SNA Basics X

Structural Equivalence

@ Sean F. Everton

Place a header at the top of your script that tells you what you called it, what it accomplishes, etc.

```
##################################################
# What: Structural Equivalence in R
# File: snab10.R
# Created: 02.28.29
# Revised: 07.12.18
##################################################
```

Data

For this exercise, we'll use two datasets. We begin by examining a relatively small network that Wasserman and Faust (1994) use because it is easier to illustrate certain blockmodeling techniques with smaller datasets than with larger ones. We'll then move to the Anabaptist Leadership network, which we've used before. It includes 55 Anabaptist leaders and 12 prominent Protestant Reformation leaders who had contact with and influenced some of the Anabaptist leaders included in this dataset. These network data build upon a smaller dataset (Matthews et al., 2013) that did not include some leading Anabaptist leaders, such as Menno Simons, who is seen as the "founder" of the Amish and Mennonites.

Note: This lab is heavily indebted to a practicum written by Phil Murphy (one of my co-authors) and Brendan Knapp for Phil Murphy's introductory to social network analysis at Middlebury Institute of International Studies in Monterey, CA.

Note We'll use more than the *statnet* and *igraph* libraries for this exercise. We'll also use the *concoR* and *blockmodel* libraries. Because of this, this lab is organized a bit differently from the previous ones. Instead of beginning with *statnet* and then moving to *igraph*, we'll exmaine how to estimate structural equivalence three different ways:

- Euclidian Distance
- CONCOR
- Optimization

Setup

Clear the workspace each time before beginning.

```
rm(list=ls())
```

Set your working directory to where the data are, so you don't have to include the entire path when importing and exporting data, files, etc.

```
setwd("~/Dropbox/Networks and Religion (Book)/Website/Labs/SNA Basics 10")
```

Load initial libraries

Load the statnet libraries

```
library(sna)
```

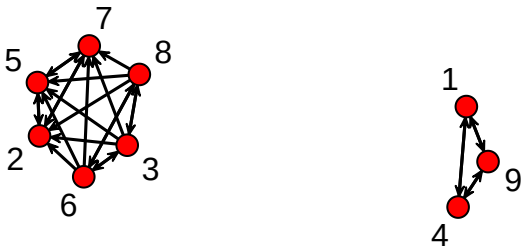
```
## Loading required package: statnet.common
##
## Attaching package: 'statnet.common'
## The following object is masked from 'package:base':
##
##     order
## Loading required package: network
## network: Classes for Relational Data
## Version 1.13.0.1 created on 2015-08-31.
## copyright (c) 2005, 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.
## sna: Tools for Social Network Analysis
## Version 2.4 created on 2016-07-23.
## copyright (c) 2005, Carter T. Butts, University of California-Irvine
##  For citation information, type citation("sna").
##  Type help(package="sna") to get started.
library(network)
```

Import the Wasserman and Faust network, which is stored as a csv file.

```
wfse.mat <- as.matrix(read.csv(("WFSE.csv"),header=TRUE,row.names=1,check.names=FALSE))
wfse.net <- as.network(wfse.mat,directed=TRUE)
wfse.net
##  Network attributes:
##   vertices = 9
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##   total edges= 27
##     missing edges= 0
##     non-missing edges= 27
##
##  Vertex attribute names:
##     vertex.names
##
## No edge attributes
```

Plot (this looks correct)

```
gplot(wfse.net,usearrows=TRUE,arrowhead.cex=.5,label=network.vertex.names(wfse.net))
```



Euclidean Distance

`Euclidean distance` is a method of calculating similarities by comparing common distances from some nodes to other nodes. Euclidean distance algorithms take network data and calculate a set of points in n-dimensional space, such that the distances between them correspond as closely as possible to the input proximities. Euclidean distance differs from distance in graph theory. The latter measures the distance between two actors in terms of the number of lines in the path that connects the two points (i.e., path distance), while the former measures the distance between two actors in terms of the most direct route between them (i.e., as the crow flies).

The first command (equiv.clust) calculates the euclidean distance between the various nodes. Because the network is directed, we have to tell R that the network is a digraph (directed graph); this is actually the default, but we show it here so that readers are aware of it. If you're analyzing an undirected graph, then we give the command, "mode = 'graph' ".

