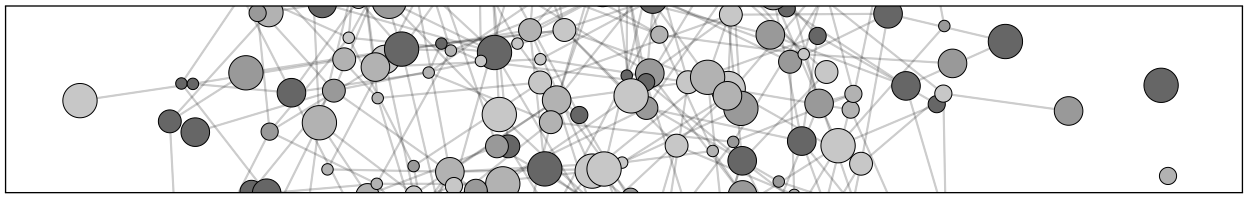
An Introduction to Network Analysis with `R` and `statnet`

Sunbelt XXXII Workshop Series · *March 13, 2012*



Workshop Presenters:

Ryan M. Acton
Assistant Professor
Department of Sociology
University of Massachusetts Amherst
racton@soc.umass.edu

Lorien Jasny
Post-doctoral Researcher
Department of Environmental Science and Policy
University of California, Davis
ljasny@uci.edu

Contents

1 Getting Started

1.1 Warning

The document is best viewed on a Mac using Acrobat Reader!

1.2 Basic resources

- `R` website: `http://www.r-project.org`
- Helpful `R` tutorials: `http://cran.r-project.org/other-docs.html`
- `statnet` website: `http://statnet.org`
- `statnet` help: `statnet_help@statnet.org`
- Workshop web site: `http://sunbelt2012.statnet.org/`

1.3 Typographical conventions

Text in a grey box, as below, represents code for you to type (or copy & paste into `R`).

```
gplot(g,gmode="graph",vertex.cex=1.5)
```

The lighter text inside the grey box represents comments. Comments are ignored by R, but are typically useful for humans.

All other text is meant to provide background or guidance.

1.4 Download and install the latest version of R

1. Go to http://cran.r-project.org/, and select Mirrors from the left-hand menu.
2. Select a location near you.
3. From the "Download and Install R " section, select the link for your operating system.
4. Follow the instructions on the relevant page.
5. Note that you need to download only the base distribution, not the contributed packages.
6. Once you've downloaded the installation file, follow the instructions for installation.

1.5 Download and attaching statnet and associated packages

1. Open R.
2. Install the statnet installer. At the R cursor, type:

```
install.packages("statnet")
```

3. Now, and in the future, you can install/update statnet at any point using the installer that comes with statnet. The previous step is only necessary the first time you wish to use statnet but does not need to be repeated each time. At the R cursor, type:

```
update.packages("statnet")
```

   Follow the directions; feel free to say no to any optional packages, although we recommend saying yes. The first choice provided is to install all the required and optional packages.

4. Attach statnet to your R session by typing:

```
library(statnet)
```

1.6 Download and install supplemental packages

1.7 Set a specific working directory for this tutorial if you wish

1. If you are using Windows, go to File ¿ Change Dir and choose the directory.
2. On a Mac, go to Misc ¿ Change Working Directory and choose the directory.
3. Otherwise, use the setwd() command:

```
setwd("full.path.for.the.folder")
```

1.8 A few facts to remember about R

- R mostly runs through the command line or through batch files.
- Everything in R is an object, including data, output, functions—everything.
- Objects that are created during a session are saved in the "global environment" by default, which is stored as a single file (".RData ") in the working directory.
- R is case sensitive.
- R comes with a set of pre-loaded functions. Others can be added by downloading from the R project website. Downloaded packages to be used must be attached using the library() command during any R session in which they are to be used. For instance:

```
install.packages("coda")    # install
    package from CRAN
library(coda)      # attach installed
    package
```

