National Research University Higher School of Economics

Faculty of Social Sciences

Social Network Analysis

Seminar 6 and HW3: Community Detection

Contents

Welcome back!

Today we are getting immersed in yet another purely network type of models - community detection. While communities are critically important to human mere survival, no methods, other than network, can offer models of finding and studying these communities. Yes, there is cluster analysis and some geographical data approaches, but they do not take into account connections between people. Networks do. And today, you will be able to learn how to work with data to find such communities.

Learning objectives

The goal is to discuss various approaches to community detection and look at a network-level method of doing so: blockmodels.

Upon completion of the exercises in this seminar you should be able to do the following:

1. Build community detection models in R
2. Understand the various approaches to blockmodeling method of community detection

Data

All the necessary data files are provided with the lecture. We are using the same data we've used before for models of social influence, plus a couple of files with full matrices for more complex blockmodeling. You can either use the NetData package, or unzip and load the files I refer to in the seminar - they are included.

Required packages

For Seminar 6, the following packages are required:

1. sna
2. network
3. foreign
4. igraph
5. intergraph
6. NetCluster
7. dendextend

The only packages that are new are the *NetCluster* and *dendextend*; please install it before starting the segment.

Seminar 6 Submission and Homework 3

All previous submission rules apply, the deadline for the seminar and HW is posted.

Methodological digression: working with loops

Please make sure your working directory is set to a folder where the data files are located; otherwise, you will be forced to provide a full path to data every time.

Extracting networks from data frames

For this assignment, you will need the *NetData* package, but unfortunately, it was removed from CRAN. So if you can find it, great. If not, I have created an archive with all the necessary files - I will load it into your folder as well.

If you do have NetData, great. This package contains a number of datasets, which can be used for labs and practice - if you are looking for more data to practice on your own, check this package out (using the "?NetData" command in Console). We are using the kracknets dataset, which, actually, is the same as the HiTech dataset I've given you already.

```
##install.packages("NetData")
library(NetData)

#Pull the dataset out:
data(kracknets, package = "NetData")
```

If you don't have NetData and use the archived set, unzip it first. Then, load the data:

```
load("advice_data_frame.Rdata")
load("friendship_data_frame.Rdata")
load("reports_to_data_frame.Rdata")
```

In your Global Environment, you now see three sets of data, all in format "name"_data_frame. So, we have three different datasets in one, and they are obviously dataframes. Let's look at what these datasets are:

```
head(advice_data_frame)
```

```
##   ego alter advice_tie
## 1   1     1          0
## 2   1     2          1
## 3   1     3          0
## 4   1     4          1
## 5   1     5          0
## 6   1     6          0
```

```
head(friendship_data_frame)
```

```
##   ego alter friendship_tie
## 1   1     1              0
## 2   1     2              1
## 3   1     3              0
## 4   1     4              1
## 5   1     5              0
## 6   1     6              0
```

```
head(reports_to_data_frame)
```

```
##   ego alter reports_to_tie
## 1   1     1              0
## 2   1     2              1
## 3   1     3              0
## 4   1     4              0
## 5   1     5              0
## 6   1     6              0
```

So obviously, we have ego networks of people who give advice to each other, who are friends, and who reports to whom. From these three dataframes, we can make an array of data, combining all three dataframes into one.

```r
# First, make an array of 3 the 3 data frames
krack <- list(advice_data_frame,
              friendship_data_frame,
               reports_to_data_frame)

## add names to objects in list
graphs <- c('advice','friendship','reports')
names(krack) <- graphs
## check on list
length(krack) #how many elements we have in our krack dataset
```

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

```r
names(krack)
```

```
## [1] "advice"     "friendship" "reports"
```

"For" loops

Now we need to turn each element into a matrix. We already know how to extract elements from a matrix, and we can certainly do it here one element at a time. We have only three - it won't take too much time. But what if we have a dozen? Two dozen? It's the same code over and over, only one element (matrix number) will change. In programming, repetitive tasks that are almost identical are executed with the help of a *loop* - and there are several types. Below is a simple *"for" loop*, which contains several elements:

1. for (i in n) - how long to repeat the loop and for what elements. Here, we tell R that code below needs to be executed for the ith element in n total elements.
2. code to execute (note the curly brackets)

R recognizes the "for" loop and will increment i after completing the code - it will start with number 1 (first i), run the code, increment the i to the second i, run the code, and so on until it's done (reaches the end, n). In our case, the loop will only run three times:

```r
for (i in 1:length(krack)){
  krack[[i]] <- as.matrix(krack[[i]])
}
```

Loops are generally very useful, so we encourage you to familiarize yourselves with them.

Another useful command is a "subset" command - it can take only a part of data. For analyzing network structure, once we've examined the network composition, it is sometimes helpful to remove nodes with zero edges - meaning, nodes that are isolates, not connected to anyone. This is what we do with the loop below (notice the syntax of the *subset* command:

```r
for(i in 1:3){
krack[[i]] <- subset(krack[[i]],
                (krack[[i]][,3] > 0 ))
}
dim(krack[[1]]) # What is the size now?
```

```
## [1] 190   3
```

```r
# Quick look:
head(krack[[1]])
```

```
##      ego alter advice_tie
## [1,]   1     2          1
## [2,]   1     4          1
## [3,]    1     8          1
## [4,]    1    16          1
## [5,]    1    18          1
## [6,]    1    21          1
```

Now, we make three graphs of our networks, and you should already understand what we do with each of the commands below:

```
names(attributes)
```

```
## [1] "AGE"    "TENURE" "LEVEL"  "DEPT"
```

```
library(network)
```

```
## network: Classes for Relational Data
## Version 1.16.1 created on 2020-10-06.
## copyright (c) 2005, Carter T. Butts, University of California-Irvine
##                     Mark S. Handcock, University of California -- Los Angeles
##                     David R. Hunter, Penn State University
##                     Martina Morris, University of Washington
##                     Skye Bender-deMoll, University of Washington
##  For citation information, type citation("network").
##  Type help("network-package") to get started.
```

```
for (i in 1:3){
krack[[i]] <- network(krack[[i]],
                matrix.type = 'edgelist',
                  vertex.attr = list(attributes[,1], attributes[,2],
                                         attributes[,3], attributes[,4]),
                  vertex.attrnames = list("AGE","TENURE","LEVEL","DEPT"))
}
advice <- krack$advice
friendship <- krack$friendship
reports <- krack$reports


# Check networks
advice
```

```
##  Network attributes:
##   vertices = 21
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##   total edges= 190
##     missing edges= 0
##     non-missing edges= 190
##
##  Vertex attribute names:
##     AGE DEPT LEVEL TENURE vertex.names
##
## No edge attributes
```

```
friendship
```

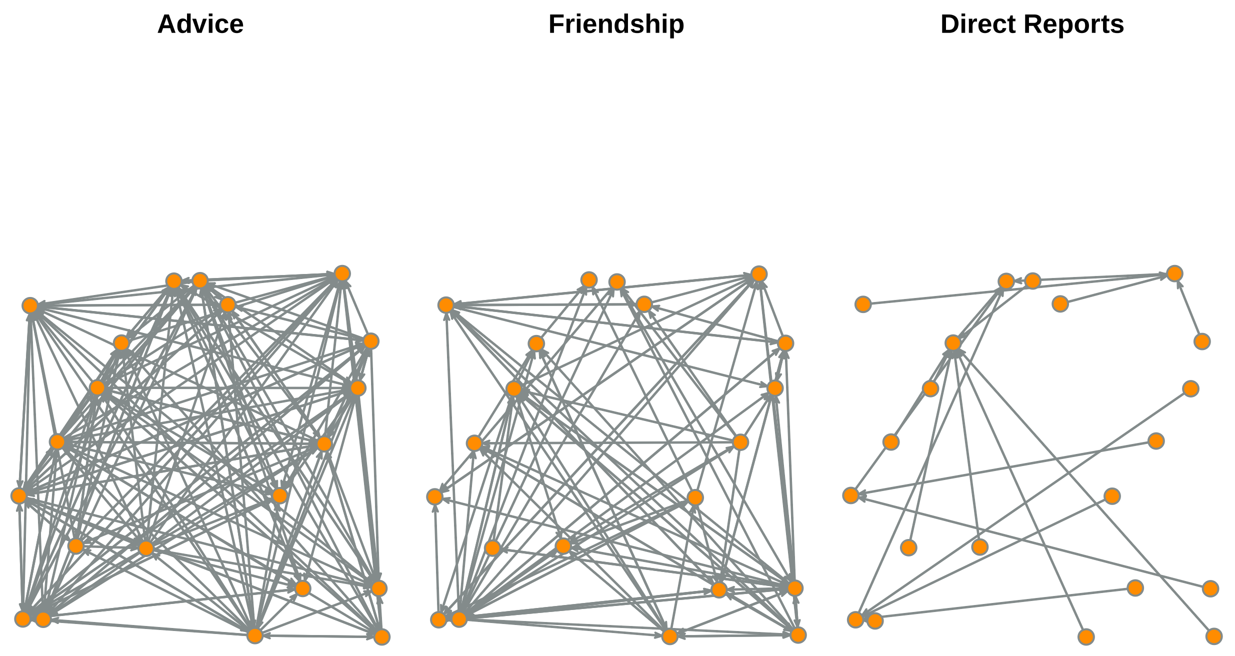
```
##  Network attributes:
```

```
##   vertices = 21
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##   total edges= 102
##     missing edges= 0
##     non-missing edges= 102
##
##  Vertex attribute names:
##     AGE DEPT LEVEL TENURE vertex.names
##
## No edge attributes
```

```
reports
```

```
##  Network attributes:
##   vertices = 21
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##   total edges= 20
##     missing edges= 0
##     non-missing edges= 20
##
##  Vertex attribute names:
##     AGE DEPT LEVEL TENURE vertex.names
##
## No edge attributes
```

```
# Let's take a look at these data.
# First, we create a set of coordinates to run the plot in.
# Detailed explanation of what we do here is provided in the answers to the third seminar.
n<-network.size(advice)
v1<-sample((0:(n-1))/n) #create a vector of random numbers
v2<-sample(v1)
x <- n/(2 * pi) * sin(2 * pi * v1)
y <- n/(2 * pi) * cos(2 * pi * v2)
mycoord <- cbind(x,y)

# Plot networks:
par(mar=c(0,0,1,0))
par(mfrow=c(1,3))
plot(advice, edge.col='azure4', vertex.col='darkorange',
     vertex.border='azure4',vertex.cex=2,coord=mycoord,
     main ='Advice')
plot(friendship, edge.col='azure4', vertex.col='darkorange',
     vertex.border='azure4',vertex.cex=2, coord=mycoord,
     main ='Friendship')
plot(reports, edge.col='azure4', vertex.col='darkorange',
     vertex.border='azure4',vertex.cex=2, coord=mycoord,
     main='Direct Reports')
```



Assignment task 1

For the networks we've obtained, please calculate the following:

1. Dyad census
2. At least three types of centrality
3. Triad census
4. Transitivity

Having performed the calculations, please compare your results for each network and make appropriate inferences.