The second command creates the blockmodel. Here we use the "k = 3"" command to tell R to "cut" the network into three blocks. There is also an "h" command, which we can use to tell R how many times to "split" the network into blocks.

```
wfse.eq<-equiv.clust(wfse.net,method="euclidean",mode="digraph")
wfse.bl <- blockmodel(wfse.net,wfse.eq,k=3,mode="graph")
```

The next two commands list the actor labels and then the blocks to which they belong.

```
wfse.bl$plabels
## [1] "9" "1" "4" "7" "2" "5" "8" "3" "6"
wfse.bl$block.membership
## [1] 1 1 1 2 2 2 3 3 3
```

The first command displays the blockmodel; the second combines the information from above with the model iteslf

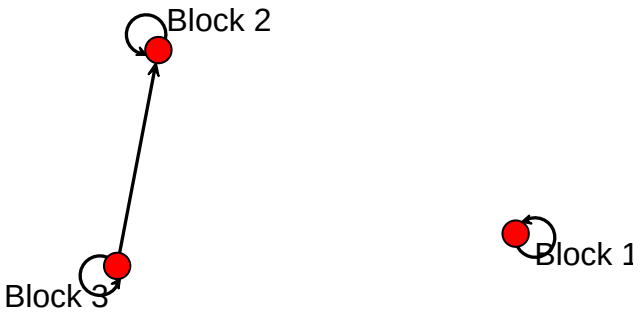
```
wfse.bl$block.model
##         Block 1 Block 2 Block 3
## Block 1       1       0       0
## Block 2       0       1       0
## Block 3       0       1       1
wfse.bl
##
## Network Blockmodel:
##
## Block membership:
##
## 1 2 3 4 5 6 7 8 9
## 1 2 3 1 2 3 2 3 1
```

```
##
## Reduced form blockmodel:
##
##   1 2 3 4 5 6 7 8 9
##         Block 1 Block 2 Block 3
## Block 1       1       0       0
## Block 2       0       1       0
## Block 3       0       1       1
```

The first command converts the blockmodel to matrix; the next one converts it to a network object, and the third visualizes the matrix with loops displayed.

As we can see the first block only interacts with itself, the second interacts and itself and sends ties to the third block, and the third only interacts with itself and receives ties from second block.

```
wfse.blmat <- as.matrix(wfse.bl$block.model)
wfse.blnet <- as.network(wfse.bl$block.model)
gplot(wfse.blmat,usearrows=TRUE,arrowhead.cex=.5,label=network.vertex.names(wfse.blnet),
      diag=TRUE,loop.cex=1.5)
```



As you may have guessed, the density within and between blocks seldom exactly equals either 1.00 or 0.00 (in other words, actors are seldom perfectly similar or dissimilar), so we generally have to choose some sort of criterion that distinguishes "one-blocks" (sometimes referred to as complete blocks) from "zero-blocks" (a.k.a., null blocks). We'll examine how to do this below with the Anabaptist Leadership network.

One last thing before moving on. The initial command "calls" a distance function ("sedist") to calculate the euclidean distance between the nodes. If we want to see what the distances are, we can use it.

```
sedist(wfse.net,method="euclidean",mode="digraph")
##           [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
##  [1,] 0.000000 3.316625 3.316625 0.000000 3.316625 3.316625 3.316625
##  [2,] 3.316625 0.000000 2.000000 3.316625 0.000000 2.000000 0.000000
##  [3,] 3.316625 2.000000 0.000000 3.316625 2.000000 0.000000 2.000000
##  [4,] 0.000000 3.316625 3.316625 0.000000 3.316625 3.316625 3.316625
##  [5,] 3.316625 0.000000 2.000000 3.316625 0.000000 2.000000 0.000000
##  [6,] 3.316625 2.000000 0.000000 3.316625 2.000000 0.000000 2.000000
##  [7,] 3.316625 0.000000 2.000000 3.316625 0.000000 2.000000 0.000000
##  [8,] 3.316625 2.000000 0.000000 3.316625 2.000000 0.000000 2.000000
##  [9,] 0.000000 3.316625 3.316625 0.000000 3.316625 3.316625 3.316625
##           [,8]     [,9]
##  [1,] 3.316625 0.000000
##  [2,] 2.000000 3.316625
##  [3,] 0.000000 3.316625
##  [4,] 3.316625 0.000000
##  [5,] 2.000000 3.316625
##  [6,] 0.000000 3.316625
```

```
##  [7,] 2.000000 3.316625
##  [8,] 0.000000 3.316625
##  [9,] 3.316625 0.000000
```

Note that these distances are different from the ones we get using UCINET because with the latter we used geodesic distance to calculate distances (see the companion lab). Nevertheless, we basically get the same results. A distance of 0.0 indicates "perfect similarity," 2.0 is the next level, and 3.316625 is the greatest distance (in this matrix).

CONCOR

`CONCOR` stands for "CONvergence of iterated CORrelations." It is based on the insight that repeated calculation of correlations between a matrix's rows (or columns) eventually results in correlation matrix consisting of only +1.00's and -1.00's. The algorithm was developed by Ron Breiger, Scott Boorman, and Phipps Arabie. If you are interested, you can check out their original (1975) paper: "An Algorithm for Clustering Relational Data with Applications to Social Network Analysis and Comparison with Multidimensional Scaling. *Journal of Mathematical Psychology*, 12: 328–383.

