comm) # modularity score
```

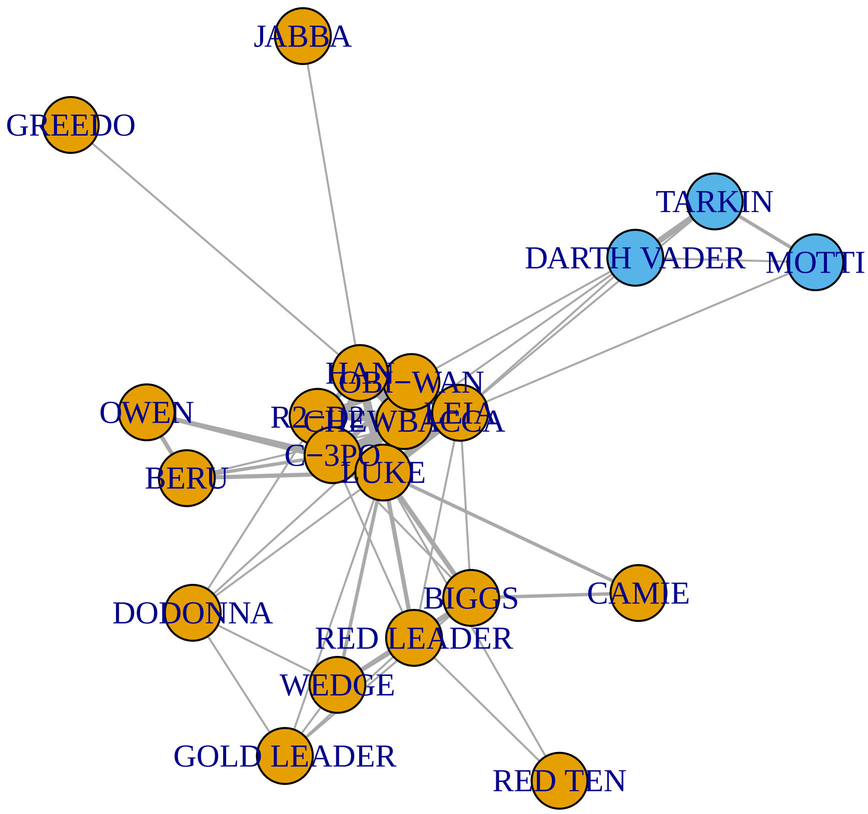
```
## [1] 0.06420569
```

```
par(mar=c(0,0,0,0)); plot(comm, giant)
```



Alternatively, we can also add the membership to different communities as a color parameter in the `igraph` object.

```
V(giant)$color <- membership(comm)
par(mar=c(0,0,0,0)); plot(giant)
```



The final way in which we can think about network communities is in terms of hierarchy or structure. We'll discuss one of these methods.

K-core decomposition allows us to identify the core and the periphery of the network. A k-core is a maximal subnet of a network such that all nodes have at least degree K.

```
coreness(g)
```

```
##       R2-D2   CHEWBACCA       C-3PO        LUKE DARTH VADER       CAMIE
##           6           6           6           6           3           2
##       BIGGS        LEIA        BERU        OWEN     OBI-WAN       MOTTI
##           5           6           3           3           6           3
##      TARKIN         HAN      GREEDO       JABBA     DODONNA GOLD LEADER
##           3           6           1           1           5           5
##       WEDGE  RED LEADER     RED TEN   GOLD FIVE
##           5           5           2           0
```

