Social Network Analysis: Lab 1

José Luis Estévez; contact: jose.luis.estevez@liu.se

January 26th, 2024

0) Lab Contents

1) R software and R language
2) Basic procedures in R
3) R packages for social network analysis
4) The `sna` and `igraph` packages

---

1) R software & R language

Installing **R** (step 1) and **RStudio** (step 2) https://posit.co/download/rstudio-desktop/

Advantages of R (or other script-based software) over button-pressing software (e.g., SPSS)

- Reproducibity (more journals request replication package, see: https://pubsonline.informs.org/page/mnsc/datapolicy)
- Better understanding of procedures
- Sharing (with collaborators, colleagues, or the scientific community as a whole)
- Extensions (install.packages(" "))
- No economic cost (licenses)

For students unfamiliar with programming, I recommend this book:

- https://www.oreilly.com/library/view/r-for-data/9781491910382/

Also, online courses (some of them are free or partly free)

- https://www.datacamp.com/courses/free-introduction-to-r
- https://www.udemy.com/course/r-basics/
- https://www.edx.org/learn/r-programming
- https://www.coursera.org/learn/r-programming
- https://www.codecademy.com/learn/learn-r
- https://alison.com/course/introduction-to-r
- https://www.codespaces.com/best-r-programming-certifications-courses-trainings.html

---

2) Basic procedures in R

The R console works as a calculator

For instance, try running

```
6+8; 5-7; 5*12; (5+10)/2
```

```
## [1] 14
```

```
## [1] -2
```

```
## [1] 60
```

```
## [1] 7.5
```

You can also ask for powers, square roots

```
2^5; sqrt(9); 27^(1/3)
```

```
## [1] 32
```

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

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

Al well logarithms, trigonometric functions, etc.

```
log(1000,base=10); log(exp(9)); cos(2*pi)
```

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

```
## [1] 9
```

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

Rather than numbers, more often you will use (`<-` or `->`) to store your values into objects

```
x <- 5
4 -> y
z <- x*y
print(z)
```

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

Getting familiar with functions

All the above (`sqrt`, `log`, `cos`, etc.) are FUNCTIONS

- INPUT –> function –> OUTPUT

Inputs can be one number or element (as above), but also a vector, a matrix, an array, etc.

```r
sum(c(1,2,3,4)); prod(c(1,2,3,4))
```

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

```
## [1] 24
```

As with the input, the output can be formed by more than one element

```r
cumsum(c(1,2,3,4)); cumprod(c(1,2,3,4))
```

```
## [1]  1  3  6 10
```

```
## [1]  1  2  6 24
```

Notice that most functions you will use have additional arguments

```r
log(16); log(16,base=2)
```

```
## [1] 2.772589
```

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

To learn more about the arguments of a certain function, use `?` or `help()`

```r
# ?log()
# help(sum)
```

Vectors, data frames, and matrices

A **vector** is collection of elements (often of the same type)

```r
v <- c(0,1,2,3,4,5)
v
```

```
## [1] 0 1 2 3 4 5
```

Operations involving a *scalar* and a *vector*, apply the operation to all the elements of the vector

```r
v+1 # scalars are applied to all elements of a vector
```

```
## [1] 1 2 3 4 5 6
```

```r
v*rev(v) # in vectors of the same length, operations are performed element to element
```

```
## [1] 0 4 6 6 4 0
```

```r
v*c(1,2) # otherwise, R recycles the information in the shorter vector
```

```
## [1]  0  2  2  6  4 10
```

As data users, we are accustomed to see vectors as columns in a **data frame** (or *data table*)

```r
data <- mtcars
class(data)
```

```
## [1] "data.frame"
```

```r
#View(data)
head(data,8) # show the first eight rows of the data frame
```

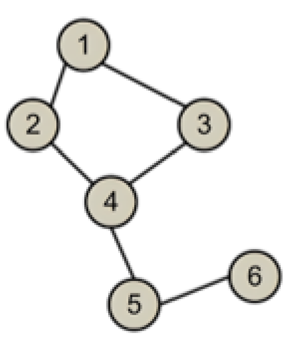
```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
```