The original algoirthm was written in Fortran. Since then, Adam Slez has rewritten it in R. You can find out more by checking his website: Bad Hessian. He has not committed his `concoR` package to CRAN, so we have to install it from his github site. We only have to do this once. However, we have to install `devtools` if we haven't already.

```
install.packages("devtools", dependencies = TRUE)
```

Load devtools and install concoR

```
# Load devtools
library(devtools)

# Install concoR
devtools::install_github("aslez/concoR")
```

Load the concoR library

```
library(concoR)
```

`CONCOR` requires a matrix, or stack of matrices to make its calculations, so we'll read the Anabaptist Leadership network into R as a matrix although we'll save it as a network object too, which will help with

```
anabaptist.mat <- as.matrix(read.csv(("Anabaptist Leaders.csv"),header=TRUE,
                            row.names=1,check.names=FALSE))
anabaptist.net <- as.network(anabaptist.mat,directed=FALSE)
```

Let's run the `concoR` algorithm to identify the various structrual equivalence "blocks." **Note**: The `list()` command in `concor_hca` is necessary if we use only one matrix (like we do here). The program was developed to work with arrays (lists of matrices), so it doesn't work well with single matrices without this command.

The "p=3" command tells R how many times to partition (split) the matrix. Here we'll follow what we did in the UCINET version of this lab, and split it three times (yielding 8 blocks).

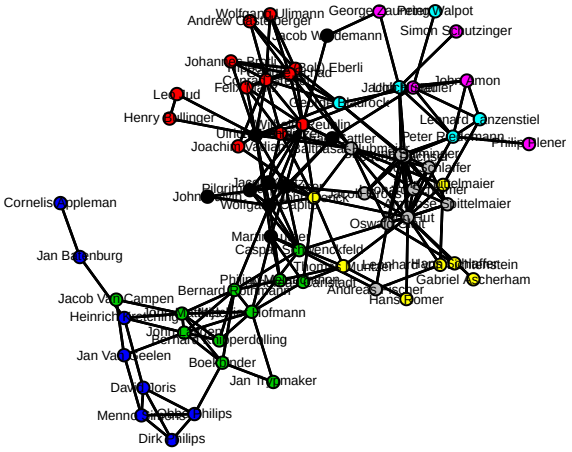
```
anabaptist.blks <- concor_hca(list(anabaptist.mat),p=3)
```

The *concoR* package generates a list of vertex names, as well as the "blocks" or classes into which they have been partitioned. You can view the partition with the view command

```
View(anabaptist.blks)
```

We can also use it to color the nodes of the original network

```
anabaptist.par <- anabaptist.blks$block
gplot(anabaptist.net,label=network.vertex.names(anabaptist.net),label.pos= 5,
                vertex.col=anabaptist.par,label.col="black",label.cex=0.4,
                usearrows=FALSE)
```



We can create a blockmodel using *statnet's* `blockmodel` function. The "reduced form blockmodel" presents the density of each block in the matrix.

Note: We are using network data that was formatted for *statnet* here (`anabaptist.net`)

```
anabaptist.bl <- blockmodel(anabaptist.net,anabaptist.blks$block)
anabaptist.bl
```

```
##
## Network Blockmodel:
##
## Block membership:
##
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
##  1  1  1  2  2  2  5  2  2  2  2  2  2  2  7  7  3  8  8  1  5  1  8  8  3
## 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
##  7  7  3  3  3  3  8  8  8  8  5  7  6  1  6  6  5  5  1  7  4  4  8  2  3
## 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67
##  6  8  2  4  6  4  4  7  4  3  3  3  3  1  1  4  4
##
## Reduced form blockmodel:
##
##   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
##            Block 1    Block 2    Block 3    Block 4    Block 5    Block 6
## Block 1 0.33333333 0.13888889 0.10101010 0.00000000 0.02222222 0.04444444
## Block 2 0.13888889 0.34848485 0.00000000 0.00000000 0.06666667 0.00000000
## Block 3 0.10101010 0.00000000 0.36363636 0.05681818 0.00000000 0.00000000
## Block 4 0.00000000 0.00000000 0.05681818 0.35714286 0.00000000 0.00000000
## Block 5 0.02222222 0.06666667 0.00000000 0.00000000 0.20000000 0.44000000
## Block 6 0.04444444 0.00000000 0.00000000 0.00000000 0.44000000 0.10000000
## Block 7 0.06349206 0.03571429 0.03896104 0.00000000 0.02857143 0.00000000
## Block 8 0.02222222 0.03333333 0.01818182 0.00000000 0.14000000 0.00000000
##            Block 7    Block 8
## Block 1 0.06349206 0.02222222
## Block 2 0.03571429 0.03333333
```

```
## Block 3 0.03896104 0.01818182
## Block 4 0.00000000 0.00000000
## Block 5 0.02857143 0.14000000
## Block 6 0.00000000 0.00000000
## Block 7 0.14285714 0.15714286
## Block 8 0.15714286 0.60000000
```