2 A Brief R Tutorial

2.1 Introduction to basic R syntax

```
1 + 3       # evaluation
```

```
a <- 3       # assignment
a        # evaluation
```

```
a<-3        # spacing does not matter
a <- 3      # spacing does not matter
```

```
sqrt(a)      # use the square root function
b <- sqrt(a)      # use function and save result
b
```

```
c        # evaluate something that is not there
```

```
a == b       # is a equivalent to b?
a != b       # is a not equal to b?
```

```
ls() # list objects in the global environment
```

```
help.start() # get help with R generally
?sqrt       # get specific help for a function
??sqrt       # looking for help pertaining to sqrt
```

```
apropos("sq")      # it's on the tip of my tongue...
```

```
rm(a)       # remove a single object
rm(list=ls())      # remove everything from the environment
```

2.2 Vectors and matrices in R

Creating vectors using the "combine" operator

```
a <- c(1,3,5)      # create a vector by combining values
a
a[2]        # select the second element
b <- c("one","three","five")   # also works with strings
b
b[2]
a <- c(a,a)      # can apply recursively
a
a <- c(a,b)      # mixing types---what happens?
a        # all converted to the same type
```

Sequences and replication

```
a <- seq(from=1,to=5,by=1)    # from 1 to 5 the slow way
b <- 1:5      # a shortcut!
a==b        # all TRUE
rep(1,times=5)     # a lot of ones
rep(1:5,times=2)     # repeat an entire sequence
rep(1:5,each=2)     # same, but element-wise
rep(1:5,times=5:1)     # can vary the count of each element
```

Any, all, and which (with vectors)

```
a <- 1:5      # create a vector
a>2        # some TRUE, some FALSE
any(a>2)      # are any elements TRUE?
all(a>2)      # are all elements TRUE?
which(a>2)      # which indices are TRUE?
```

From vectors to matrices

```
a <- matrix(data=1:25, nrow=5, ncol=5)  # create a matrix the "formal" way
a
a[1,2]       # select a matrix element (two dimensions)
a[1,]       # just the first row
all(a[1,]==a[1,1:5])    # show the equivalence
a[,2]       # can also perform for columns
a[2:3,3:5]      # select submatrices
a[-1,]       # nice trick: negative numbers omit cells!
a[-2,-2]      # get rid of row two, column two
```

```
b <- cbind(1:5,1:5)     # another way to create matrices
b
d <- rbind(1:5,1:5)     # can perform with rows, too
d
cbind(b,d)      # no go: must have compatible dimensions!
dim(b)       # what were those dimensions, anyway?
dim(d)
NROW(b)
NCOL(b)
cbind(b,b)      # combining two matrices
```

```
t(b)        # can transpose b
cbind(t(b),d)      # now it works
rbind(t(b),d)      # now it works
```

2.3 Element-wise operations

Most arithmetic operators are applied element-wise

```
a <- 1:5
a + 1       # addition
a * 2       # multiplication
a / 3       # division
a - 4       # subtraction
a ^ 5       # you get the idea...
```

```
a + a       # also works on pairs of vectors
a * a
a + 1:6      # problem: need same length
```

```
a <- rbind(1:5,2:6)     # same principles apply to matrices
b <- rbind(3:7,4:8)
a + b
a / b
```

```
a %*% t(b)      # matrix multiplication
```

Logical operators (generally) work like arithmetic ones

```
a > 0       # each value greater than zero?
a == b       # corresponding values equivalent?
a != b       # corresponding values not equivalent?
!(a == b)      # same as above
(a>2) | (b>4)      # the OR operator
(a>2) & (b>4)      # the AND operator
(a>2) || (b>4)     # beware the "double-pipe"!
(a>2) && (b>4)     # (and the "double-ampersand"!)
```

Ditto for many other basic transformations

```
log(a)
exp(b)
sqrt(a+b)      # note that we can nest statements!
log((sqrt(a+b)+a)*b)    # as recursive as we wanna be
```

2.4 Data frames

Build a data frame from scratch, containing three columns of data

```
d <- data.frame(income=1:5,sane=c(T,T,T,T,F),name=LETTERS[1:5])
d
d[1,2]       # acts a lot like a matrix!
d[,1]*5
d[-1,]
d$sane       # can use dollar sign notation
d$sane[3]<-FALSE     # making changes
d
d[2,3]       # shows factors for string values
d$name <- LETTERS[1:5]    # eliminate evil factors by overwriting
d[2,3]
```