A data frame is a collection of vectors

```r
data$mpg # The first column is a vector
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

For *Social network analysis*, however, it is important to get some fluidity with **matrices**



	1	2	3	4	5	6
1	0	1	1	0	0	0
2	1	0	0	1	0	0
3	1	0	0	1	0	0
4	0	1	1	0	1	0
5	0	0	0	1	0	1
6	0	0	0	0	1	0

Figure 1: Network graph and adjacency matrix

In essence, whereas a vector is a uni-dimensional array, a matrix is two-dimensional array.

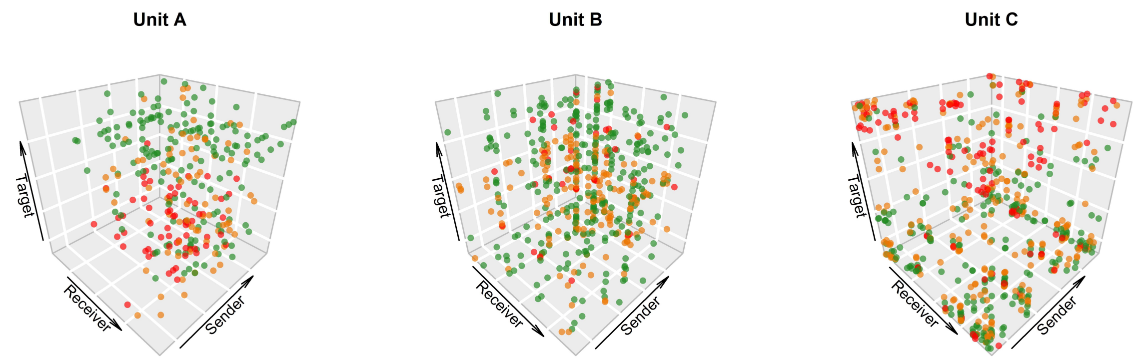


Figure 2: Gossip cube (source: Estévez, Wittek, et al., 2022, Social Network Analysis & Mining)

A cube is a three three-dimensional array (but we need not get there)

Although we are not using real data in this first lab, we can create some just to see how it looks. This is simply the network displayed in Figure 1:

```
mtx <- matrix(c(0,1,1,0,0,0,
                1,0,0,1,0,0,
                1,0,0,1,0,0,
                0,1,1,0,1,0,
                0,0,0,1,0,1,
                0,0,0,0,1,0),
              nrow=6,ncol=6,byrow=TRUE)
mtx
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    1    1    0    0    0
## [2,]    1    0    0    1    0    0
## [3,]    1    0    0    1    0    0
## [4,]    0    1    1    0    1    0
## [5,]    0    0    0    1    0    1
## [6,]    0    0    0    0    1    0
```

Notice that, when you ask for the class of the object, R replies it is a matrix and an array. You can also check the dimensions (number of rows and columns)

```
class(mtx); dim(mtx)
```

```
## [1] "matrix" "array"
```

```
## [1] 6 6
```

To make these network a bit less aseptic, let's name the nodes

```
nodenames <- c('Amy','Bjorn','Carl','Dan','Ebba','Frank')
dimnames(mtx) <- list(nodenames,nodenames)
mtx
```

```
##       Amy Bjorn Carl Dan Ebba Frank
## Amy     0     1    1   0    0     0
## Bjorn   1     0    0   1    0     0
## Carl    1     0    0   1    0     0
## Dan     0     1    1   0    1     0
## Ebba    0     0    0   1    0     1
## Frank   0     0    0   0    1     0
```

By the way, elements of the matrix can be selected using their position (or the actual names, if available), similar to vectors…

```
# Does Dan has a tie with Ebba? Let us ask the matrix
mtx['Dan','Ebba']
```

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

This means, you can also use this to correct information.

Say you know for sure that Dan and Ebba cannot connected. So let's severe the tie