As we noted earlier, the density within and between blocks seldom exactly equals either 1.00 or 0.00 (in other words, actors are seldom perfectly similar or dissimilar), so we generally have to choose some sort of criterion that distinguishes "one-blocks" (i.e., complete blocks) from "zero-blocks" (a.k.a., null blocks).

How do we do this? There are a number of different approaches. The zero block (or lean fit) approach only classifies a block as a null block if its density equals 0.00 (i.e., there's a complete absence of ties), and all of the rest of the blocks are classified as one blocks. When we do this, half of the blocks are classified as complete blocks and half are classified as null blocks (see the UCINET/Pajek Lab). By contrast, if we use the one block approach, which only classifies a block as a complete block if its density equals 1.00, w end up with a final image matrix full of zeros and no ones, which in this case (and in most cases) is unhelpful.

Probably the most common approach for distinguishing complete blocks from null blocks is to set a threshold such that if a particular block's density is greater than or equal to that threshold, it is classified as a complete block, and if it is not, it is classified as a null block.

To use density as a cutoff for complete and zero (null) blocks, we first have to compute density (which we already did in SNA Basics #5).

```
anabaptist.den <- gden(anabaptist.net,mode="graph")
anabaptist.den
## [1] 0.08276798
```

As you can see, the network's density = 0.083, and with the following commands, we can dichotomize the network by assigning 1 to blocks whose density is greater than or equal to 0.083 and 0 to blocks whose density is less that 0.083.

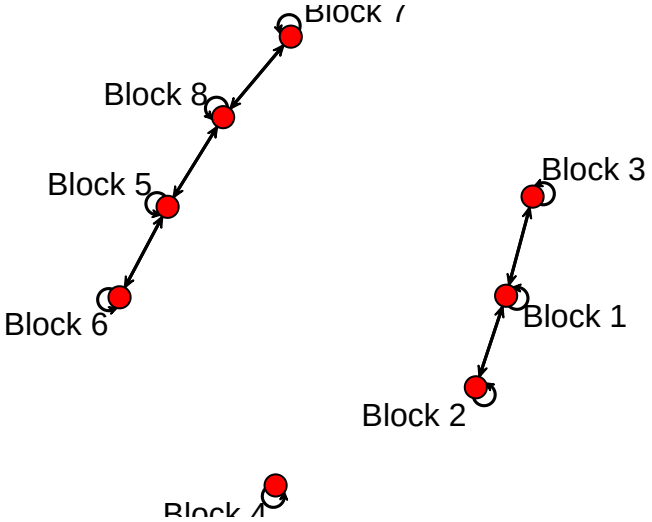
```
anabaptist.blmat <- as.matrix(anabaptist.bl$block.model)
anabaptist.blmat
##            Block 1    Block 2    Block 3    Block 4    Block 5    Block 6
## Block 1 0.33333333 0.13888889 0.10101010 0.00000000 0.02222222 0.04444444
## Block 2 0.13888889 0.34848485 0.00000000 0.00000000 0.06666667 0.00000000
## Block 3 0.10101010 0.00000000 0.36363636 0.05681818 0.00000000 0.00000000
## Block 4 0.00000000 0.00000000 0.05681818 0.35714286 0.00000000 0.00000000
## Block 5 0.02222222 0.06666667 0.00000000 0.00000000 0.20000000 0.44000000
## Block 6 0.04444444 0.00000000 0.00000000 0.00000000 0.44000000 0.10000000
## Block 7 0.06349206 0.03571429 0.03896104 0.00000000 0.02857143 0.00000000
## Block 8 0.02222222 0.03333333 0.01818182 0.00000000 0.14000000 0.00000000
##            Block 7    Block 8
## Block 1 0.06349206 0.02222222
## Block 2 0.03571429 0.03333333
## Block 3 0.03896104 0.01818182
## Block 4 0.00000000 0.00000000
## Block 5 0.02857143 0.14000000
## Block 6 0.00000000 0.00000000
## Block 7 0.14285714 0.15714286
## Block 8 0.15714286 0.60000000

anabaptist.blmat[anabaptist.blmat < 0.083] <- 0 # assign 0s for blocks less than 0.083
anabaptist.blmat[anabaptist.blmat >= 0.083] <- 1 # assign 1s for blocks greater than 0.083
```

```
anabaptist.blmat
##         Block 1 Block 2 Block 3 Block 4 Block 5 Block 6 Block 7 Block 8
## Block 1       1       1       1       0       0       0       0       0
## Block 2       1       1       0       0       0       0       0       0
## Block 3       1       0       1       0       0       0       0       0
## Block 4       0       0       0       1       0       0       0       0
## Block 5       0       0       0       0       1       1       0       1
## Block 6       0       0       0       0       1       1       0       0
## Block 7       0       0       0       0       0       0       1       1
## Block 8       0       0       0       0       1       0       1       1
```