If you want to do without factors

```
d <- data.frame(income=1:5,sane=c(T,T,T,T,F),name=LETTERS[1:5],stringsAsFactors=FALSE)
d
```

```
d <- as.data.frame(cbind(1:5,2:6))  # can create from matrices
d
is.data.frame(d)     # how can we tell it's not a matrix?
is.matrix(d)      # the truth comes out
```

2.5 Finding built-in data sets

Many packages have built-in data for testing and educational purposes

```
data()       # lists them all
?USArrests      # get help on a data set
data(USArrests)     # load the data set
USArrests      # view the object
```

2.6 Elementary visualization

R's workhorse is the "plot" command

```
plot(USArrests$Murder,USArrests$UrbanPop)  # using dollar sign notation
plot(USArrests$Murder,USArrests$UrbanPop,log="xy") # log-log scale
```

Adding plot title and axis labels

```
plot(USArrests$Murder,USArrests$Assault,xlab="Murder",ylab="Assault",main="USArrests")
```

Can also add text

```
plot(USArrests$Murder,USArrests$Assault,xlab="Murder",ylab="Assault", main="USArrests",type="n")
text(USArrests$Murder,USArrests$Assault,rownames(USArrests))
```

Histograms and boxplots are often helpful

```
hist(USArrests$Murder)
boxplot(USArrests)
```

3 network Objects: Importing, Exploring, and Manipulating Network Data

3.1 Built-in Network Datasets

```
library(network) # Make sure that network package is loaded
data(package="network") # List available datasets in network package
data(flo) # Load a built-in data set; see ?flo for more
flo # Examine the flo adjacency matrix
```

For more information...

```
?data
```

```
?flo
```

3.2 Importing Relational Data

Be sure to be in the directory where you stored the data for the workshop. If you've not set the working directory, you must do so now. See Section 1.7 for how to do this.

```
list.files()      # Check what's in the working directory
```

Read an adjacency matrix (R stores it as a data frame by default)

```
relations <- read.csv("relationalData.csv",header=FALSE,stringsAsFactors=FALSE)
relations
```

Here's a case where matrix format is preferred

```
relations <- as.matrix(relations)  # convert to matrix format
```

Read in some vertex attribute data (okay to leave it as a data frame)

```
nodeInfo <- read.csv("vertexAttributes.csv",header=TRUE,stringsAsFactors=FALSE)
nodeInfo
```

Since our relational data has no row/column names, let's set them now

```
rownames(relations) <- nodeInfo$name
colnames(relations) <- nodeInfo$name
relations
```

For more information...

```
?list.files
?read.csv
```

```
?as.matrix
?rownames
```

```
?colnames
```

3.3 Creating `network` Objects

```
nrelations<-network(relations,directed=FALSE)  # Create a network object based on flo
nrelations       # Get a quick description of the data
nempty <- network.initialize(5)    # Create an empty graph with 5 vertices
nempty        # Compare with nrelations
```

For more information...

```
?network
```

```
?as.network.matrix
```

3.4 Description and Visualization

```
summary(nrelations)       # Get an overall summary
network.dyadcount(nrelations)     # How many dyads in nflo?
network.edgecount(nrelations)     # How many edges are present?
network.size(nrelations)      # How large is the network?
as.sociomatrix(nrelations)      # Show it as a sociomatrix
nrelations[,]        # Another way to do it
```

```
plot(nrelations,displaylabels=T)    # Plot with names
plot(nrelations,displaylabels=T,mode="circle")  # A less useful layout...
library(sna)        # Load the sna library
gplot(nrelations)       # Requires sna
```

For more information...

```
?summary.network
?network.dyadcount
?network.edgecount
```

```
?as.sociomatrix
?as.matrix.network
?is.directed
```

```
?plot.network
```

3.5 Working With Edges

Adding edges

```
g <- network.initialize(5)     # Create an empty graph
g[1,2] <- 1       # Add an edge from 1 to 2
g[2,] <- 1       # Add edges from 2 to everyone else
g         # Examine the result
m <- matrix(0, nrow=5, ncol=5)    # Create an adjacency matrix
m[3,4:5] <- 1       # Add entries from 3 to 4 and 5
g[m>0] <- 1       # Add more entries
g
```