Blockmodeling

Now, we started experimenting with building models out of our data. Remember that blockmodeling is rearrangement of data based on some *a priori* theoretical attribute, such as role or position in the network. We are going to look at building blockmodels from two datasets, "formal" - an adjacency matrix and "roles" - an attribute table where roles for each member of the "formal" group have been set in advance. In "real life" and with real data, you will be selecting roles yourself, and this is called the "exploratory" blockmodeling - we will learn how to do that, too.

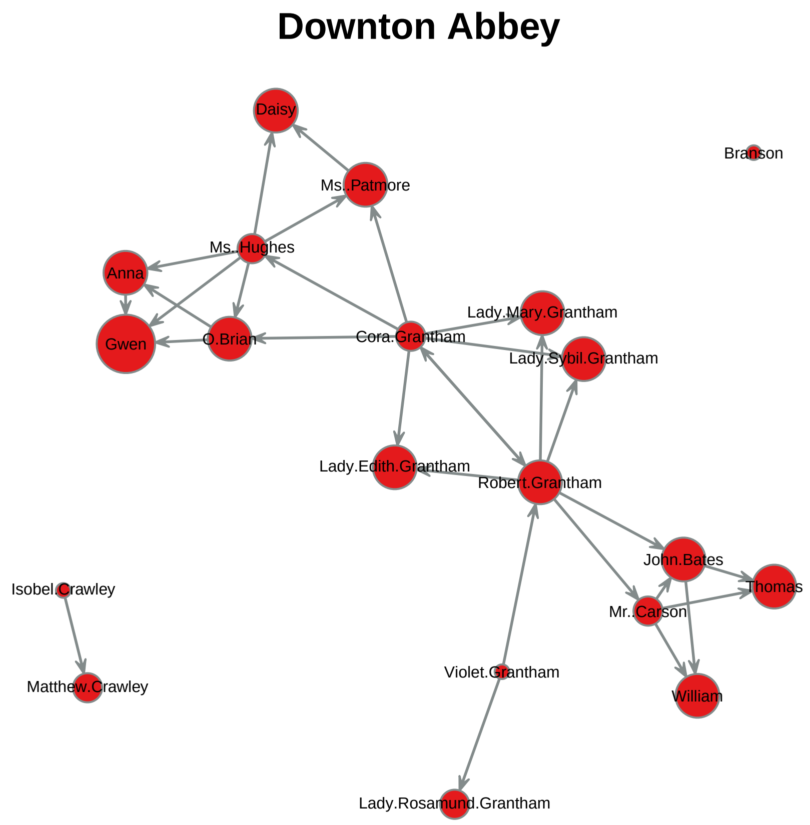
A priori blockmodel

For this part of our work, we will need packages *network* and *sna*. Read the data:

```
library(network)
library(sna)
```

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

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

## The following object is masked from 'package:base':
##
##     order

## sna: Tools for Social Network Analysis
## Version 2.6 created on 2020-10-5.
## copyright (c) 2005, Carter T. Butts, University of California-Irvine
##  For citation information, type citation("sna").
##  Type help(package="sna") to get started.
```

```
formal<-as.matrix(read.csv("formal.csv", header = TRUE, row.names=1))
roles<-read.csv("roles.csv", header=TRUE, row.names=1)

formalnet <- network(formal)
par(mar=c(0,0,2,0))
indeg <- degree(formalnet, cmode = 'indegree')
mycoord <- plot(formalnet, displaylabels=TRUE, edge.col='azure4',
                vertex.col="#E41A1C", vertex.border='azure4',
                vertex.cex = indeg + 1 , main ='Downton Abbey',
                label.cex=0.5, label.pos = 5)
```

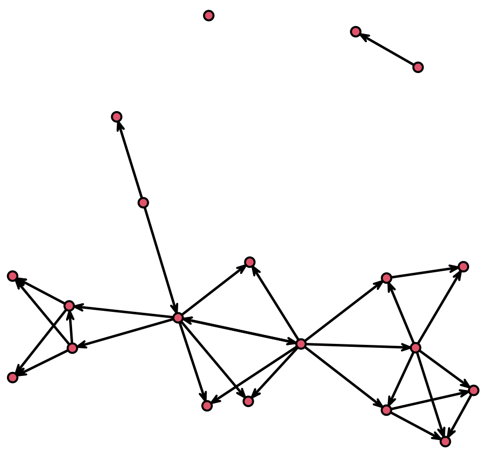


Next, we need to symmetrize our data. Remember how we symmetrized the matrix before? We can symmetrize the graph the same way. Also remember that symmetrizing the network is the transformation of a directed/asymmetric one-mode network into an undirected/symmetric one-mode network. There are also two symmetry types:

1. Strong: $i <-> j$ iff (means "if and only if") $i-> j$ and $j-> i$ (we also refer to it as the "AND rule").
2. Weak: $i <-> j$ iff $i-> j$ or $j-> i$ (the "OR rule").

We also know the drawbacks and benefits of symmetrizing, and one of the benefits, among others, is the ability to build meaningful blockmodels. If we look at the network without the bells and whistles I've added to the plot above, this is what it looks like:

```
plot(formalnet)
```



Obviously, it's a directed network with both one-directional and mutual ties, so we can use both the AND and the OR rules for symmetrizing the network.

```
orRule <- symmetrize(formalnet, rule='weak')    # "or" rule
class(orRule) # symmetrize transformed the network into a matrix
```

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

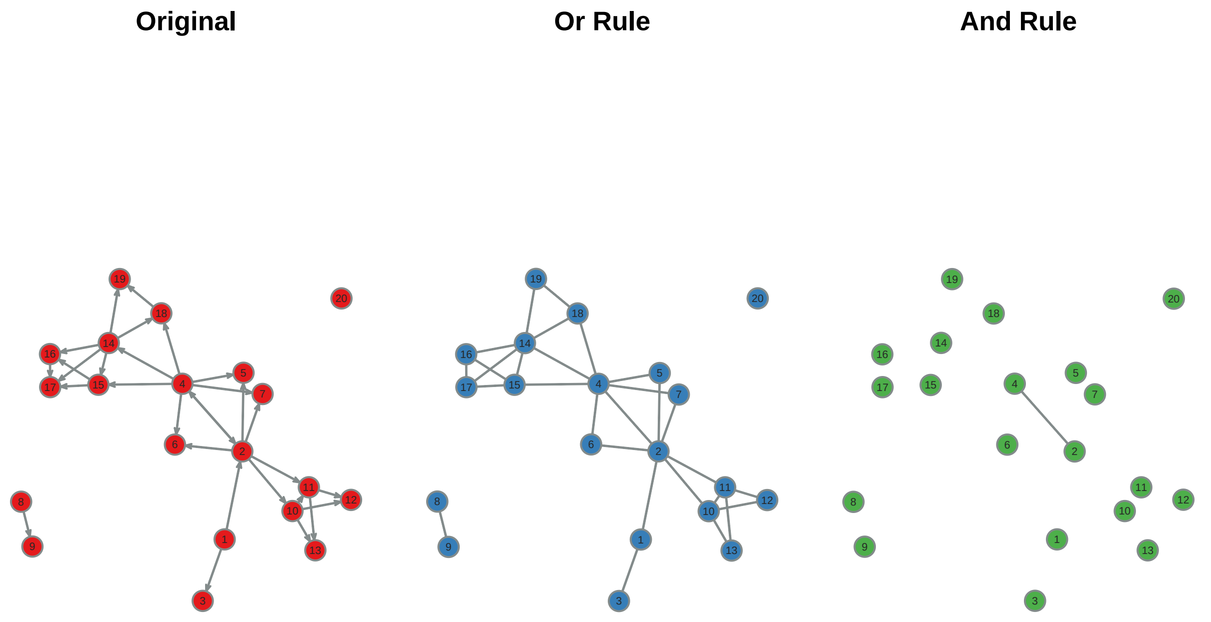
```
orRule <- network(symmetrize(formalnet, rule='weak'),
                  directed = FALSE) # 'or' rule
class(orRule) # network
```

```
## [1] "network"
```

```
andRule <- network(symmetrize(formalnet, rule='strong'),
                   directed = FALSE) # 'and' rule
```

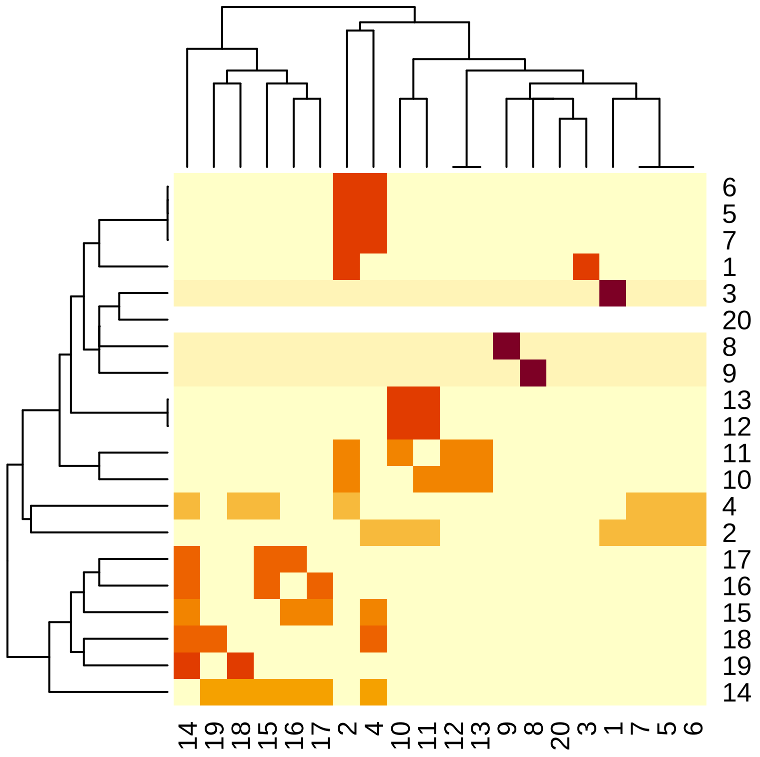
Let's look at what we have as a result:

```
par(mar=c(1,1,2,1))
par(mfrow=c(1,3))
plot(formalnet, main = 'Original', coord=mycoord, vertex.cex =3,
     edge.col='azure4', vertex.col="#E41A1C", vertex.border='azure4',
     label=seq(1:20),label.pos=5,label.cex=.5,label.col='gray15')
plot(orRule, main = 'Or Rule', coord=mycoord, vertex.cex =3,
     edge.col='azure4', vertex.col="#377EB8", vertex.border='azure4',
     label=seq(1:20),label.pos=5,label.cex=.5,label.col='gray15')
plot(andRule, main = 'And Rule', coord=mycoord, vertex.cex =3,
     edge.col='azure4', vertex.col="#4DAF4A", vertex.border='azure4',
     label=seq(1:20),label.pos=5,label.cex=.5,label.col='gray15')
```



Now, let's create our first blockmodel based on the community detection output (in our file "roles," it's the last column, called "commdetect"). We will call this "a priori" formal blockmodel, because we are building a block models on roles that have been predetermined.