```
which(coreness(g)==6) # what is the core of the network?
```

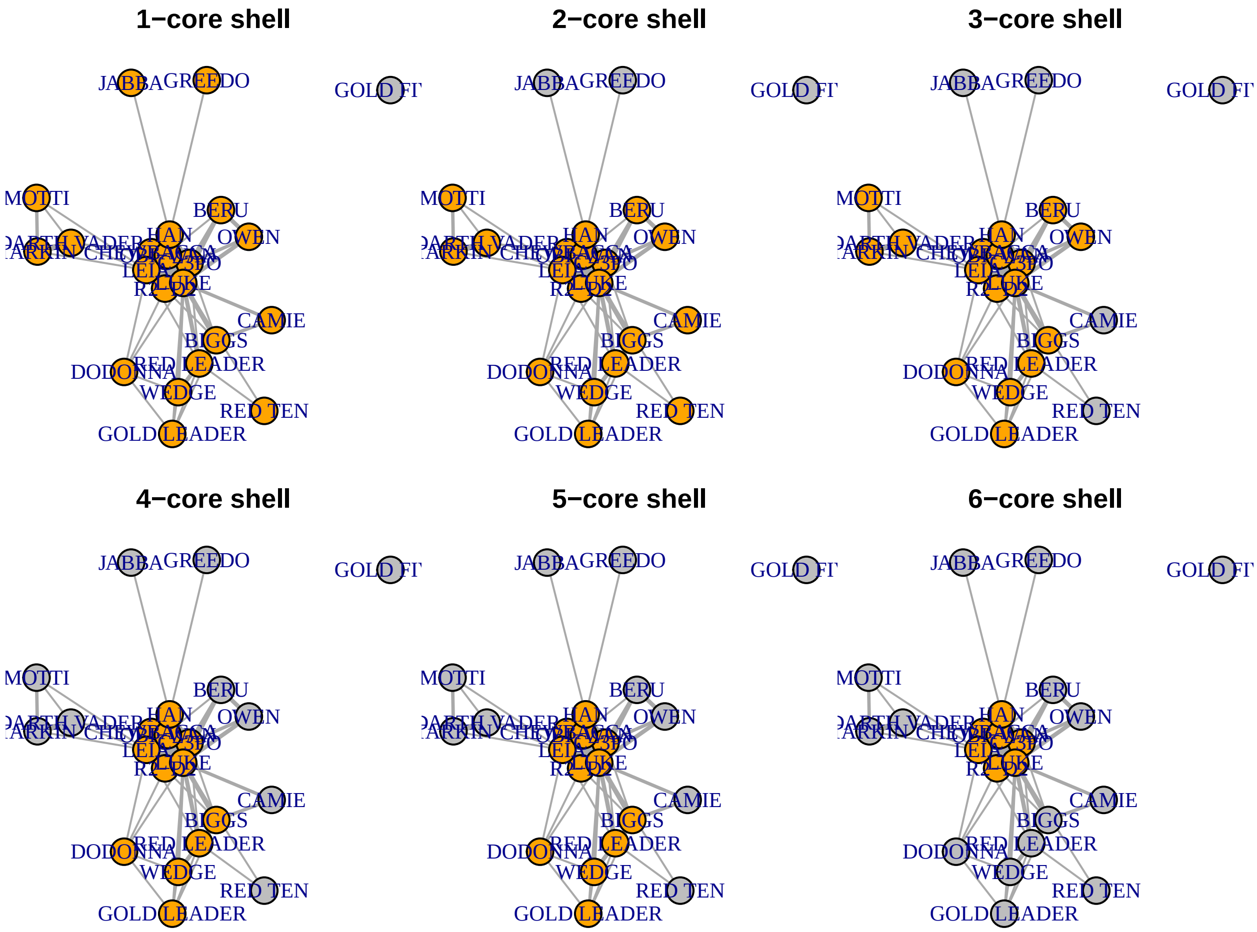
```
##     R2-D2 CHEWBACCA     C-3PO      LUKE      LEIA   OBI-WAN       HAN
##         1         2         3         4         8        11        14
```

```
which(coreness(g)==1) # what is the periphery of the network?
```

```
## GREEDO  JABBA
##     15     16
```

```
# Visualizing network structure
V(g)$coreness <- coreness(g)
par(mfrow=c(2, 3), mar=c(0.1,0.1,1,0.1))
set.seed(777); fr <- layout_with_fr(g)
```

```r
for (k in 1:6){
  V(g)$color <- ifelse(V(g)$coreness>=k, "orange", "grey")
  plot(g, main=paste0(k, '-core shell'), layout=fr)
}
```



5 Social network analysis with `vosonSML`

The `vosonSML` R package is a suite of easy to use functions for collecting and generating different types of networks from social media data. The package supports the collection of data from `twitter`, `youtube` and `reddit`, as well as `hyperlinks` from web sites. Networks in the form of node and edge lists can be generated from collected data, supplemented with additional metadata and used to create graphs for Social Network Analysis.

Install and load the latest release of the package:

```r
library(magrittr) # %>%
```

```
## Warning: package 'magrittr' was built under R version 4.1.2
```

```r
library(vosonSML)
```

The following Reddit example will provide a quick start to using vosonSML functions. Additionally there is an Introduction to vosonSML vignette that is a practical and explanatory guide to collecting data and creating networks.

Collecting Data from Reddit

The `vosonSML` does not require `Reddit` API credentials to be provided when collecting the data, unlike `twitter` and `youtube`.

To collect Reddit comment data, first construct a character vector containing the post URL(s). In the example below, a post relating to the politics around the Australian bushfires was used: https://www.reddit.com/r/worldnews/comments/elcb9b/australias_leaders_deny_link_between_climate/.