Delete edges

```
g[3,5] <- 0       # Remove the edge from 3 to 5
g         # Its gone!
g[,] <- 0       # Remove all edges
g         # Now, an empty graph
```

Testing adjacency

```
nrelations[4,5]      # Emma to Sarah?
nrelations[4,]      # Entire Emma row
nrelations[1:4,5:8]      # Subsets are possible
nrelations[-9,-9]      # Leaving Gil out of the picture
```

Setting edge values

```
m<-nrelations[,]      # copy over the relational structure
m[upper.tri(m)>0]<-rep(1:3,times=3)   # give different edge values
m<-symmetrize(m,rule="upper")
m
```

```
nrelations %e% "strength" <- m    # Add the valued ties back in
```

Retrieving edge values

```
list.edge.attributes(nrelations)    # See whats available
nrelations %e% "strength"     # Use the %e% operator
as.sociomatrix(nrelations,attrname="strength") # Can also do it this way
```

```
nrelations[,]      # Original tie structure is preserved
```

For more information...

```
?network.extraction
?add.edge
?delete.edges
```

```
?delete.vertices
?get.edges
?upper.tri
```

```
?symmetrize
```

3.6 Network and Vertex Attributes

Add some attributes

```
nrelations %v% "id" <- nodeInfo$id    # Add in our vertex attributes
nrelations %v% "age" <- nodeInfo$age
nrelations %v% "sex" <- nodeInfo$sex
nrelations %v% "handed" <- nodeInfo$handed
nrelations %v% "lastDocVisit" <- nodeInfo$lastDocVisit
```

Listing attributes

```
list.vertex.attributes(nrelations)    # List all vertex attributes
list.network.attributes(nrelations)    # List all network attributes
```

Retrieving attributes

```
nrelations %v% "age"      # Retrieve vertex ages
nrelations %v% "id"       # Retrieve vertex ids
```

For more information...

```
?attribute.methods
```

4 Classical Network Analysis with the sna Package for R

4.1 Getting started

```
library(sna) # Load the sna library
library(help="sna") # See a list of help pages for sna package
load("introduction_sunbelt_2012.Rdata") # Load supplemental workshop data
```

For more information...

```
?help.start
```

```
?library
```

```
?sna
```

4.2 Network data in sna

The sna package can handle network data in many forms. For instance, the function gden() calculates network density; we can use it on a network object, an adjacency matrix, a list of such matrices, etc.

```
data(flo)
flo # Adjacency form
gden(flo)
nflo<-network(flo,directed=FALSE) # Network form
gden(nflo)
```

The sna package also supports a special kind of matrix called an "sna edgelist." These are three-column matrices, each row of which represents an edge (via its sender, recipient, and value, respectively). These "sna edgelists" have special attributes that indicate their size, vertex names (if any), and bipartite status (if applicable).

```
eflo<-as.edgelist.sna(flo) # Coerce flo to an sna edgelist
eflo
attr(eflo,"n") # How many vertices are there?
attr(eflo,"vnames") # Are there vertex names?
as.sociomatrix.sna(eflo) # Can transform back w/ as.sociomatrix.sna
```

"sna edgelists" can be handy with large data sets (as a simple alternative to network objects). To make one, just add an "n" attribute to a valid three-column matrix!

```
mat<-cbind(rep(2,4),3:6,rep(1,4)) # Create edges from 2 to 3:6
attr(mat,"n")<-6 # Set total number of vertices to 6
mat
gden(mat) # Can now pass to sna routines
as.sociomatrix.sna(mat) # Can see in adjacency form
```

For more information...

```
?as.edgelist.sna
?as.sociomatrix.sna
```

```
?attr
?sna
```

4.3 Network visualization with `gplot()`

Plotting the data we imported earlier

```
gplot(nrelations)
gplot(nrelations,displaylabels=TRUE)  # This time with labels
```

Let's color the nodes in sex-stereotypic colors

```
nodeColors<-ifelse(nodeInfo$sex=="F","hotpink","dodgerblue")
gplot(relations,gmode="graph",displaylabels=TRUE,vertex.col=nodeColors)
```