Now, let's plot the reduced network, following the same steps that we used above with the Wasserman and Faust network. Here, I've reduced the size of the loops from before (1 is actually the default, so all I really needed to do was eliminate the command).

```
anabaptist.blnet <- as.network(anabaptist.blmat)
gplot(anabaptist.blmat,usearrows=TRUE,arrowhead.cex=.5,
      label=network.vertex.names(anabaptist.blnet),diag=TRUE,loop.cex=1)
```



Optimization

The "optimization" approach is where you assign some number of random partitions and ask the algorithm to re-sort the network to a point where the various blocks contain a best fit to the network.

Fit is determined by an error score (the lower the better) that compares the final model with a perfect model. Recall that in the case of perfect structural equivalence, in a complete block all possible ties are present, while in a null block no ties are present. Consequently, with the optimization approach an error is considered to occur when a tie is present in a null block or a tie is missing in a complete block. The optimization approach continues until it can no longer lower the error score. That said, most of the time increasing the "number of blocks" reduces the error score, so it's probably best to begin with a "theory" as to the number of blocks, rather than continually increasing the number until the error score stops decreasing.

For more information on your options with this algorithm, and generalized blockmodeling in general, check out this article: Žiberna, Aleš (2007): Generalized Blockmodeling of Valued Networks. *Social Networks*, 29: 105-126.

Aleš Žiberna, the author of this article and a student of Vlado Batagelj (one of Pajek's developers), has written the `blockmodeling` package in R, which (unsurprisingly) mimics Pajek's optimization approach.

We'll use that now:

```
install.packages("blockmodeling",dependencies=TRUE)
```

Load the blockmodeling library

```
library(blockmodeling)
```

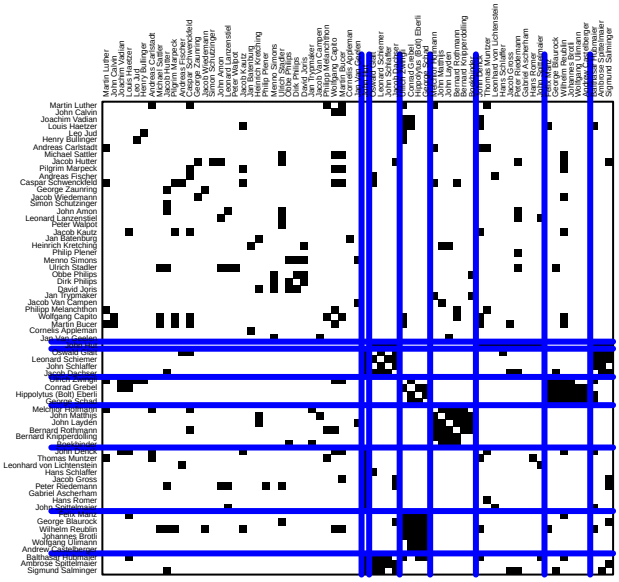
Here's the relevant optimization command; note that we request that we want complete and null blocks (as in Pajek) and that the matrix is binary (approaches = "bin"). We also ask for 500 repetitions in order to increase the possibility that we find the optimal solution.

```
anabaptist.opt <- optRandomParC(M=anabaptist.mat,k=8,rep=500,approaches="bin",
                                blocks=c("nul","com"))
##
##
## Starting optimization of the partiton 50 of 500 partitions.
##
##
## Starting optimization of the partiton 100 of 500 partitions.
##
##
## Starting optimization of the partiton 150 of 500 partitions.
##
##
## Starting optimization of the partiton 200 of 500 partitions.
##
##
## Starting optimization of the partiton 250 of 500 partitions.
##
##
## Starting optimization of the partiton 300 of 500 partitions.
##
##
## Starting optimization of the partiton 350 of 500 partitions.
##
##
## Starting optimization of the partiton 400 of 500 partitions.
##
##
## Starting optimization of the partiton 450 of 500 partitions.
##
##
## Starting optimization of the partiton 500 of 500 partitions.
##
##
## Optimization of all partitions completed
## 1 solution(s) with minimal error = 244 found.
```

Note that the optimization program found at least one solution; sometimes it will find more.