```
myThreadUrls <- c("https://www.reddit.com/r/worldnews/comments/elcb9b/australias_leaders_deny_link_betw
```

This post was created on 7th January 2020 and it had attracted over 4000 comments. According to the vignettes of the package, the reddit web endpoint used for collection has maximum limit of 500 comments per thread url, but this limit might have been updated since then.

Collect and explore the data using the codes below.

```
redditData <- Authenticate("reddit") %>%
              Collect(threadUrls = myThreadUrls,
                      writeToFile = FALSE) # whether to write the returned dataframe to file as an .rds
```

```
## Collecting comment threads for reddit urls...
## Waiting between 3 and 10 seconds per thread request.
## Request thread: r/worldnews (elcb9b)
## Continue thread: r/worldnews (elcb9b) - fdhvspn
## Continue thread: r/worldnews (elcb9b) - fdhqght
## Continue thread: r/worldnews (elcb9b) - fdhy26s
## Continue thread: r/worldnews (elcb9b) - fdhtd7v
## Continue thread: r/worldnews (elcb9b) - fdhvm74
## Continue thread: r/worldnews (elcb9b) - fdhgyqk
## Continue thread: r/worldnews (elcb9b) - fdho9wa
## Continue thread: r/worldnews (elcb9b) - fdhs3qw
## Continue thread: r/worldnews (elcb9b) - fdhvi6h
## Continue thread: r/worldnews (elcb9b) - fdi0u9c
## HTML decoding comments.
## thread_id | title                                             | subreddit | count
## --------------------------------------------------------------------------------
## elcb9b    | Australia's leaders deny link between clim... | worldnews | 761
## Collected 761 total comments.
## Done.
```

```
str(redditData)
```

```
## tibble[,23] (S3: tbl_df/tbl/data.frame/datasource/reddit)
##  $ id             : int [1:761] 1 2 3 4 5 6 7 8 9 10 ...
##  $ structure      : chr [1:761] "1" "4_1_1_1_1_1_1_1_1_1" "4_1_1_4_2_1_1_1_1" "4_1_1_4_3_1_1_1_3_
##  $ post_date      : chr [1:761] "2020-01-07 14:34:58" "2020-01-07 14:34:58" "2020-01-07 14:34:58" "2
##  $ post_date_unix : num [1:761] 1.58e+09 1.58e+09 1.58e+09 1.58e+09 1.58e+09 ...
##  $ comm_id        : chr [1:761] "fdhjzyd" "fdhvspn" "fdhqght" "fdhy26s" ...
##  $ comm_date      : chr [1:761] "2020-01-07 19:11:10" "2020-01-07 21:04:05" "2020-01-07 20:15:49" "2
##  $ comm_date_unix : num [1:761] 1.58e+09 1.58e+09 1.58e+09 1.58e+09 1.58e+09 ...
##  $ num_comments   : int [1:761] 4261 4261 4261 4261 4261 4261 4261 4261 4261 4261 ...
##  $ subreddit      : chr [1:761] "worldnews" "worldnews" "worldnews" "worldnews" ...
##  $ upvote_prop    : num [1:761] 0.91 0.91 0.91 0.91 0.91 0.91 0.91 0.91 0.91 0.91 ...
##  $ post_score     : int [1:761] 45731 45727 45729 45725 45731 45730 45728 45728 45736 45728 ...
##  $ author         : chr [1:761] "Gboard2" "Gboard2" "Gboard2" "Gboard2" ...
##  $ user           : chr [1:761] "sandwooder" "[deleted]" "smackofham" "LazerSturgeon" ...
```

```
##  $ comment_score   : int [1:761] 1901 140 16 13 7 9 125 4 5 10 ...
##  $ controversiality: int [1:761] 0 0 0 0 0 0 0 0 0 0 ...
##  $ comment         : chr [1:761] "[And now a report from 2006 predicting it](https://www.tai.org.au/s
##  $ title           : chr [1:761] "Australia’s leaders deny link between climate change and the countr
##  $ post_text       : chr [1:761] "" "" "" "" ...
##  $ link            : chr [1:761] "https://www.theglobeandmail.com/world/article-australias-leaders-un
##  $ domain          : chr [1:761] "theglobeandmail.com" "theglobeandmail.com" "theglobeandmail.com" "t
##  $ url             : chr [1:761] "https://www.reddit.com/r/worldnews/comments/elcb9b/australias_leade
##  $ rm              : logi [1:761] FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ thread_id       : chr [1:761] "elcb9b" "elcb9b" "elcb9b" "elcb9b" ...
```

Creating Reddit Networks

It is currently possible to create two types of networks using Reddit data: (1) actor network and (2) activity network.

Actor Network In the Reddit actor network, nodes represent users who have posted original posts and comments and the edges are the interactions between users in the comments i.e. where there is an edge from user i to user j if i writes a comment that replies to user j's comment (or the original post).

Create a Reddit actor network with comment text as an edge attribute by the following steps:

- Run `Create(data = redditData,type = "actor")` which returns a named list containing two dataframes named "nodes" and "edges".
- Pass the list to `AddText()`, which adds the comment text data to the network dataframe, stored as an edge attribute.
- Pass the list to `Graph()`, which returns an igraph graph object.