```
mtx['Dan','Ebba'] <- mtx['Ebba','Dan'] <- 0
mtx
```

```
##       Amy Bjorn Carl Dan Ebba Frank
## Amy     0     1    1   0    0     0
## Bjorn   1     0    0   1    0     0
## Carl    1     0    0   1    0     0
## Dan     0     1    1   0    0     0
## Ebba    0     0    0   0    0     1
## Frank   0     0    0   0    1     0
```

- **Question**: *Notice that I changed two cells in the matrix. Do you understand why?*

In social network analysis, one important function when operating matrices is the *diagonal*: the elements in positions (1,1), (2,2), (3,3),… (n,n)

```
diag(mtx)
```

```
##   Amy Bjorn  Carl   Dan  Ebba Frank
##     0     0     0     0     0     0
```

Often, you would send the values in the diagonal to NA.

```
diag(mtx) <- NA # Often sent to missing data
mtx
```

```
##       Amy Bjorn Carl Dan Ebba Frank
## Amy    NA     1    1   0    0     0
## Bjorn   1    NA    0   1    0     0
## Carl    1     0   NA   1    0     0
## Dan     0     1    1  NA    0     0
## Ebba    0     0    0   0   NA     1
## Frank   0     0    0   0    1    NA
```

- **Question**: *Can you explain the logic behind the procedure just described?*

One last thing, although matrices are the most natural way of operating with network data, when the number of nodes is large, they become very heavy (often, you are saving a lot of zeroes). In these cases, instead of matrix, what researchers use is an **edge list**.

The transformation can be done in several ways (next week, we will use some packages for doing this more efficiently), for now let's just focus on the output

```
library(data.table);library(reshape2)
edgelist <- as.data.table(melt(mtx))
colnames(edgelist) <- c('node1','node2','tie')
edgelist <- edgelist[!is.na(tie) & tie == 1]
edgelist
```

```
##     node1 node2 tie
##  1: Bjorn   Amy   1
##  2:  Carl   Amy   1
##  3:   Amy Bjorn   1
##  4:   Dan Bjorn   1
##  5:   Amy  Carl   1
##  6:   Dan  Carl   1
##  7: Bjorn   Dan   1
##  8:  Carl   Dan   1
##  9: Frank  Ebba   1
## 10:  Ebba Frank   1
```

Since the ties are undirected, we can simplify further.

```
edgelist[,dyad := apply(edgelist[,.(node1,node2)],1,
               function(row) paste(sort(row),collapse = '-'))]
edgelist <- edgelist[!duplicated(dyad)]
edgelist
```

```
##    node1 node2 tie       dyad
## 1: Bjorn   Amy   1  Amy-Bjorn
## 2:  Carl   Amy   1   Amy-Carl
## 3:   Dan Bjorn   1  Bjorn-Dan
## 4:   Dan  Carl   1   Carl-Dan
## 5: Frank  Ebba   1 Ebba-Frank
```

---

3) R packages for social network analysis

There are plenty of packages developed to perform SNA in R. Among these,

- `sna`, `network` (all the `statnet` family; see: https://statnet.org/packages/)
- `igraph`
- `ggraph` (for visualisations; it builds on `ggplot2`)

- `intergraph` (if you need to convert between network data objects: matrix, edgelist, etc.)
- `RSiena` (for Stochastic Actor-Oriented Models or SAOMs, covered later in this course)
- If you want to start playing around with real (or fictional) data, `networkdata` contains a large sample of networks in igraph format (to install networkdata, see: https://github.com/schochastics/networkdata)

To install most of these packages, just use the function `install.packages`. For instance:

```r
# install.packages(c('sna','igraph'))
```

And to use the function therein contained, do not forget to load the packages in your R session

```r
library(sna)
```

```
## Loading required package: statnet.common

##
## Attaching package: 'statnet.common'

## The following objects are masked from 'package:base':
##
##     attr, order

## Loading required package: network

##
## 'network' 1.18.1 (2023-01-24), part of the Statnet Project
## * 'news(package="network")' for changes since last version
## * 'citation("network")' for citation information
## * 'https://statnet.org' for help, support, and other information