Using data we just loaded in, plot the contiguity among nations in 1993 (from the Correlates of War (CoW)[1] project)

```
gplot(contig_1993) # The default visualization
gplot(contig_1993, usearrows=FALSE) # Turn off arrows manually
gplot(contig_1993, gmode="graph") # Can also tell gplot the data is undirected
```

We can add labels to the vertices

```
gplot(contig_1993, gmode="graph",displaylabels=TRUE,label.cex=0.5,label.col="blue")
```

Other ways to specify the labeling

```
gplot(contig_1993, gmode="graph",label=colnames(contig_1993[,]),label.cex=0.5,label.col="blue")
```

or

```
gplot(contig_1993,
    gmode="graph",label=network.vertex.names(contig_1993),label.cex=0.5,label.col="blue")
```

Here's an example of directed data—militarized interstate disputes (MIDs) for 1993

```
gplot(mids_1993,label.cex=0.5,label.col="blue",displaylabels=TRUE)
```

All those isolates can get in the way—we can suppress them using `displayisolates`

```
gplot(mids_1993,label.cex=0.5,label.col="blue",displaylabels=TRUE,displayisolates=FALSE)
```

The default layout algorithm is that of Frutchterman-Reingold (1991), can use others

[1] See `http://www.correlatesofwar.org/` for more information.

```
gplot(mids_1993,label.cex=0.5,label.col="blue",
displaylabels=TRUE,displayisolates=FALSE,mode="circle") # The infamous circle
```

or perhaps

```
gplot(mids_1993,label.cex=0.5,label.col="blue",
displaylabels=TRUE,displayisolates=FALSE,mode="mds")  # MDS of position similarity
```

When a layout is generated, the results can be saved for later reuse:

```
coords <- gplot(contig_1993) # Capture the magic of the moment
coords # Show the vertex coordinates
```

Saved (or *a priori*) layouts can be used via the `coord` argument

```
gplot(contig_1993,
    gmode="graph",label=colnames(contig_1993[,]),label.cex=0.5,label.col="blue",coord=coords)
```

When the default settings are insufficient, interactive mode allows for tweaking

```
coords <- gplot(contig_1993, interactive=TRUE) # Modify and save
gplot(contig_1993,coord=coords,displaylabels=TRUE,
gmode="graph",label.cex=0.5,label.col="blue") # Should reproduce the modified layout
```

For more information...

```
?gplot
```

```
?gplot.layout
```

4.4 Three-dimensional visualization with `gplot3d()` (requires the `rgl` package)

```
gplot3d(contig_1993,displaylabels=TRUE)     # Experience the future!
```

Other layouts are possible here, too:

```
gplot3d(contig_1993,displaylabels=TRUE,mode="kamadakawai")
```

For more information...

```
?gplot3d
```

```
?gplot3d.layout
```

4.5 Basic centrality indices: degree, betweenness, and closeness

We begin with the simplest case: degree centrality

```
degree(mids_1993) # Default: total degree
ideg <- degree(mids_1993, cmode="indegree") # Indegree for MIDs
odeg <- degree(mids_1993, cmode="outdegree") # Outdegree for MIDs
all(degree(mids_1993) == ideg+odeg) # In + out = total?
```

Once centrality scores are computed, we can handle them using standard `R` methods:

```
plot(ideg, odeg, type="n", xlab="Incoming MIDs", ylab="Outgoing MIDs") # Plot ideg by odeg
abline(0, 1, lty=3)
text(jitter(ideg), jitter(odeg), network.vertex.names(contig_1993), cex=0.75, col=2)
#Plot simple histograms of the degree distribution:
par(mfrow=c(2,2)) # Set up a 2x2 display
hist(ideg, xlab="Indegree", main="Indegree Distribution", prob=TRUE)
hist(odeg, xlab="Outdegree", main="Outdegree Distribution", prob=TRUE)
hist(ideg+odeg, xlab="Total Degree", main="Total Degree Distribution", prob=TRUE)
par(mfrow=c(1,1)) # Restore display
```

Centrality scores can also be used with other `sna` routines, e.g., `gplot()`

```
gplot(mids_1993, vertex.cex=(ideg+odeg)^0.5/2, vertex.sides=50,label.cex=0.4,
    vertex.col=rgb(odeg/max(odeg),0,ideg/max(ideg)),displaylabels=TRUE)
```