We can plot the network as a permuted matrix like this.

```
plot(anabaptist.opt,main="Permuted Matrix")
```



What information does the package generate? The command below tells us all of what the package generated (a lot).

```
ls(anabaptist.opt)
## [1] "best"          "call"          "checked.par"   "err"
## [5] "initial.param" "M"             "nIter"         "Random.seed"
```

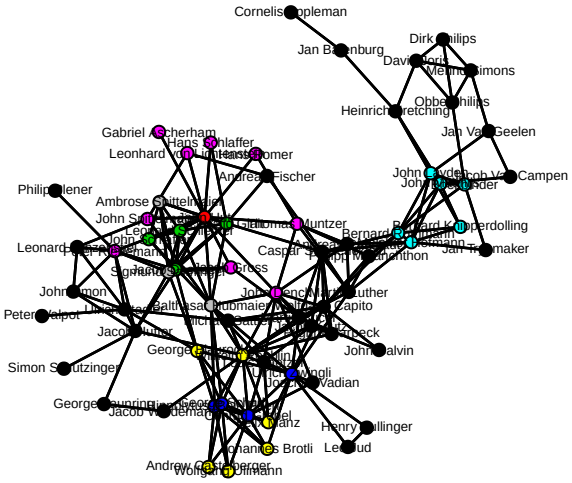
This command tells us where we can find the partitions of the solution(s): best1 and (maybe) best2, best3, etc.

```
ls(anabaptist.opt$best)
## [1] "best1"
```

We can also create vectors from the solution(s) to color the nodes of the original network.

```
anabaptist.best1 <- anabaptist.opt$best$best1$clu

coord <- gplot(anabaptist.net,label=network.vertex.names(anabaptist.net),label.pos= 5,
               vertex.col=anabaptist.best1,label.col="black",label.cex=0.4,
               usearrows=FALSE)
```



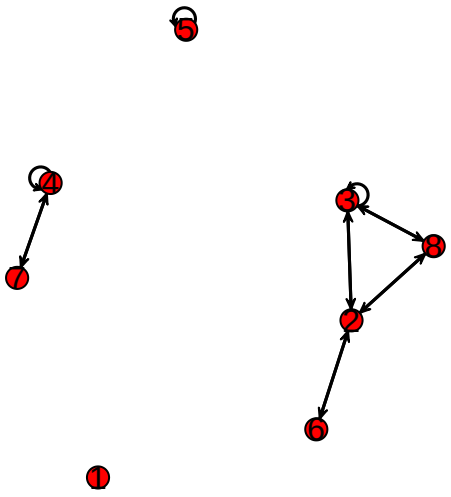
We can also create an image matrix from the results with the following commands:

```
anabaptist.optim <- IM(anabaptist.opt)
anabaptist.optim <- as.matrix(anabaptist.optim[1,,])
anabaptist.optim
##      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]
## [1,] "nul" "nul" "nul" "nul" "nul" "nul" "nul" "nul"
## [2,] "nul" "nul" "com" "nul" "nul" "com" "nul" "com"
## [3,] "nul" "com" "com" "nul" "nul" "nul" "nul" "com"
## [4,] "nul" "nul" "nul" "com" "nul" "nul" "com" "nul"
## [5,] "nul" "nul" "nul" "nul" "com" "nul" "nul" "nul"
## [6,] "nul" "com" "nul" "nul" "nul" "nul" "nul" "nul"
## [7,] "nul" "nul" "nul" "com" "nul" "nul" "nul" "nul"
## [8,] "nul" "com" "com" "nul" "nul" "nul" "nul" "nul"
```

We can change the "nul" and "com" to 0s and 1s and then plot the image matrix. Compare this with the permuted matrix above.

```
anabaptist.optim[anabaptist.optim == "nul"] <- 0
anabaptist.optim[anabaptist.optim == "com"] <- 1

anabaptist.optnet <- as.network(anabaptist.optim,loops=TRUE)
gplot(anabaptist.optnet,arrowhead.cex=.5,label=network.vertex.names(anabaptist.optnet),
      diag=TRUE,loop.cex=1,label.pos = 5)
```