## sna: Tools for Social Network Analysis
## Version 2.7-1 created on 2023-01-24.
## copyright (c) 2005, Carter T. Butts, University of California-Irvine
##  For citation information, type citation("sna").
##  Type help(package="sna") to get started.
```

```r
library(igraph)
```

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

## The following objects are masked from 'package:sna':
##
##     betweenness, bonpow, closeness, components, degree, dyad.census,
##     evcent, hierarchy, is.connected, neighborhood, triad.census

## The following objects are masked from 'package:network':
##
##     %c%, %s%, add.edges, add.vertices, delete.edges, delete.vertices,
##     get.edge.attribute, get.edges, get.vertex.attribute, is.bipartite,
##     is.directed, list.edge.attributes, list.vertex.attributes,
##     set.edge.attribute, set.vertex.attribute
```

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

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

Notice that the last package loaded can mask functions in another package. Because of this, it is sometimes convenient to detach a package (`detach("package:sna")`), or call the function with reference to the package included

```
# sna::betweenness()
# igraph::betweenness()
```

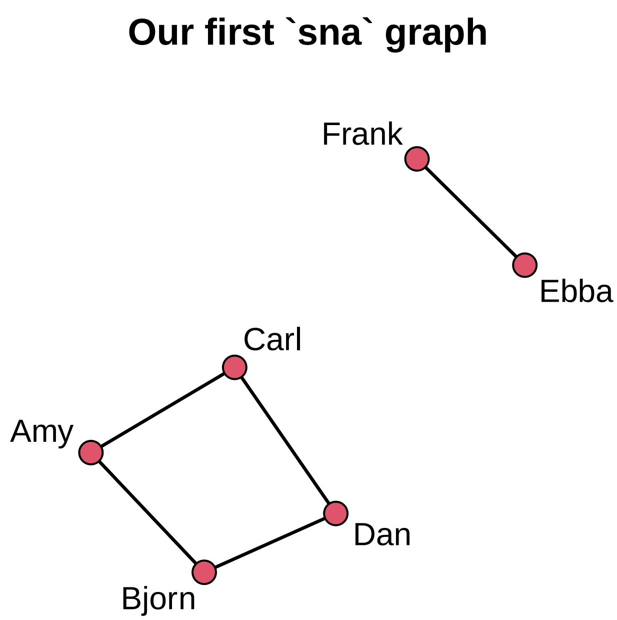
---

4) The `sna` and `igraph` packages

4.1) `sna` (https://cran.r-project.org/web/packages/sna/sna.pdf)

The function `gplot` enables easy visualization

```
gplot(mtx,
      displaylabels=TRUE, # show the names
      usearrows = FALSE, # don't show arrowhead (undirected network)
      main ='Our first `sna` graph')
```



With `sna`, you can obtain descriptive statistics.

```
sna::degree(mtx,cmode='indegree') # In/out degrees
```

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

```
isolates(mtx) # Whether there are any isolates
```

```
## integer(0)
```

The density, reciprocity, and transitivity of a network

```
gden(mtx) # Density (Number of actual ties over Number of potential ties)
```

```
## [1] 0.3333333
```

```
grecip(mtx,measure='edgewise') # Reciprocity (pointless for an undirected network)
```

```
## Mut
##   1
```

```r
# Don't forget to add 'edgewise', otherwise mutual zeroes will be counted as reciprocal too
gtrans(mtx,measure='weak') # Transitivity
```

```
## [1] 0
```

```r
# Note that the weak form is the most commonly used
```

Also, dyadic and triadic configurations

```r
sna::dyad.census(mtx)
```

```
##      Mut Asym Null
## [1,]   5    0   10
```

```r
sna::triad.census(mtx)
```

```
##      003 012 102 021D 021U 021C 111D 111U 030T 030C 201 120D 120U 120C 210 300
## [1,]   4   0  12    0    0    0    0    0    0    0   4    0    0    0   0   0
```