```
# A more descriptive name, so we don't get confused:
snasymmformal <- orRule

aprioriformal<-blockmodel(snasymmformal, roles$commdetect,
                          block.content="density", mode="graph",
                          diag=FALSE)

# We can build what is called a heatmap, showing the relationships between blocks in color:
heatmap(aprioriformal[[4]])
```



Now, that may have looked cool, but what does that all mean? We can color nodes by blocks they belong to.

```
# Let's visualize the network with nodes colored by block.
# These are our blocks.
apriorif ormal[[1]]
```

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

```
apriorif ormal[[2]]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

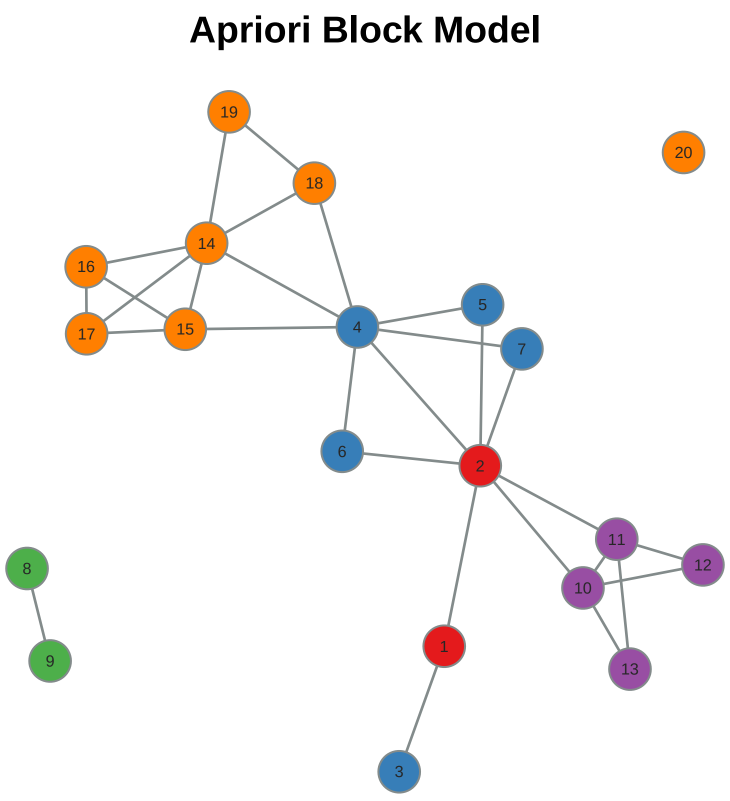
```
apriorif ormal[[3]]
```

```
## [1] "density"
```

```
apriorif ormal[[4]]
```

```
##    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
## 1  0 1 1 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 2  1 0 0 1 1 1 1 0 0  1  1  0  0  0  0  0  0  0  0  0
## 3  1 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 4  0 1 0 0 1 1 1 0 0  0  0  0  0  1  1  0  0  1  0  0
## 5  0 1 0 1 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 6  0 1 0 1 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 7  0 1 0 1 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 8  0 0 0 0 0 0 0 0 1  0  0  0  0  0  0  0  0  0  0  0
## 9  0 0 0 0 0 0 0 1 0  0  0  0  0  0  0  0  0  0  0  0
## 10 0 1 0 0 0 0 0 0 0  0  1  1  1  0  0  0  0  0  0  0
## 11 0 1 0 0 0 0 0 0 0 1  0  1  1  0  0  0  0  0  0  0
## 12 0 0 0 0 0 0 0 0 0 1  1  0  0  0  0  0  0  0  0  0
## 13 0 0 0 0 0 0 0 0 0 1  1  0  0  0  0  0  0  0  0  0
## 14 0 0 0 1 0 0 0 0 0 0  0  0  0  0  1  1  1  1  1  0
## 15 0 0 0 1 0 0 0 0 0 0  0  0  0  1  0  1  1  0  0  0
## 16 0 0 0 0 0 0 0 0 0 0  0  0  0  1  1  0  1  0  0  0
## 17 0 0 0 0 0 0 0 0 0 0  0  0  0  1  1  1  0  0  0  0
## 18 0 0 0 1 0 0 0 0 0 0  0  0  0  1  0  0  0  0  1  0
## 19 0 0 0 0 0 0 0 0 0 0  0  0  0  1  0  0  0  1  0  0
## 20 0 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0
```

```r
library(RColorBrewer)
par(mar=c(1,1,1,1),mfrow=c(2,3))
col5 <- brewer.pal(5, 'Set1')
cols <- ifelse(aprioriformal[[1]] == 1, col5[1],
          ifelse(aprioriformal[[1]] == 2, col5[2],
           ifelse(aprioriformal[[1]] == 3, col5[3],
            ifelse(aprioriformal[[1]] == 4, col5[4], col5[5]))))
par(mar=c(1,1,2,1),mfrow=c(1,1))
plot(snasymmformal, main = 'Apriori Block Model', coord=mycoord,
     vertex.cex =3, edge.col='azure4', vertex.col=cols,
     vertex.border='azure4', label=seq(1:20), label.pos=5,
     label.cex=.5, label.col='gray15')
```



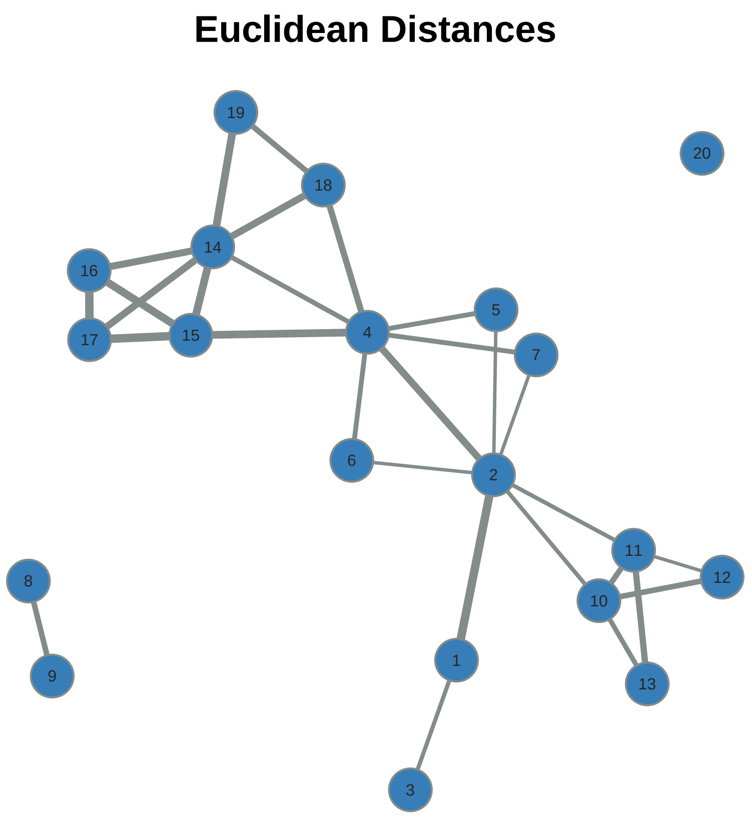
Exploratory block model

Of course, it would be just great if the roles have always been identified for us, but in real life, it's not going to happen. So we have to have a way to find these roles ourselves, and as we've seen in lecture, there are a few ways to do that. So we are going to use the instruments we have already to try and extract the roles from the network.

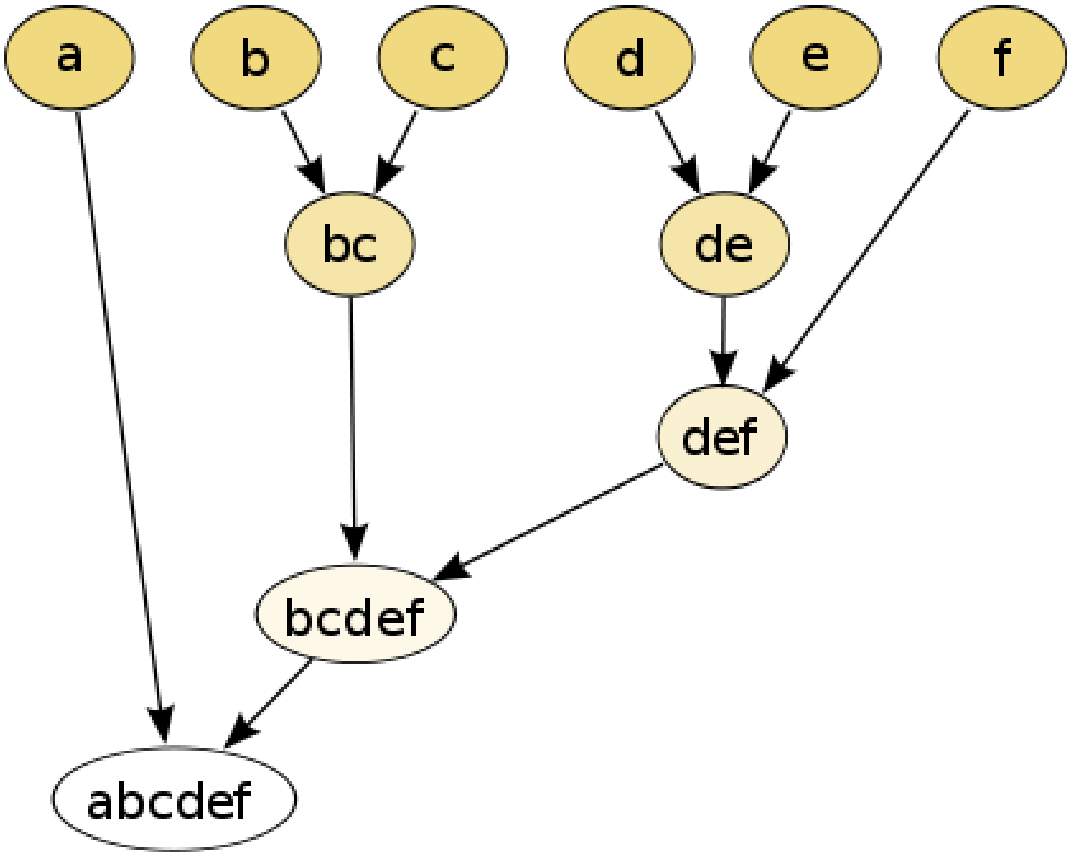
Distance Matrix & Hierarchical Clustering