```
actorNetwork <- Create(data = redditData,type = "actor") %>% AddText(redditData)
```

```
## Adding text to network...Generating reddit actor network...Done.
## Done.
```

```
actorGraph <- actorNetwork %>% Graph(writeToFile = F)
```

```
## Creating igraph network graph...Done.
```

```
actorGraph
```

```
## IGRAPH 8d7cd0f DN-- 385 762 --
## + attr: type (g/c), name (v/c), user (v/c), label (v/c), subreddit
## | (e/c), thread_id (e/c), comment_id (e/n), comm_id (e/c),
## | vosonTxt_comment (e/c), title (e/c)
## + edges from 8d7cd0f (vertex names):
##  [1] 1 ->385 2 ->2   3 ->99  4 ->105 5 ->105 6 ->2   2 ->172 7 ->16  8 ->16
## [10] 9 ->2   2 ->2   10->1   11->2   12->3   13->4   14->5   2 ->2   15->7
## [19] 16->8   17->9   2 ->2   18->10  2 ->11  2 ->3   2 ->2   19->18  20->2
## [28] 3 ->2   21->2   22->1   23->2   2 ->21  1 ->22  24->2   25->2   19->1
## [37] 26->2   27->25  28->1   2 ->26  29->2   19->28  30->2   2 ->29  31->1
## [46] 32->2   33->2   34->1   35->2   36->2   37->1   2 ->35  38->2   19->1
## + ... omitted several edges
```

The Reddit actor network contains a graph attribute `type` (set to "reddit").

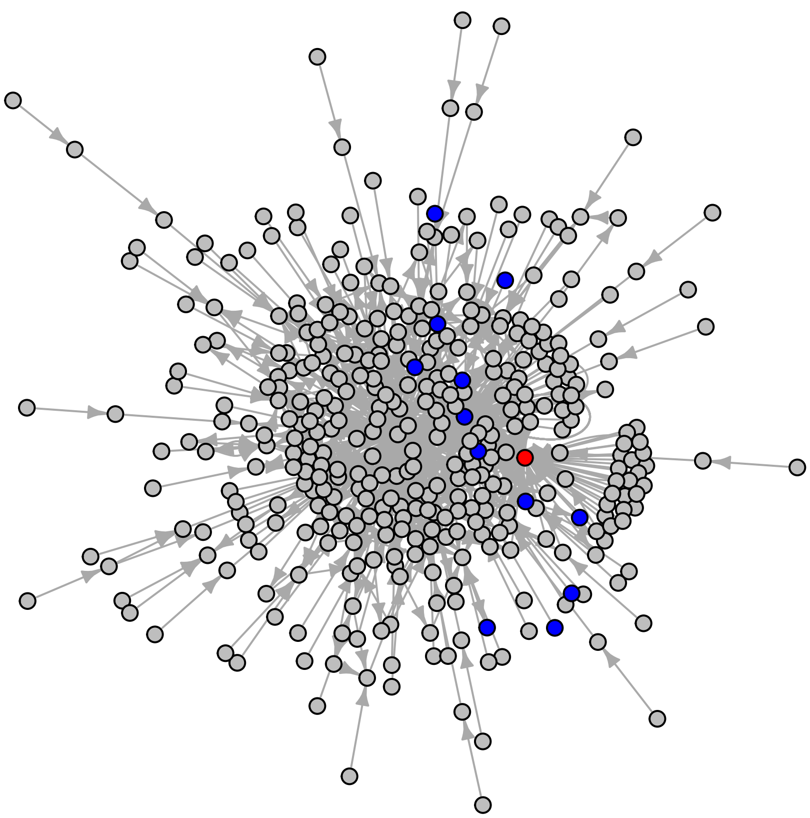
The node attributes are: - `name` (sequential ID number for actor, generated by `vosonSML`), - `user` (Reddit handle or screen name)) and - `label` (a concatenation of the ID and screen name).

The edge attributes are: - `subreddit` (the subreddit from which the post is collected), - `thread_id` (the 6 character ID of the thread or post), - `comment_id` (sequential ID number for comment, generated by `vosonSML`). - There is also an edge attribute `title`, which is set to `NA` for all comments except the comment representing the original post. Further note that the original post is represented as a self-loop edge from the user who authored the post (and this is how the post text can be accessed, as an edge attribute). - Finally, because we used `AddText()` in the above example, there is also an edge attribute `vosonTxt_comment` which is the text associated with the comment, or original post.

How many nodes and edges are there in the example Reddit actor network?

Use the following codes for a visualization of the actor network. Note that the author of the post is indicated by a red node, and blue nodes indicate those people who mentioned "arson" or "starting fires" in at least one of their comments.