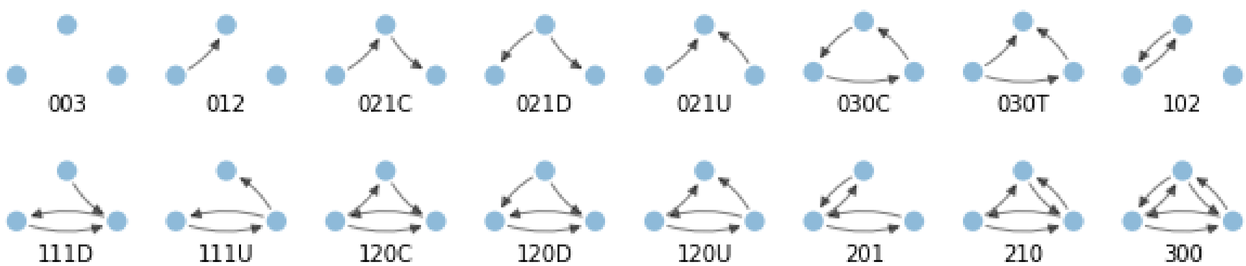
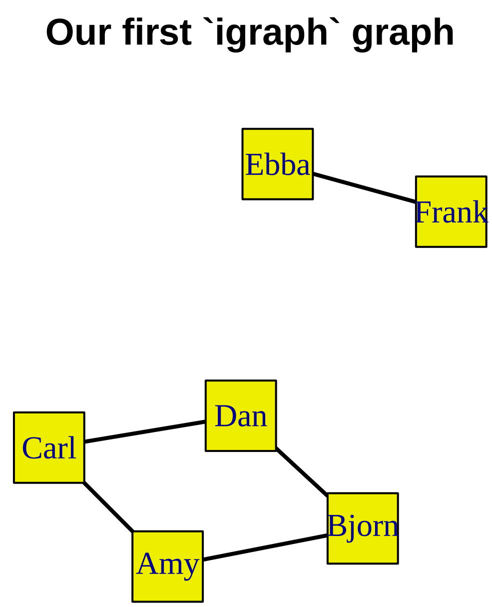


Figure 3: Triadic motifs

4.2) igraph (https://igraph.org/r/pdf/latest/igraph.pdf)

Notice that, unlike `sna` which works with matrices as inputs, `igraph` requires its own object

```r
#plot.igraph(mtx)
grph <- graph.adjacency(mtx,mode='undirected')

plot.igraph(grph,
            vertex.shape='square',
            vertex.size=35,
            vertex.color='yellow2',
            edge.color='black',
            edge.width=2,
            main='Our first `igraph` graph')
```



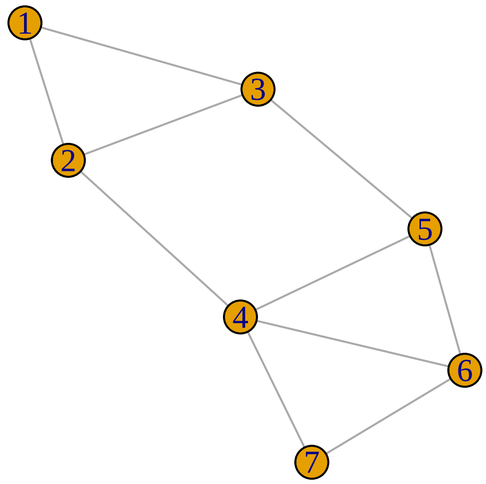
You can customize visualizations further, but we will see this in future labs

Edge lists can be easily inputted as follows

```
undirect.g <- graph.formula(1-2, 1-3, 2-3, 2-4, 3-5, 4-5, 4-6,4-7, 5-6, 6-7)
undirect.g
```

```
## IGRAPH 94736fa UN-- 7 10 --
## + attr: name (v/c)
## + edges from 94736fa (vertex names):
##  [1] 1--2 1--3 2--3 2--4 3--5 4--5 4--6 4--7 5--6 6--7
```

```
plot(undirect.g)
```