Betweenness and closeness are also popular measures

```
bet <- betweenness(contig_1993, gmode="graph") # Geographic betweenness
bet
gplot(contig_1993, vertex.cex=sqrt(bet)/25, gmode="graph") # Use w/gplot
clo <- closeness(contig_1993) # Geographic closeness
clo # A large world after all?
```

Can use `sna` routines to explore alternatives to the common measures...

```
closeness2 <- function(x){ # Create an alternate closeness function!
   geo <- 1/geodist(x)$gdist # Get the matrix of 1/geodesic distance
   diag(geo) <- 0 # Define self-ties as 0
   apply(geo, 1, sum) # Return sum(1/geodist) for each vertex
}
```

```
clo2 <- closeness2(contig_1993) # Use our new function on contiguity data
hist(clo2, xlab="Alt. Closeness", prob=TRUE) # Much better behaved!
cor(clo2, bet) # Correlate with betweenness
plot(clo2, bet) # Plot the bivariate relationship
all(clo2/185==closeness(contig_1993,cmode="suminvundir")) # Actually, we support this in sna!
```

For more information...

```
?betweenness
?bonpow
?closeness
```

```
?degree
?evcent
?graphcent
```

```
?infocent
?prestige
?stresscent
```

4.6 From centrality to centralization

```
centralization(mids_1993, degree, cmode="indegree") # Do MIDs concentrate?
centralization(contig_1993, evcent) # Eigenvector centralization
```

For more information...

```
?centralization
```

4.7 Elementary graph-level indices

```
gden(mids_1993) # Density
grecip(mids_1993) # Dyadic reciprocity
grecip(mids_1993, measure="edgewise") # Edgewise reciprocity
gtrans(mids_1993) # Transitivity
```

For more information...

```
?gden
```

```
?grecip
```

```
?gtrans
```

4.8 Subgraph census routines

```
dyad.census(mids_1993) # M,A,N counts
dyad.census(contig_1993) # No As in undirected graphs
triad.census(mids_1993) # Directed triad census
triad.census(contig_1993, mode="graph") # Undirected triad census
kpath.census(mids_1993, maxlen=6, tabulate.by.vertex=FALSE) # Count paths of length <=6
kcycle.census(mids_1993, maxlen=6, tabulate.by.vertex=FALSE) # Count cycles of length <=6
clique.census(mids_1993, tabulate.by.vertex=FALSE, enumerate=FALSE) # Find maximal cliques
```

Can also look at more detailed tabulation/co-membership for paths/cycles/cliques

```
kpath.census(mids_1993, maxlen=4) # Show tabulation by vertex
```

```
indirect <- kpath.census(mids_1993, maxlen=6, dyadic.tabulation="sum")$paths.bydyad
gplot(indirect,label.cex=0.4,vertex.cex=0.75,
displaylabels=TRUE,edge.col=rgb(0,0,0,0.25))  # Plot indirect MIDs
```

Component information can be obtained in various ways

```
components(mids_1993) # Strong component count
components(mids_1993, connected="weak") # Weak component count
cd <- component.dist(mids_1993, connected="weak") # Get weak components
cd
```

Component sizes

```
plot(1:length(cd$cdist),cd$cdist,xlab="Size",ylab="Frequency")
```

Who's in the largest component?

```
cl <- component.largest(mids_1993, connected="weak")
cl
```

Plot the largest weak component

```
gplot(mids_1993[cl,cl], boxed.lab=FALSE, label.cex=0.5,
    label.col=4,label=network.vertex.names(mids_1993)[cl])
```

Likewise, many routines exist for handling isolates

```
is.isolate(mids_1993, 3)     # Is the third vertex (BHM) an isolate?
is.isolate(mids_1993, "BHM")    # The peaceful islands?
is.isolate(mids_1993, "USA")    # Perhaps less so....
isol <- isolates(mids_1993)    # Get the entire list of isolates
isol
```

```
network.vertex.names(mids_1993)[isol]   # Which countries are isolates?
```

Another way to remove isolates from sociograms

```
gplot(mids_1993[-isol,-isol], label.cex=0.5,
    label.col=4,label=network.vertex.names(mids_1993)[-isol])
```