We can use Euclidian distances to create a distance matrix. Remember we created an "or rule" for our data? We'll refer to it when generating euclidian distances:

```r
# Create an object of distances in the "OR rule," and turn it into a vector
distformal <- dist(snasymmformal, method="euclidian", diag=FALSE)
thick <- as.vector(distformal)

# Now, let's visualize these distances as edge thickness
par(mar=c(0.5,0,2,0))
plot(snasymmformal, main = 'Euclidean Distances', coord=mycoord,
     vertex.cex =3, edge.col='azure4', vertex.col=col5[2],
     vertex.border='azure4', label=seq(1:20),label.pos=5,
     label.cex=.5,label.col='gray15', edge.lwd = thick^2)
```



What we have to do next is hierarchical clustering, a method of data clustering based not on even clusters, but on clusters as subclusters of larger structures (you may have seen this method in your previous statistical courses). I've shown you the picture of it in the lecture, but here is another example:



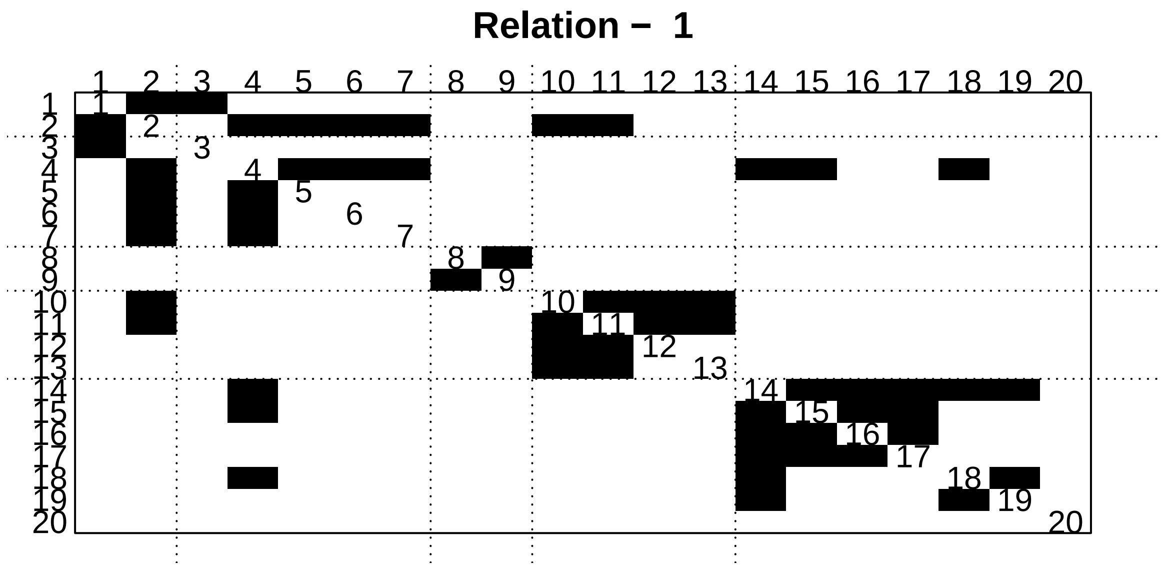
Command for doing so is just “hclust,” it’s an R-native command:

```
# Cluster analysis
formalclust <- hclust(distformal, method="complete")
```

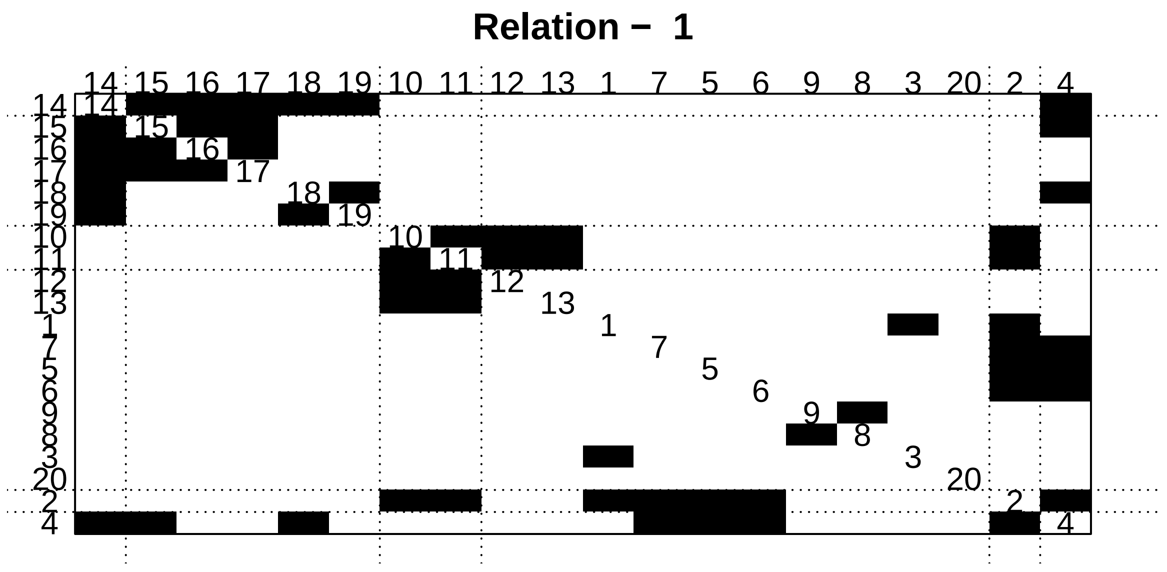
Exploratory blockmodel

Once we have created a formal set of clusters based on the *hclust* command, we can use clusters to build blockmodels:

```
# And now, a blockmodel based on clustering:
exploratoryformal<-blockmodel(snasymmformal, formalclust, k=6,
                              block.content="density", mode="graph",
                              diag=FALSE)

# Plot the two blockmodels one after another for comparison:
par(mar=c(0,0,2,0))
plot.blockmodel(aprioriformal)
```



```
plot.blockmodel(exploratoryformal)
```

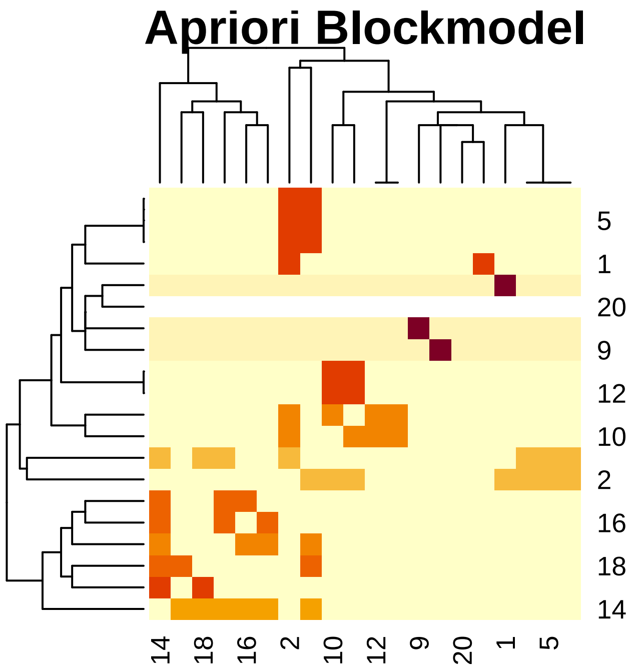


Assignment task 2

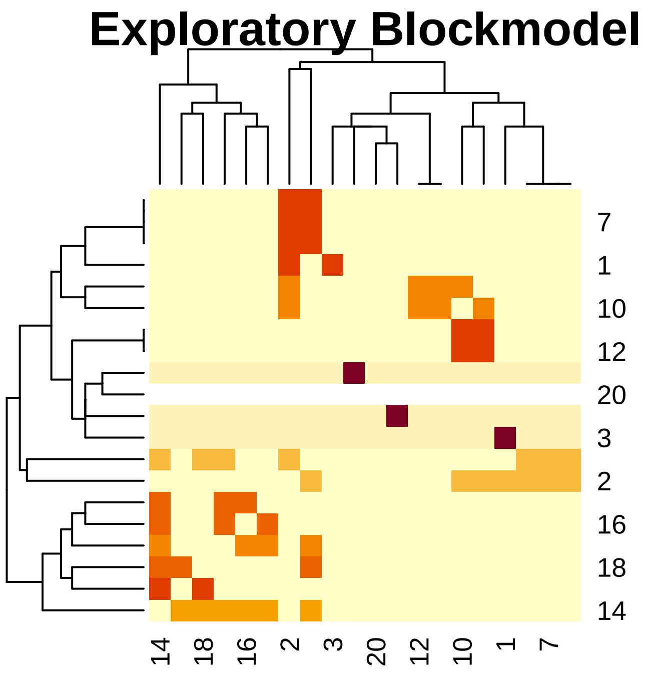
1. Experiment with k. We've set it to 6, but would another number make more sense?
2. Which of the two blockmodels appear to be more accurate to you? Why?

Finally, we can make a heatmap of the two blockmodels:

```
par(mar = c(1,1,4,1), mfrow = c(1,2))
heatmap(aprioriformal[[4]], main ='Apriori Blockmodel')
```



```
heatmap(exploratoryformal[[4]], main ='Exploratory Blockmodel')
```



Blockmodeling based on CONCOR function

You will need to carefully look at the lecture to realize what a CONCOR function does. Remember, this is a method that is based on CONvergence of iterated CORrelations, when we use the similarity between the columns of a correlation matrix to group data together. A very reasonable reference for how it works is provided here: https://www.r-bloggers.com/concor-in-r/, even with the CONCOR routine available as a package. However, without seeing the code of the routine, we can't be certain that it's done correctly, so we are using a function that we can guarantee works correctly (you can check it for yourself - just follow my code, we have put it in the section "For the Most Curious.") The function itself is written below, and you need to copy it entirely into the console and into the RMarkdown, or the method will not work.

When you run your matrix through a CONCOR function, it will generate a set of blocks. The nodes inside them are the closest to each other, but it does not mean the partition is final. They are the closest to each other only *in relation* to the other blocks - meaning, through the iteration of correlations, they've become separated from the rest. But we can - and should - examine each obtained block again (on the original matrix) to make sure that it can't be partitioned further.