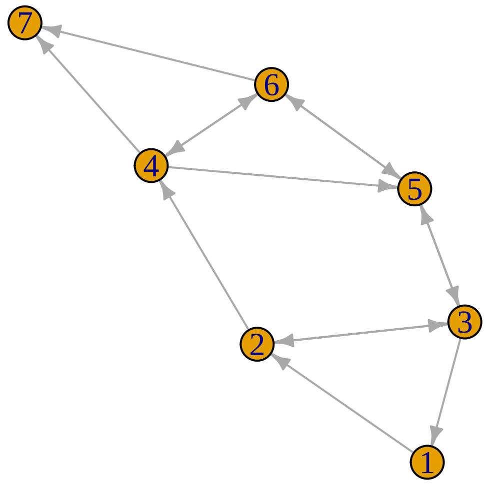


For directed networks, we need the addition symbol (-+, +-, or ++)

```
direct.g <- graph.formula(1-+2, 1+-3, 2++3, 2-+4, 3++5, 4-+5, 4++6,4-+7, 5++6, 6-+7)
direct.g
```

```
## IGRAPH 9476348 DN-- 7 14 --
## + attr: name (v/c)
## + edges from 9476348 (vertex names):
##  [1] 1->2 2->3 2->4 3->1 3->2 3->5 4->5 4->6 4->7 5->3 5->6 6->4 6->5 6->7
```

```
plot(direct.g,
     edge.arrow.size=.5)
```



`V()` and `E()` retrieve the nodes (vectors) and ties (edges), respectively

```
V(direct.g)
```

```
## + 7/7 vertices, named, from 9476348:
## [1] 1 2 3 4 5 6 7
```

```
E(direct.g)
```

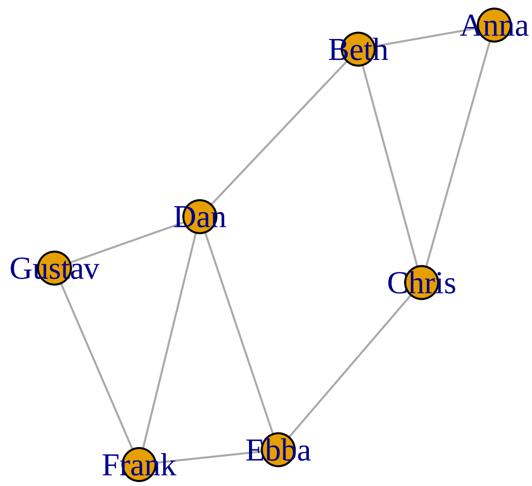
```
## + 14/14 edges from 9476348 (vertex names):
##  [1] 1->2 2->3 2->4 3->1 3->2 3->5 4->5 4->6 4->7 5->3 5->6 6->4 6->5 6->7
```

We can rename the nodes

```
V(undirect.g)$name <- c('Anna','Beth','Chris','Dan','Ebba','Frank','Gustav')
undirect.g
```

```
## IGRAPH 94736fa UN-- 7 10 --
## + attr: name (v/c)
## + edges from 94736fa (vertex names):
##  [1] Anna --Beth   Anna --Chris  Beth --Chris  Beth --Dan    Chris--Ebba
##  [6] Dan  --Ebba   Dan  --Frank  Dan  --Gustav Ebba --Frank  Frank--Gustav
```

```r
plot(undirect.g)
```