```
# set node color of original post to red based on presence of title edge attribute
# set other node colors to grey
V(actorGraph)$color <- "grey"
V(actorGraph)$color[tail_of(actorGraph, which(!is.na(E(actorGraph)$title)))] <- "red"

# get node indexes for the tails of edges that have comments containing words of interest: e.g., "arson
# set their node colors to blue
ind <- tail_of(actorGraph, grep("arson|starting fires", tolower(E(actorGraph)$vosonTxt_comment)))
V(actorGraph)$color[ind] <- "blue"

par(mfrow=c(1,1), mar=c(0,0,0,0))
plot(actorGraph, vertex.label = "", vertex.size = 4, edge.arrow.size = 0.5)
```



Activity Network In the Reddit activity network, nodes are either comments and/or initial thread posts and the edges represent replies to the original post, or replies to comments. Create a Reddit activity network as before with `type = "activity"`.

```
activityNetwork <- Create(data = redditData,type = "activity") %>% AddText(redditData)
```

```
## Adding text to network...Generating reddit activity network...
## -------------------------
## collected reddit comments | 761
## threads                   | 1
## comments                  | 761
## nodes                     | 762
## edges                     | 761
## -------------------------
## Done.
## Done.
```

```
activityGraph <- activityNetwork %>% Graph(writeToFile = F)
```

```
## Creating igraph network graph...Done.
```

```
activityGraph
```

```
## IGRAPH 980f432 DN-- 762 761 --
## + attr: type (g/c), name (v/c), thread_id (v/c), comm_id (v/c),
## | datetime (v/c), ts (v/n), subreddit (v/c), user (v/c), node_type
## | (v/c), vosonTxt_comment (v/c), title (v/c), label (v/c), edge_type
## | (e/c)
## + edges from 980f432 (vertex names):
## [1] elcb9b.1                  ->elcb9b.0
## [2] elcb9b.4_1_1_1_1_1_1_1_1_1->elcb9b.4_1_1_1_1_1_1_1_1
## [3] elcb9b.4_1_1_4_2_1_1_1_1_1->elcb9b.4_1_1_4_2_1_1_1_1
## [4] elcb9b.4_1_1_4_3_1_1_1_3_1->elcb9b.4_1_1_4_3_1_1_1_3
## [5] elcb9b.4_1_1_4_3_1_1_1_3_2->elcb9b.4_1_1_4_3_1_1_1_3
## + ... omitted several edges
```

The Reddit activity network contains a graph attribute `type` (set to "reddit").

The node attributes are: - `name` (string showing position of the comment in the thread), - `date` (date when the comment was authored, in DD-MM-YY format), - `subreddit` (the subreddit from which the post is collected), - `user` (Reddit handle or screen name of the user who authored the comment or post), - `node_type` ('comment' or 'thread'), - `title` (`NA` for all nodes except that representing the original post), - `label` (a concatenation of name and user). - Because we used `AddText()` in the above example, there is also a node attribute `vosonTxt_comment` which is the text from the comment, or original post.

The edge attribute contains `edge_type` which is 'comment' for all edges.

How many nodes and edges are there in the example Reddit activity network?

Use the following codes for a visualization of the actor network.

```
# set original post node colors to red based on a node type of thread
# set other node colors to grey
V(activityGraph)$color <- "grey"
V(activityGraph)$color[which(V(activityGraph)$node_type == "thread")] <- "red"

# get node indexes for nodes that have comment attributes containing words of interest
# set their node colors to blue
ind <- grep("arson|starting fires", tolower(V(activityGraph)$vosonTxt_comment))
V(activityGraph)$color[ind] <- "blue"
```

```r
plot(activityGraph, vertex.label = "", vertex.size = 4, edge.arrow.size = 0.5)
```