For more information...

```
?clique.census
?components
?component.dist
```

```
?dyad.census
?is.isolate
?isolates
```

```
?kcycle.census
?kpath.census
?triad.census
```

4.9 Elementary random graph generation

```
rgraph(10)       # A uniform random digraph of order 10
rgraph(10, tprob=3/9)    # Homogeneous Bernoulli w/mean degree 3
rgraph(10, tprob=3/9, mode="graph")  # As above, but undirected
rgnm(1, 10, 20)     # Uniform conditional on order, edges
rguman(1, 10, mut=0.5, asym=0.25, null=0.25) # Homogeneous multinomial on dyad census
rguman(1, 10, mut=0, asym=1, null=0)  # An extreme case: random tournament
gplot3d(rgws(1,50,1,2,0))    # A Watts-Strogatz process - baseline
gplot3d(rgws(1,50,1,2,0.05))   # ...with rewiring probability 0.05
gplot3d(rgws(1,50,1,2,0.2))   # ...with rewiring probability 0.2
```

For more information...

```
?rgbn
?rgmn
```

```
?rgraph
?rguman
```

```
?rgws
```

4.10 Basic connectivity/distance measurement, and cohesion

```
g <- rgraph(20, tprob=3/19)    # Start with a random digraph
g
is.connected(g)      # Is g strongly connected?
is.connected(g, connected="weak")   # How about weakly connected?
geodist(g)       # Get information on geodesics
reachability(g)      # Return the reachability matrix
symmetrize(g)       # Symmetrize g using the "or" rule
symmetrize(g, rule="strong")    # Symmetrize g using the "and" rule
```

Several ways to get relatively cohesive groups

```
clique.census(g)      # Maximal clique census
kcores(g)       # k-cores (by degree)
cutpoints(g)       # find articulation points
```

Showing cohesion information can aid visualization. Here, show critical positions

```
gplot(contig_1993,vertex.col=2+cutpoints(contig_1993,mode="graph",return.indicator=T))
```

Show the nesting of cores

```
kc<-kcores(contig_1993,cmode="indegree")
gplot(contig_1993,vertex.col=rainbow(max(kc)+1)[kc+1])
```

Showing members of the 5-core only

```
gplot(contig_1993[kc>4,kc>4],vertex.col=rainbow(max(kc)+1)[kc[kc>4]+1])
```

For more information...

```
?bicomponent.dist
?cutpoints
?geodist
```

```
?kcores
?is.connected
?reachability
```

```
?symmetrize
```

4.11 Positional analysis

Generate a structural equivalence clustering of the CoW alliance data

```
gplot(alliances_1993,gmode="graph",vertex.cex=0.5) #An initial look....
ec <- equiv.clust(alliances_1993, mode="graph",plabels=network.vertex.names(alliances_1993))
ec # The clustering
plot(ec) # Plot the dendrogram
rect.hclust(ec$cluster, h=20)
```

Use the clustering to form an structural equivalence (SE) blockmodel

```
bm <- blockmodel(alliances_1993, ec, h=20)
bm
```

We can display the blockmodel in several ways...

```
plot.sociomatrix(alliances_1993[bm$order.vector,bm$order.vector],drawlab=FALSE)
```

Extract the block image

```
bimage <- bm$block.model
bimage
bimage[is.nan(bimage)] <- 1
```

Visualizing the block image (with self-reflexive ties)

```
gplot(bimage, diag=TRUE, edge.lwd=bimage*5, vertex.cex=sqrt(table(bm$block.membership))/2,
   gmode="graph", vertex.sides=50, vertex.col=gray(1-diag(bimage)))
```

5 Credits

5.1 The statnet core development team

- Carter T. Butts (UC Irvine)
- Steven M. Goodreau (University of Washington)
- Mark S. Handcock (UCLA)
- David R. Hunter (Penn State University)
- Martina Morris (University of Washington)

5.2 Document authorship credits

- Section 1
  - Steven M. Goodreau
  - Ryan M. Acton
- Section 2
  - Steven M. Goodreau
  - Carter T. Butts
  - Ryan M. Acton
- Section 3
  - Carter T. Butts
  - Ryan M. Acton
- Section 4
  - Carter T. Butts
  - Ryan M. Acton

5.3 Document LaTeX source code

- Ryan M. Acton