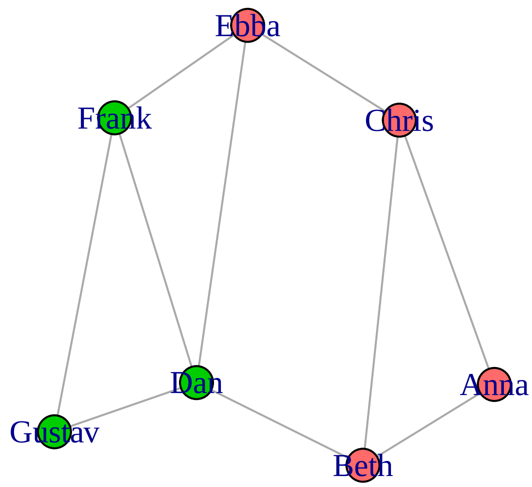


And add additional attributes

```r
V(undirect.g)$gender <- c('w','w','w','m','w','m','m')
undirect.g
```

```
## IGRAPH 94736fa UN-- 7 10 --
## + attr: name (v/c), gender (v/c)
## + edges from 94736fa (vertex names):
##  [1] Anna --Beth   Anna --Chris  Beth --Chris  Beth --Dan    Chris--Ebba
##  [6] Dan  --Ebba   Dan  --Frank  Dan  --Gustav Ebba --Frank  Frank--Gustav
```

```r
plot(undirect.g,
     vertex.color=ifelse(V(undirect.g)$gender == "m","green3","indianred1"))
```



Attributes of the ties are also possible

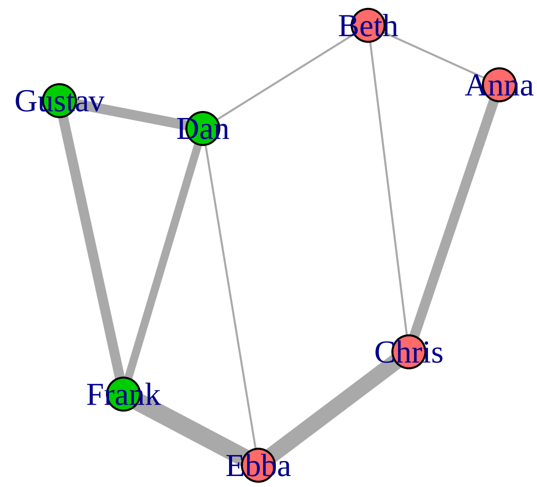
```r
ecount(undirect.g) # remember it is 10 ties we have
```

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

```r
E(undirect.g)$strength <- c(1,5,1,1,8,1,4,5,9,5)
undirect.g
```

```
## IGRAPH 94736fa UN-- 7 10 --
## + attr: name (v/c), gender (v/c), strength (e/n)
## + edges from 94736fa (vertex names):
##  [1] Anna --Beth   Anna --Chris  Beth --Chris  Beth --Dan    Chris--Ebba
##  [6] Dan  --Ebba   Dan  --Frank  Dan  --Gustav Ebba --Frank  Frank--Gustav
```

```r
plot(undirect.g,
     vertex.color=ifelse(V(undirect.g)$gender == "m","green3","indianred1"),
     edge.width=E(undirect.g)$strength)
```



As with `sna`, you can get network descriptives

```
edge_density(undirect.g)
```

```
## [1] 0.4761905
```

```
reciprocity(undirect.g)
```

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

```
transitivity(undirect.g)
```

```
## [1] 0.45
```

And, since we have some node attributes (in our case, gender), we can measure homophily too

```
# Use the following code to install the "isnar" package
#install.packages("remotes")
#remotes::install_github("mbojan/isnar")
isnar::ei(undirect.g,'gender')
```

```
## [1] -0.4
```

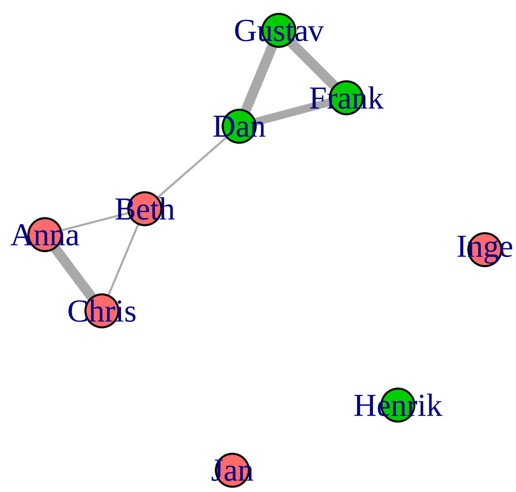
```r
isnar::assort(undirect.g,'gender')
```

```
## [1] 0.3939394
```

```r
# Note that, when using the Krackhardt's E/I ratio, -1 means perfect homophily, +1 means perfect hetero
```