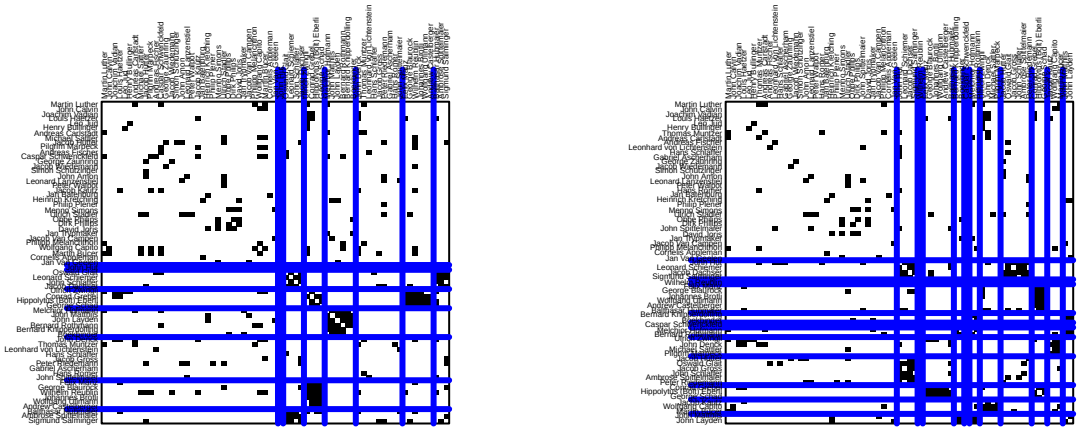
Above we asked for 8 partitions. However, as we noted, although it isn't a great strategy, we can keep running the optimization algorithm with different sized paritions until we find the best fit. Just for "fun," let's generate a second optimization blockmodel of 12 partitions and then compare it to the original.

```
anabaptist.opt2 <- optRandomParC(M=anabaptist.mat,k=12,rep=500,approaches="bin",
                                 blocks=c("nul","com"))
##
##
## Starting optimization of the partiton 50 of 500 partitions.
##
##
## Starting optimization of the partiton 100 of 500 partitions.
##
##
## Starting optimization of the partiton 150 of 500 partitions.
##
##
## Starting optimization of the partiton 200 of 500 partitions.
##
##
## Starting optimization of the partiton 250 of 500 partitions.
##
##
## Starting optimization of the partiton 300 of 500 partitions.
##
##
## Starting optimization of the partiton 350 of 500 partitions.
##
##
## Starting optimization of the partiton 400 of 500 partitions.
##
##
## Starting optimization of the partiton 450 of 500 partitions.
##
##
## Starting optimization of the partiton 500 of 500 partitions.
##
##
## Optimization of all partitions completed
## 1 solution(s) with minimal error = 218 found.

par(mfrow=c(1,2)) # set the plot window for one row and two columns
plot(anabaptist.opt, main="")
  title("Eight Block Partition")
plot(anabaptist.opt2, main="")
  title("Twelve Block Partition")
```

Eight Block Partition

Twelve Block Partition



```
par(mfrow=c(1,1)) # reset the plot window back to one row and one column
```

As you can see the error score is lower, but it's unclear from the permuted matrix whether the 12-partition solution is really an improvement.

Now, let's plot a few of these in *igraph*

Begin by loading the *igraph* and *intergraph* libraries and detaching the *sna* library

```
library(igraph)
library(intergraph)
detach("package:sna",unload=TRUE)
```

Next, transform the networks from *statnet* objects to *igraph* objects

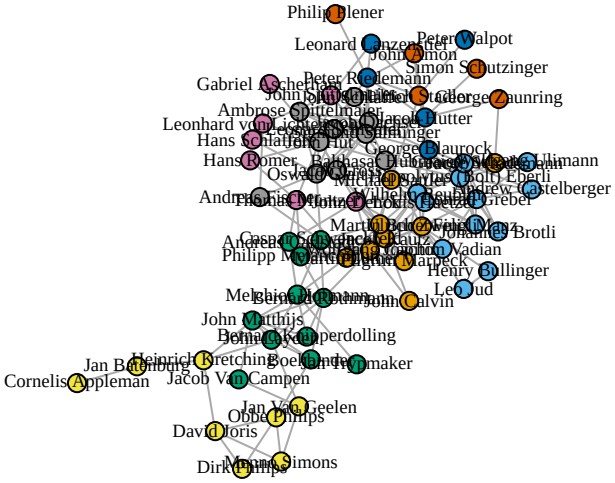
```
anabaptist.ig <- asIgraph(anabaptist.net)
anabaptist.blig <- asIgraph(anabaptist.blnet)
anabaptist.optig <- asIgraph(anabaptist.optnet)
```

Assign the vertex names as labels

```
V(anabaptist.ig)$label = V(anabaptist.ig)$vertex.names
V(anabaptist.blig)$label = V(anabaptist.blig)$vertex.names
V(anabaptist.optig)$label = V(anabaptist.optig)$vertex.names
```