The CONCOR function

Let's build the CONCOR function. First, remove the isolate that is not connected to anyone or correlates with anyone (node 20):

```
connectedformal<-formal[-20,-20] # operation on the matrix
class(connectedformal)
```

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

Now, the CONCOR function:

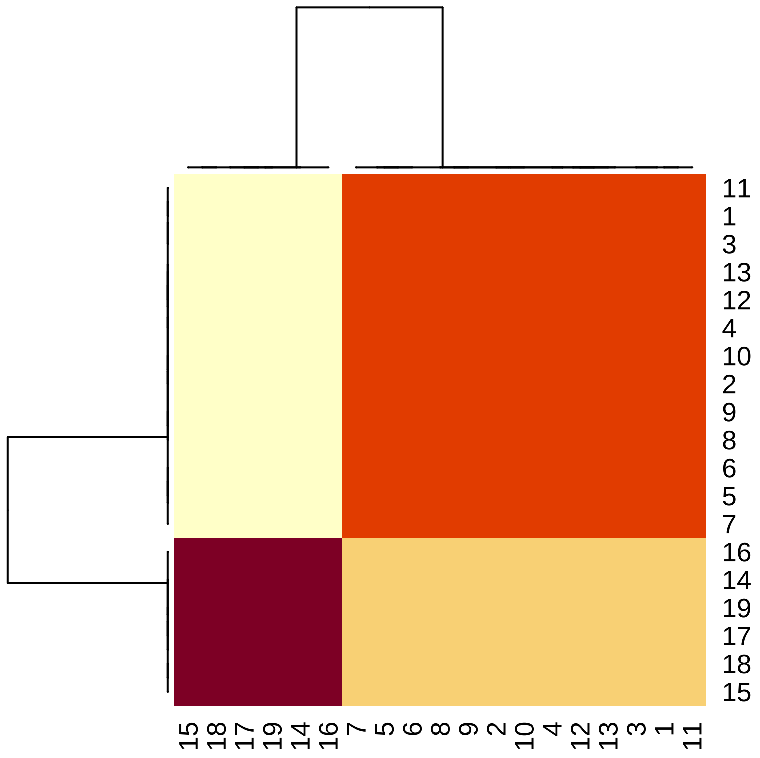
```
CONCOR <- function(mat, max.iter=1000, epsilon=1e-10){
  mat <- rbind(mat, t(mat)) # stack
  colN <- ncol(mat) # width
  X <- matrix(rep(0, times=colN*colN), nrow=colN, ncol=colN)
  target.abs.value <- colN * colN - epsilon # convergence target
  for (iter in 1:max.iter){
    for(i in 1:colN){
      for(j in i:colN){
        X[i,j]<-cor(mat[,i], mat[,j], method=c("pearson"))
      } # end for j
    } # end for i
    mat <- X+(t(X)-diag(diag((X))))
    if (sum(abs(mat)) > target.abs.value) { # test convergence
      #Finished before max.iter iterations
      return(mat)
    } # end if
  } # end for iterations
  return(mat) # return matrix
} # end function
```

Blockmodeling based on CONCOR

Now, let's create a blockmodel based on CONCOR function. We take the original matrix of data and run it through the function.

```
rownames(connectedformal) <- row.names(roles)[1:19]
colnames(connectedformal) <- row.names(roles)[1:19]

# Command below is commented out, but you can look at it to see what the resulting matrix is.
```

```
## connectedformal

#Now, run the matrix through the CONCOR function and show the blockmodel:
CONCORFORMAL<-CONCOR(connectedformal)

# You can look at the matrix on your own; we commented it out to save space in the document:
## print(CONCORFORMAL)
heatmap(CONCORFORMAL)
```

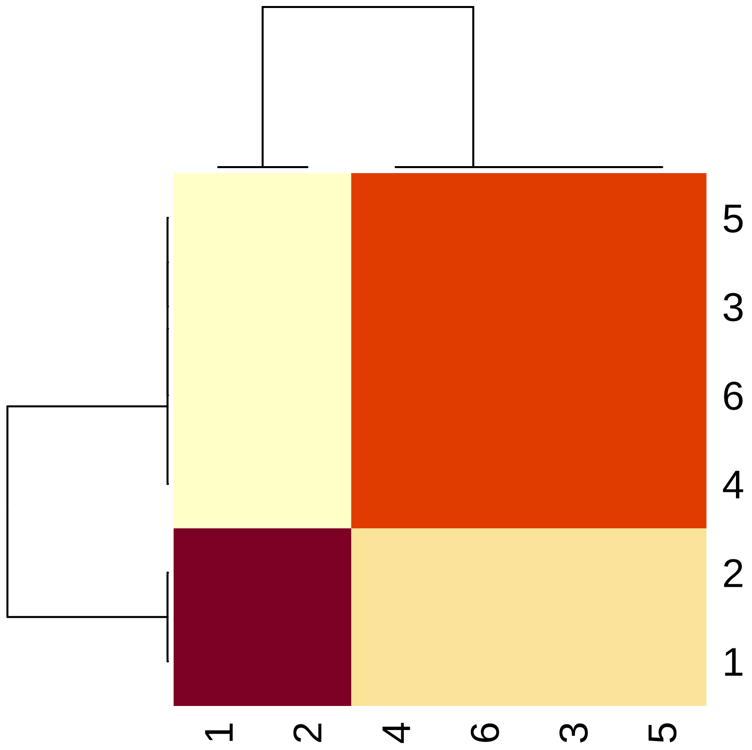


We have obtained the first result. Now, obviously we have four distinct blocks (actually, it's 2x2, so we only need to examine two blocks). Can we separate them further? We should at least try. Are they meaningful? Let's look at them:

```
## part 1  -it's blocks from 14 to 19:
part1 <- connectedformal[14:19,14:19]
colnames(part1) # Who are in this partition?
```

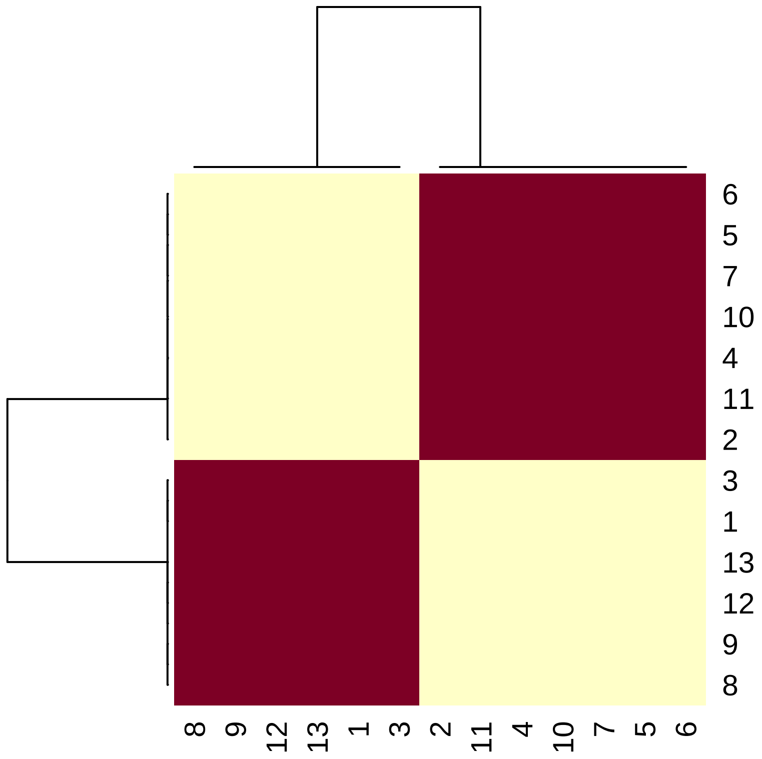
```
## [1] "Ms. Hughes"  "O'Brian"     "Anna"        "Gwen"        "Ms. Patmore"
## [6] "Daisy"
```

```
concor1 <- CONCOR(part1)
heatmap(concor1)
```



In partition Part11 we have Hughes and O'Brian and Part12 = Anna, Gwen, Ms.Patmore, and Daisy. This partition seems meaninful, given our data. What about the second block? Actually, it's rather large, from 1 to the 13th node. Let's isolate it into a separate matrix and run it through CONCOR again:

```
part2 <- connectedformal[1:13,1:13] # isolate the first 13 nodes

# We commented the matrix out, but you can look at it on your own
##part2
concor2 <- CONCOR(part2) # Run through CONCOR
heatmap(concor2) # Look at the result
```



Looks like we've gotten another set of blocks, which we can look at separately. Let's create a third partition:

```
part3<-c(1,3,8,9,12,13) # isolate the needed nodes
part3.1<-part2[part3,part3] # remove the isolates from partition 2
colnames(part3.1) # Who is here?
```

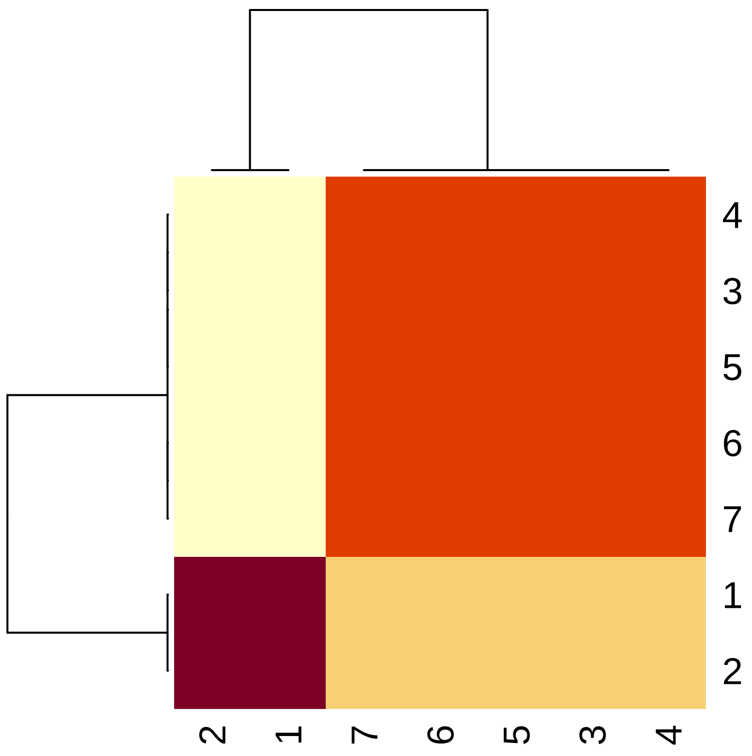
```
## [1] "Violet Grantham"        "Lady Rosamund Grantham" "Isobel Crawley"
## [4] "Matthew Crawley"        "Thomas"                 "William"
```

Now, to see if we can partition block 31 further, we need to run it through CONCOR function again. If it does not run, generating an error, it means that the partition we've obtained was final. And it was final - run the code below in CONSOLE ONLY, because it will choke the RMarkdown with errors it generates:

```
## concor3.1<-CONCOR(part3.1)
```

But this is your indication that we do not continue breaking up the matrix - we have found the final partition for section 31. What about section 32? Let's extract what's left of the partition 2 after we removed the first six nodes:

```
part3.2 <- part2[-part3,-part3] # Extract remaining nodes from part2
concor3.2 <- CONCOR(part3.2)  # Run it through CONCOR
heatmap(concor3.2)
```



So unlike part 3.1, part 3.2 could be partitioned again - it ran through CONCOR and we got yet another heatmap. We can look at the two partitions to see if we can further break them up:

```
colnames(part3.2[1:2,1:2]) # Names in the first subpart
```

```
## [1] "Robert Grantham" "Cora Grantham"
```

```
colnames(part3.2[3:7,3:7]) # Names in the second subpart
```

```
## [1] "Lady Mary Grantham"  "Lady Edith Grantham" "Lady Sybil Grantham"
## [4] "Mr. Carson"          "John Bates"
```

Now, in nodes 3-7 we have three Grahams and two random people. But the final attempt at partitioning them will indicate that this block cannot be broken further. There is obviously something in that group of people that makes them connected:

```
part3.2.2 <- part3.2[3:7,3:7] # Create a partition

# Code below will choke RMarkdown, run it in CONSOLE ONLY (it's commented out here):
##concor3.2.2<-CONCOR(part3.2.2)
```

So, as a result, we have broken up our matrix, using the CONCOR function, on a number of blocks.