Addition and subtraction of nodes/edges

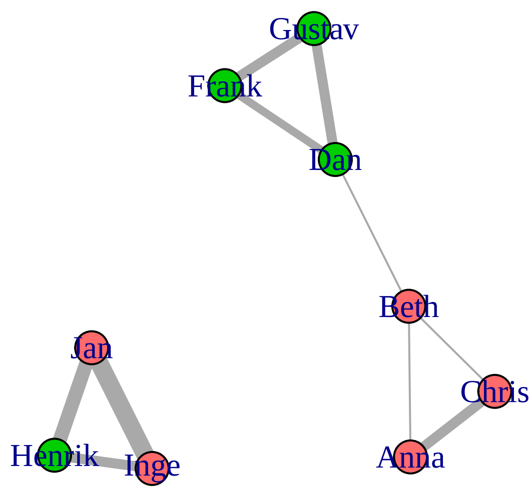
```r
undirect.g <- add.vertices(undirect.g,
                           3, # Three new nodes
                           attr=list(name=c('Henrik','Inge','Jan'), # names
                                     gender=c('m','w','w'))) # gender

undirect.g <- delete.vertices(undirect.g,'Ebba')

plot(undirect.g,
     vertex.color=ifelse(V(undirect.g)$gender == "m","green3","indianred1"),
     edge.width=E(undirect.g)$strength)
```



Let's make the three new nodes form a triad

```r
undirect.g <- add.edges(undirect.g,
                        c('Henrik','Inge','Henrik','Jan','Inge','Jan'),
                        attr=list(strength=c(5,7,9))) # tie strenth
```

```r
plot(undirect.g,
     vertex.color=ifelse(V(undirect.g)$gender == "m","green3","indianred1"),
     edge.width=E(undirect.g)$strength)
```



Now, the new network comprises two components

```r
components(undirect.g)
```

```
## $membership
##   Anna   Beth  Chris    Dan  Frank Gustav Henrik   Inge    Jan
##      1      1      1      1      1      1      2      2      2
##
## $csize
## [1] 6 3
##
## $no
## [1] 2
```

To conclude this first lab, let's turn the `object` into an edge list and matrix

```r
# Retrieve edge list and matrix
get.edgelist(undirect.g)
```

```
##       [,1]     [,2]
```

```
##  [1,] "Anna"   "Beth"
##  [2,] "Anna"   "Chris"
##  [3,] "Beth"   "Chris"
##  [4,] "Beth"   "Dan"
##  [5,] "Dan"    "Frank"
##  [6,] "Dan"    "Gustav"
##  [7,] "Frank"  "Gustav"
##  [8,] "Henrik" "Inge"
##  [9,] "Henrik" "Jan"
## [10,] "Inge"   "Jan"
```

```
as.matrix(get.adjacency(undirect.g))
```

```
##        Anna Beth Chris Dan Frank Gustav Henrik Inge Jan
## Anna      0    1     1   0     0      0      0    0   0
## Beth      1    0     1   1     0      0      0    0   0
## Chris     1    1     0   0     0      0      0    0   0
## Dan       0    1     0   0     1      1      0    0   0
## Frank     0    0     0   1     0      1      0    0   0
## Gustav    0    0     0   1     1      0      0    0   0
## Henrik    0    0     0   0     0      0      0    1   1
## Inge      0    0     0   0     0      0      1    0   1
## Jan       0    0     0   0     0      0      1    1   0
```

Voluntary homework

- We talked about the meaningless of the diagonal in SNA. However, could you think of an example in which the diagonal can have a substantive meaning?
- Try to familiarize yourself with the `sna` and `igraph` packages, especially try to understand the additional arguments in functions like `grecip` (edgewise), `gtrans` (measure), etc.
- Look for information about Krackhardt's E/I index or ratio. How does it compare with Yules' Q? What are the weaknessess of these measures?