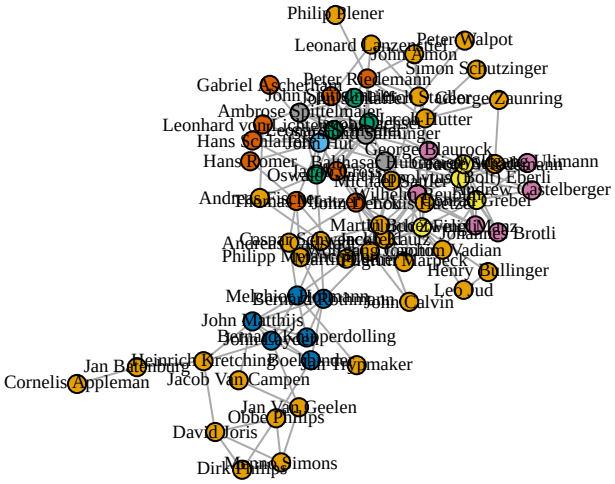
Visualize the network using the CONCOR partition; we'll save the coordinates so that we can compare it to the optimization graph.

```
anabaptist.ig$layout <- layout.kamada.kawai(anabaptist.ig)
plot(anabaptist.ig,vertex.label.cex=.6,vertex.label.color="black",edge.arrow.mode=0,
     vertex.color=anabaptist.par,vertex.size=8)
```



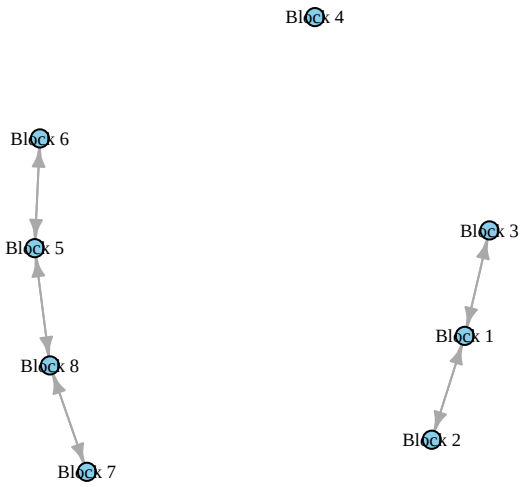
Now, with the optimization paritition

```
plot(anabaptist.ig,vertex.label.cex=.6,vertex.label.color="black",edge.arrow.mode=0,
     vertex.color=anabaptist.best1,vertex.size=8)
```

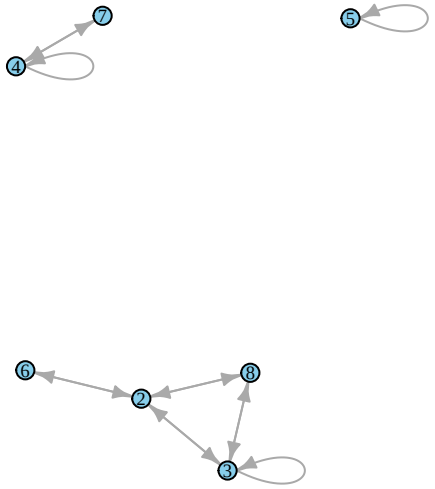


Let's look at the two reduced blockmodels; first CONCOR, then Optimization

```
coords1 <- layout.kamada.kawai(anabaptist.blig)
plot(anabaptist.blig,vertex.label.cex=.6,vertex.label.color="black",edge.arrow.size=.5,
     vertex.color="Sky Blue",vertex.size=8)
```



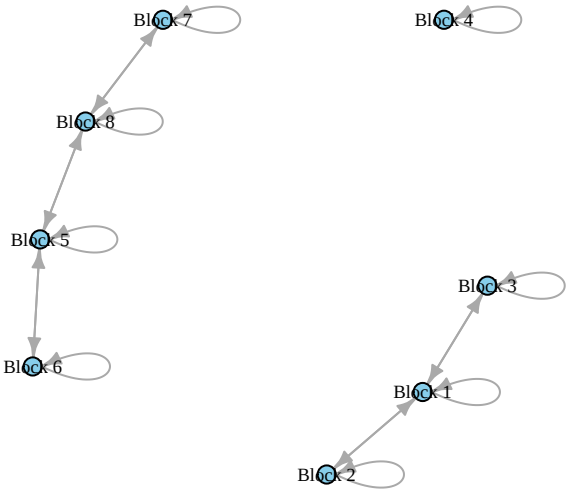

```
coords2 <- layout.kamada.kawai(anabaptist.optig)
plot(anabaptist.optig,vertex.label.cex=.6,vertex.label.color="black",edge.arrow.size=.5,
     vertex.color="Sky Blue",vertex.size=8)
```



If we want to see the loops of the first blockmodel, we need to convert the blockmodel matrix to an *igraph* object.

```
anabaptist.bligmat <- graph.adjacency(anabaptist.blmat)

plot(anabaptist.bligmat,vertex.label.cex=.6,vertex.label.color="black",edge.arrow.size=.5,
     vertex.color="Sky Blue",vertex.size=8,layout=coords1)
```



That's all for now