Assignment task 3

Try not to get lost in all the partitions! Please list all the finite block-partitions that we have generated and the names of all people that ended up in every block

For the most curious: the CONCOR function

This is how the CONCOR function was built and how it works: feel free to explore the code a line at a time.

```
# Set up an example matrix.
mat <- matrix(rbinom(25, 1, 0.5), nr = 5, nc = 5)
colnames(mat) <- c('A','B','C','D','E')
rownames(mat) <- c('A','B','C','D','E')

# Stack the matrix with its transpose
mat
```

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

```
t(mat)
```

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

```
# what if this was a symmetrical matrix? Would it work? -- think about it. :-)
mat <- rbind(mat,t(mat))

# Then concor makes a square matrix of 0s of the same dimensions as mat's width

X <- matrix(rep(0, times=5*5), nrow=5, ncol=5)
X
```

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

```
# Then for each cell in X it puts the correlation between the stack matrix's
# columm of Xrow and the colum of Xcol
X[2,4]<-cor(mat[,2], mat[,4], method=c("pearson")) ##
X
```

```
##      [,1] [,2] [,3]       [,4] [,5]
## [1,]    0    0    0  0.0000000    0
## [2,]    0    0    0 -0.2182179    0
## [3,]    0    0    0  0.0000000    0
## [4,]    0    0    0  0.0000000    0
## [5,]    0    0    0  0.0000000    0
```

```
# Rembember the formula for pearson's r?
## cor(XY) = cov(X,Y) / sd(X)*sd(Y)
# where:
```

```
## cov(X,Y) = E[(X - MuX)*(Y - MuY)]
## and sd(X) = sqrt(E( X - MuX)
###

# The function works until it finishes filling the X matrix once for each cell in X

for(i in 1:5){
  for(j in i:5){
    X[i,j]<-cor(mat[,i], mat[,j], method=c("pearson"))
  } # end for j
} # end for i
X
```

```
##      [,1] [,2]      [,3]       [,4]       [,5]
## [1,]    1 -0.6 0.0000000 -0.2182179 -0.2000000
## [2,]    0  1.0 0.4082483 -0.2182179 -0.2000000
## [3,]    0  0.0 1.0000000 -0.3563483  0.0000000
## [4,]    0  0.0 0.0000000  1.0000000 -0.2182179
## [5,]    0  0.0 0.0000000  0.0000000  1.0000000
```

```
# Then it makes a stacked matrix again
mat <- X+(t(X)-diag(diag((X))))
mat
```

```
##            [,1]       [,2]       [,3]       [,4]       [,5]
## [1,]  1.0000000 -0.6000000  0.0000000 -0.2182179 -0.2000000
## [2,] -0.6000000  1.0000000  0.4082483 -0.2182179 -0.2000000
## [3,]  0.0000000  0.4082483  1.0000000 -0.3563483  0.0000000
## [4,] -0.2182179 -0.2182179 -0.3563483  1.0000000 -0.2182179
## [5,] -0.2000000 -0.2000000  0.0000000 -0.2182179  1.0000000
```

```
# Then it iterates with the updated X and the updated mat
# Then it iterates...
# Up to n times or until it only has -1s and 1s in the matrix
```

Blockmodel-based network clustering

Next we will take our work one step further - we'll build clusters. For this purpsoe, we will work with another package from *NetData* package, called "studentnets.M182." If for some reason you do not have will need the *NetData* package, so install the package (remember - do it in Console!) and call it from RMarkdown. This package contains a number of datasets, which can be used for labs and practice - if you are looking for more data to practice on your own, check this package out (using the "?NetData" command in Console). By the way, you are welcome to use this package to find datasets when I ask you to repeat commands on a different set of data.

As you know, datasets in this package are stored as a data frame, which you already know how to work with. Today, we will be applyng *igraph* commands to extract network into a graph object.

```
##install.packages("NetData")
library(NetData)

# Pull the dataset out - same way as we did with Kracknets:
data(studentnets.M182, package = "NetData")

# Check the data we have in the dataset:
head(m182_full_data_frame)
```

```
##   ego alter friend_tie social_tie task_tie
## 1   1     1          0        0.0      0.0
## 2   1     2          0        0.0      0.0
## 3   1     3          0        0.0      0.0
## 4   1     4          0        0.0      0.0
## 5   1     5          0        1.2      0.3
## 6   1     6          0        0.0      0.0
```

If you do not have NetData, the file is provided. Unzip it first, then load the data.

```
load("friend_df.Rdata")
load("m182_full_data_frame.Rdata")
load("social_df.Rdata")
load("task_df.Rdata")
```

This dataset has information on students and their ties - friendship, social and task (apparently, doing homework together or some other similiar type of connection). So unlike last week, when in our *kracknets* we had only one column with ties, today we have three.

Remember, unlike edgelist, the dataframe stores a product of every single node with every single other node, where absense of ties is coded as 0. Even a brief look at the data shows us that we have lots of nodes with no ties to other nodes - they appear as all zeros in the last three columns. You already know how to work with *subset* command, and understand what it means to remove non-zero edges, so the code below does just that. One departure from last week: we are removing edges where ALL connections are absent, so this part of the code "friend_tie > 0 | social_tie > 0 | task_tie > 0" tells the program to extract a set of data where friend ties OR social ties AOR task ties are greater than zero:

```
m182_full_nonzero_edges <- subset(m182_full_data_frame,
  (friend_tie > 0 | social_tie > 0 | task_tie > 0))
head( m182_full_nonzero_edges) # Check what's left
```

```
##    ego alter friend_tie social_tie task_tie
## 5    1     5          0       1.20     0.30
## 8    1     8          0       0.15     0.00
## 9    1     9          0       2.85     0.30
## 10   1    10          0       6.45     0.30
## 11   1    11          0       0.30     0.00
## 12   1    12          0       1.95     0.15
```

OK, better. Please note that in each individual column we still have zeros - this is because we removed only the edges where ALL connections were zeros. It is entirely possible that some may have connection of one type, but not the other - and so they remained in the dataset. Now we just create a graph object out of our data frame (as I have told you, dataframe can become any object), and for that we need the *igraph* package:

```
suppressPackageStartupMessages(library(igraph))
m182_full <- graph.data.frame(m182_full_nonzero_edges)
summary(m182_full) #check the data
```

```
## IGRAPH a142591 DN-- 16 144 --
## + attr: name (v/c), friend_tie (e/n), social_tie (e/n), task_tie (e/n)
```

Now, we break up the full graph into subgraphs based on the type of a tie. Also, pay attention at the command I use: I remove the zero edges from each network.

```
m182_friend <- delete.edges(m182_full, E(m182_full)[E(m182_full)$friend_tie==0])
m182_social <- delete.edges(m182_full, E(m182_full)[E(m182_full)$social_tie==0])
m182_task <- delete.edges(m182_full, E(m182_full)[E(m182_full)$task_tie==0])
```

```
# You can check your data by uncommenting the following commands:
##m182_friend
##m182_social
##m182_task
```

And now we are ready to move onto working with block models.

Assignment question 4

Why do we remove the zero edges from networks? We haven't done it previously, why are we doing it now?

Other ways to blockmodel

We have already seen a few ways to group our nodes together: with *a priori* specified task list, using the euclidian distance, and using the CONvergent CORrelations procedure. Turns out, there are more ways to group our nodes.

Hierarchical clustering on social and task ties

We rarely analyze data where nodes have only one type of a tie with each other. So we need a method that would help us generate blockmodels on more than one type of a connection between nodes, and compare structural equivalence on two sets of edges instead of one. So here, we will use the "task" and "social" sub-graphs together as the basis for our structural equivalence blockmodel. First, we'll use the task graph to generate an adjacency matrix.

The code below extracts the adjacency information from the graph objects we've created. The *get.adjacency* command comes with an option to generate the matrix with the actual value for the tie (valued) or just a binary matrix, where 1 is a tie, and 0 is no tie (remember what this process is called - when we replace valued data with 1s, regardless of the value?).

```
# This is if we want to use the edge value
task_adjacency<-get.adjacency(m182_task, attr='task_tie')
# This is if we only want the tie (so it's 0 or 1)
binary_task_adjacency<-get.adjacency(m182_task)
```

We will be working with valued data, so we need the first option.

Clustering directed data

In our last seminar, we worked with undirected data, losing quite a bit of information. Of course, we can create blockmodels on valued data, but the task is more involved - we have to analyze two sets of matrices. This is because fo the purpose of this type of analysis, ties are always examined from row to column; so when there is a value in the cell (1,2), we know that it's a connection FROM node 1 TO node 2. Now, if we have directed data, a value in cell (2,1) tells us that there is a connection FROM node 2 TO node 1. So, in order for the blockmodel to recognize this connection, we, in essence, have to add the transpose of a matrix to itself, so that in the same row, we will have values in cells (1,2) and cells (2,n+1), indicating the presence of directed connections between nodes 1 and 2.

This is done with a matrix operation command *rbind*, which adds the second matrix to the bottom of the first (cbind adds a matrix to the right of an existing matrix). Because we'll be working with matrices, we need to convert our graphs into matrices first, but then the rest of the commands are self-explanatory:

```
task_adjacency<-as.matrix(task_adjacency) #generate the matrix out of a graph
# Create a nx2n matrix of directed connections
task_matrix<-rbind(task_adjacency,t(task_adjacency))
```

```r
# Same for matrix of social connections:
social_adjacency<-get.adjacency(m182_social, attr='social_tie')
binary_social_adjacency<-get.adjacency(m182_social) #this is for later
social_adjacency<-as.matrix(social_adjacency)
social_matrix<-rbind(social_adjacency,t(social_adjacency))

# Because we want to analyze social and task connections together, bind matrices:
task_social_matrix <-rbind(task_matrix,social_matrix)
dim(task_social_matrix)
```

```
## [1] 64 16
```

As a result, we have a single 4n x n (64x16) matrix that represents both in- and out-directed task and social communication. From this, we can generate an n x n correlation matrix that shows the degree of structural equivalence of each actor in the network.

```r
task_social_cors<-cor(task_social_matrix) # Correlate matrices
## task_social_cors #If you want to check the matrix, uncomment this line
```

The procedure we use next is similar to what we were doing with CONCOR, when we were using euclidian distance correlations as a base for initial block separation. For this assignment, however, we will be using the library *NetCluster*, designed specifically for clustering network data. It still looks for measures of similarity (or, more accurately, dissimilarity), so we first have to create a "dissimilarity structure" for our data using dist(). We subtract the values from 1 so that they are all greater than or equal to 0; thus, highly dissimilar (i.e., negatively correlated) actors have higher values.

Please note: I utilized the *help* files for this seminar heavily, but did not copy the entire contents of the help file into this document. If you get stuck, please go back to *?NetCluster* command to see what all is in the package we are using today.

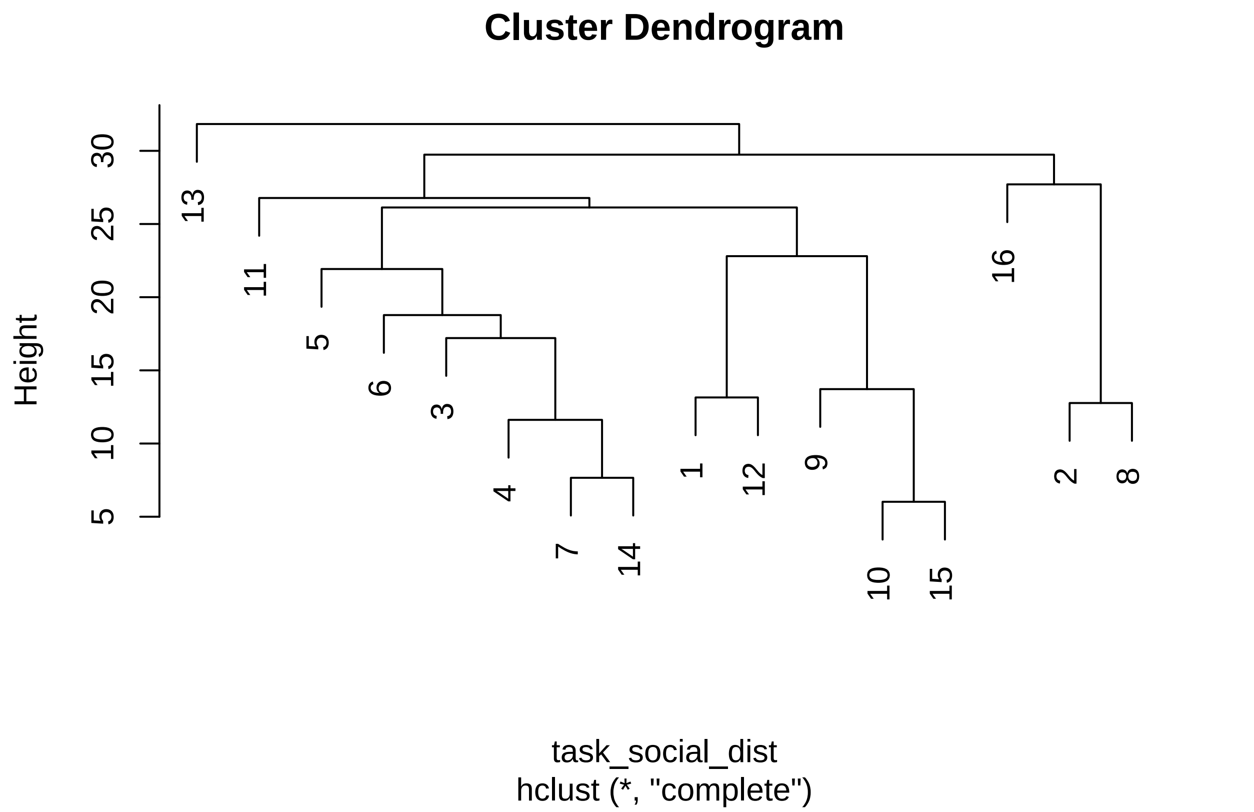
```r
dissimilarity<-1-task_social_cors #subtract matrix values from 1
task_social_dist<-as.dist(dissimilarity) #create a distance matrix

#You can check the matrix if you wish:
##task_social_dist
```

Last time we used the *dist* function directly on the matrix. But that was because we just looked at distances right away, without taking correlations into account. Since cor() looks at associations between columns and dist() looks at associations between rows, to use correlations to generate distances, you have to transpose the matrix first. So, we are back to our familiar Euclidian distances, though if you get adventurous, you can try other measures of distances.

```r
task_social_dist<-dist(t(task_social_matrix))
```

And now, we can attempt to cluster our network, using the *NetCluster* package that will perform cluster analysis directly on a network-type data (though right now, we are still with matrices). The familiar function *hclust()* performs a hierarchical agglomerative NetCluster operation based on the values in the dissimilarity matrix yielded by as.dist() above. The standard visualization is a dendrogram. By default, hclust() agglomerates clusters via a "complete linkakage" algorithm, determining cluster proximity by looking at the distance of the two points across clusters that are farthest away from one another. This can be changed via the "method" parameter.

```r
library(NetCluster) # add the library to complete the clustering
task_social_hclust <- hclust(task_social_dist)
plot(task_social_hclust)
```



This dendrogram looks better to me than the ones we did in last seminar, but it may be a matter of preference. Now, we can try to cut it to see if we can get a theoretically-appropriate number of clusters. The *cutree()* command allows us to use the output of hclust() to set different numbers of clusters and assign vertices to clusters as appropriate. For example:

```
cutree(task_social_hclust,k=2)
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
##  1  1  1  1  1  1  1  1  1  1  1  1  2  1  1  1
```

Now we'll try to figure out the number of clusters that best describes the underlying data. To do this, we'll loop through all of the possible numbers of clusters (1 through n, where n is the number of actors in the network). For each solution corresponding to a given number of clusters, we'll use cutree() to assign the vertices to their respective clusters corresponding to that solution.

From this, we can generate a matrix of within- and between-cluster correlations. Thus, when there is one cluster for each vertex in the network, the cell values will be identical to the observed correlation matrix, and when there is one cluster for the whole network, the values will all be equal to the average correlation across the observed matrix.

We can then correlate each by-cluster matrix with the observed correlation matrix to see how well the by-cluster matrix fits the data. We'll store the correlation for each number of clusters in a vector, which we can then plot.

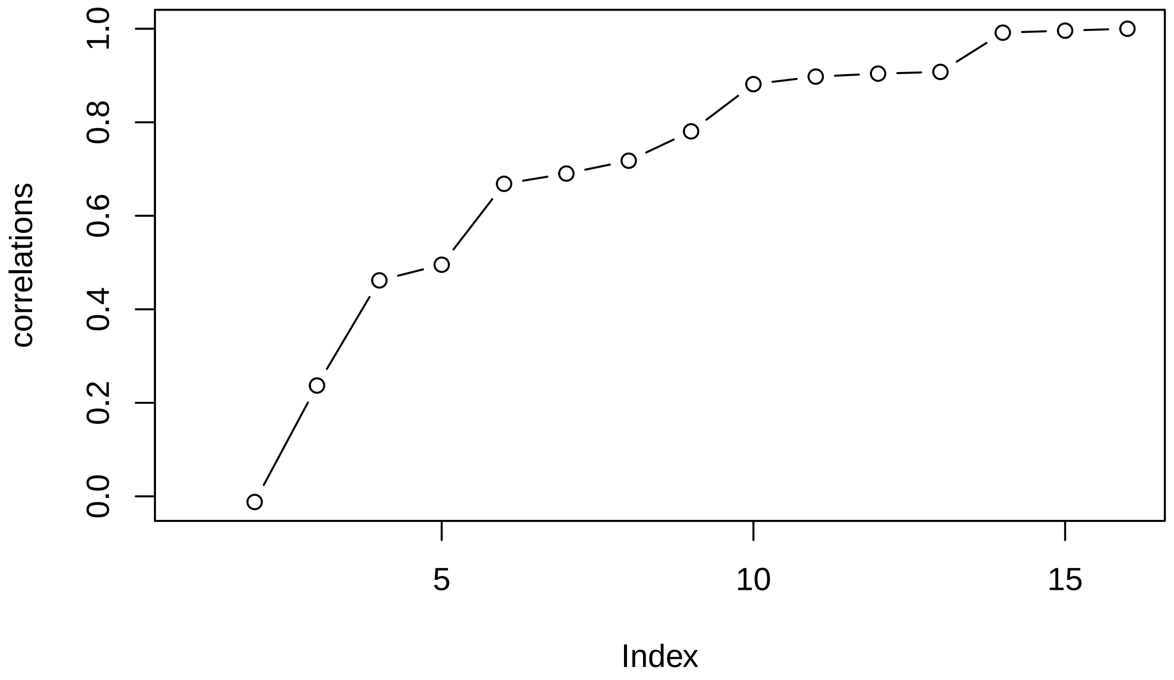
First, we initialize a vector for storing the correlations and set a variable for our number of vertices.

```
clustered_observed_cors = vector() # set it as a vector
num_vertices = length(V(m182_task)) # get the length of the vector
```

Next, we loop through the different possible cluster configurations, produce matrices of within- and between-cluster correlations, and correlate these by-cluster matrices with the observed correlation matrix. This is done via one simple command, *clustConfigurations*, which evaluates the clustering solutions to the observed correlations matrix, and returns a correlation vector and a plot.

```
clustered_observed_cors <-clustConfigurations(num_vertices,task_social_hclust,task_social_cors)
```

```
## Warning in cor(as.vector(d[g1[i], , ]), as.vector(d[g2[j], , ]), use =
## "complete.obs"): the standard deviation is zero
```



From a visual inspection of the plot and examining the correlations, we can decide on the proper number of clusters in this network. On the plot, the "index" on the x-axis indicates the number of clusters that have been examined; the line indicates the cumulative correlation accounted for with the increasing number of clusters. Note that the 1-cluster solution doesn't appear on the plot because its correlation with the observed correlation matrix is undefined. We look for the index where the line starts to flatten beyond 45 degrees, and on the plot, it appears to be number 4.

We can also look at the correlations themselves; much like with other clustering methods, we want the smallest number of clusters where the cumulative correlation increases at the fastest rate.

```
clustered_observed_cors$correlations
```

```
##  [1]          NA -0.01204435  0.23697020  0.46184446  0.49534522  0.66830515
##  [7]  0.69029066  0.71776709  0.78055897  0.88151329  0.89764596  0.90398792
## [13]  0.90765350  0.99158560  0.99590274  1.00000000
```

Now, correlations start with nothing (as explained above), as one-cluster solution is meaningless. In our example, two-cluster solution is not any more meaningful, because it actually makes it worse than not clustering at all (if that makes sense). Then there is a big jump of 23.6% with 3 clusters, and another big jump of over 20%, to 46.1%, with four. This number of clusters - four - is where the line starts to flatten out, because the next incremental increase is only about 3%, to 45.5%. But then an interesting thing happens, and we've seen it on the graph - there is another big jump to 66.8%. So what this examination tells us is that we either have 4 clusters, or we have 6 clusters, and being conservative, we start with four (but later can try six). No other number of clusters makes sense.

```
num_clusters = 4 # Test our number of clusters
clusters <- cutree(task_social_hclust, k = num_clusters)
clusters
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
##  1  2  1  1  1  1  1  2  1  1  1  1  3  1  1  4
```

```
cluster_cor_mat <- clusterCorr(task_social_cors,
                               clusters)
```

Let's look at the correlation between this cluster configuration and the observed correlation matrix. This should match the corresponding value from clustered_observed_cors above for n=4 clusters.

```
gcor(cluster_cor_mat, task_social_cors)
```

```
## [1] 0.4618445
```

Assignment questions 5

What rationale do you have for selecting the number of clusters / positions with the method above? Please rely on your knowledge of cluster analysis to answer this question.

A priori clustering

Of course, you can apply your own *a priori* set of clusters, which we learned last time, to the procedure above. Simply supply your own cluster vector (or use the one that came with the dataset), where the elements in the vector are in the same order as the vertices in the matrix, and the values represent the cluster to which each vertex belongs.

An example below is completely random for demonstration purposes only, but here actors 2, 7, and 8 formed one group, actor 16 former another group, and everyone else formed a third group. You represent this as follows:

```
apriori = c(1, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 3)
deductive_cluster_cor_mat <- generate_cluster_cor_mat(task_social_cors, apriori)
gcor(deductive_cluster_cor_mat, task_social_cors)
```

```
## [1] 0.5537549
```

Back to blockmodeling

Now that we have found more ways to find the number of appropriate blocks using alternative clustering methods, all we have left is building blockmodels out of them. We will use our 4-cluster solution to generate blockmodels, using the raw tie data from the underlying task and social networks.

```
# Blockmodel on valued task data
task_valued_blockmodel <- blockmodel(task_adjacency, clusters)

# Blockmodel on binary task data
binary_task_adjacency<-as.matrix(binary_task_adjacency) # turn graph to matrix first
task_binary_blockmodel <- blockmodel(binary_task_adjacency, clusters)

# Blockmodel on valued social data
social_valued_blockmodel <- blockmodel(social_adjacency, clusters)
```

```
#Blockmodel on binary social data
binary_social_adjacency<-as.matrix(binary_social_adjacency)
social_binary_blockmodel <- blockmodel(binary_social_adjacency, clusters)

# Now, look at the basic statistics:
task_mean <- mean(task_adjacency)
task_mean
```

```
## [1] 0.478125
```

```
task_density <- graph.density(m182_task)
task_density
```

```
## [1] 0.3666667
```

```
social_mean <- mean(social_adjacency)
social_mean
```

```
## [1] 1.157227
```

```
social_density <- graph.density(m182_social)
social_density
```

```
## [1] 0.5375
```

Assignment task 6

- Plot the resulting blockmodels in any way you wish and examine them visually. What is the story you get from viewing these clusters, and their within and between cluster densities on task and social interaction? What can you say about your network from this?
- We have learned several ways to blockmodel. Which method do you find the most intuitively appealing? Why?
- What did you learn from blockmodels about your data that you could not generate from previously learned techniques?

Homework 3

As we have seen already, homework assignments are somewhat more involved than regular seminars. Unlike previous homeworks, in today's homework we are not showing you new commands - there are plenty of them in the seminar. So we are asking you to apply what you've learned in today's seminar to a new dataset and provide interpretive explanations of your results. You are free to explore as much as possible, but at the very minimum, please do the following:

1. Choose a dataset from one of our previous labs.
2. Apply the same routines we did for the exploratory blockmodel here (it's just copy/paste and then explore the k option). Make a heatmap for your model, vary the k, make a heatmap again.....etc., until you select a k.
3. Apply the CONCOR function or any of the hierarchical approaches to the dataset you selected and plot its heatmap side by side with the heatmap for your exploratory blockmodel. What do you observe?
4. Make inference about your blockmodels.

I am going to leave here the same phrase I've left you in previous lectures. No matter what you do, I think this family of models leaves plenty of challenging opportunites. . . . but the experience is quite rewarding as well. Have fun!

Recommended literature

- Friedkin, N.E. and Johnsen, E.C., 2011. Social influence network theory: A sociological examination of small group dynamics (Vol. 33). Cambridge University Press.
- Granovetter, M.S.: The strength of weak ties. Am. J. Sociol. 78(6), 1360–1380 (1978)
- Kuskova, V., Artyukhova, E., Kamalov, R. and Danilova, D., 2016, April. Organizational Networks Revisited: Relational Predictors of Organizational Citizenship Behavior. In International Conference on Analysis of Images, Social Networks and Texts (pp. 108-117). Springer, Cham.
- Robins, G., 2015. Doing social network research: Network-based research design for social scientists. Sage.
- Wasserman, S. and Faust, K., 1994. Social network analysis: Methods and applications.

References

1. Akers, R.L., Krohn, M.D., Lanza-Kaduce, L. and Radosevich, M., 1995. Social learning and deviant behavior: A specific test of a general theory. Contemporary masters in criminology, pp.187-214.
2. Asch, S.E., 1956. Studies of independence and conformity: I. A minority of one against a unanimous majority. Psychological monographs: General and applied, 70(9), p.1.
3. Brass, D.J., Buttereld, K.D., Skaggs, B.: Relationships and unethical behavior: a social networks perspectives. Acad. Manag. Rev. 23, 14–31 (1998)
4. Bikhchandani, S., Hirshleifer, D. and Welch, I., 1992. A theory of fads, fashion, custom, and cultural change as informational cascades. Journal of political Economy, 100(5), pp.992-1026.
5. Burkhardt, M., Brass, D.J.: Changing patters or patterns of change: the effect of a change in technology on social network structure and power. Adm. Sci. Q. 35, 104–127 (1990)
6. Emerson, R.M.: Social exchange theory. Annu. Rev. Psychol. 2, 335–362 (1976)
7. Flache, A., Mäs, M., Feliciani, T., Chattoe-Brown, E., Deffuant, G., Huet, S. and Lorenz, J., 2017. Models of social influence: Towards the next frontiers. Journal of Artificial Societies and Social Simulation, 20(4).
8. Festinger, L., Schachter, S. and Back, K., 1950. Social pressures in informal groups; a study of human factors in housing.
9. Gibbons, D.: Friendship and advice networks in the context of changing professional values. Adm. Sci. Q. 49, 238–262 (2004)
10. Granovetter, M., 1978. Threshold models of collective behavior. American journal of sociology, 83(6), pp.1420-1443.
11. Granovetter, M.: The strength of weak ties: a network theory revisited (1982)
12. Gruhl, D., Guha, R., Liben-Nowell, D. and Tomkins, A., 2004, May. Information diffusion through blogspace. In Proceedings of the 13th international conference on World Wide Web (pp. 491-501).
13. Homans, G.C., 1950. The Human Group. New York: Harcourt, Brace, & World.
14. Hovland, C.I., Harvey, O.J. and Sherif, M., 1957. Assimilation and contrast effects in reactions to communication and attitude change. The Journal of Abnormal and Social Psychology, 55(2), p.244.
15. Kelman, H.C., 1958. Compliance, identification, and internalization three processes of attitude change. Journal of conflict resolution, 2(1), pp.51-60.

16. Kempe, D., Kleinberg, J. and Tardos, É., 2003, August. Maximizing the spread of influence through a social network. In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 137-146).

17. Kilduff, M.: The interpersonal structure of decision-making: a social comparison approach to organizational choice. Organ. Behav. Hum. Decis. Process. 47, 270–288 (1990)

18. Krakhardt, D., Stern, R.N.: Informal networks and organizational crises: an experimental simulation. Soc. Psychol. Q. 51, 123–140 (1988)

19. Leskovec, J., McGlohon, M., Faloutsos, C., Glance, N. and Hurst, M., 2007, April. Patterns of cascading behavior in large blog graphs. In Proceedings of the 2007 SIAM international conference on data mining (pp. 551-556). Society for Industrial and Applied Mathematics.

20. Li, K., Zhang, L. and Huang, H., 2018. Social influence analysis: models, methods, and evaluation. Engineering, 4(1), pp.40-46.

21. Myers, D.G., 1982. Polarizing effects of social interaction. Group decision making, 125, pp.137-138.

22. Robins, G., 2009. Understanding individual behaviors within covert networks: the interplay of individual qualities, psychological predispositions, and network effects. Trends in Organized Crime, 12(2), pp.166-187.

23. Salanscik, G.R., Pfeffer, J.: A social information processing approach to job attitudes and task design. Adm. Sci. Q. 23, 224–253 (1978)

24. Takács, K., Flache, A. and Mäs, M., 2016. Discrepancy and disliking do not induce negative opinion shifts. PloS one, 11(6), p.e0157948.

25. Wood, W., 2000. Attitude change: Persuasion and social influence. Annual review of psychology, 51.

26. White, H.C., Boorman, S.A. and Breiger, R.L., 1976. Social structure from multiple networks. I. Blockmodels of roles and positions. American journal of sociology, 81(4), pp.730-780.

27. Zagenczyk, T.J., Murrell, J.: It is better to receive than to give: advice network effects on job and AQ3 work-unit attachment. J. Bus. Psychol. 24(2), 139–152 (2014)

28. Internet and other open resources:
    - Types of Social Influences and Their Effect on Behavior. https://online.rider.edu/blog/types-of-social-influence/
    - The Milgram Shock Experiment. https://www.simplypsychology.org/milgram.html

Acknowledgements

In preparing this lecture, I have relied on knowledge and materials accumulated over the course of my long career. I have learned networks from many different courses. In preparation of materials for this course, I have relied on what I have learned, and on some of the information available freely on the internet and elsewhere.

My special gratitude goes to:

- Professor Stanley Wasserman, the founder of the International Laboratory for Applied Network Research (home to this course), who taught me the networks. I have referred to his materials, andcontinue to do so, on a regular basis. https://psych.indiana.edu/directory/faculty/wasserman-stanley.html
- Professor Vladimir Batagelj, current scientific supervisor of the Laboratory and a faculty member of this program. I relied on some of his work in preparing this lecture. https://www.hse.ru/en/org/persons/222862219