Sampling and Network Inference

2011 Political Networks Conference
Ann Arbor, MI
May 2010

Presenters:
Carter T. Butts (University of California, Irvine)
Zack Almquist (University of California, Irvine)
Sean Fitzhugh (University of California, Irvine)
Nicole Pierski (University of California, Irvine)
Emma Spiro (University of California, Irvine)

statnet Core Development Team:
Carter T. Butts (University of California, Irvine)
Steven M. Goodreau (University of Washington)
Mark S. Handcock (University of Washington)
David R. Hunter (Penn State University)
Martina Morris (University of Washington)

Table of contents **Author**

Basic resources

R webpage: http://www.r-project.org
Helpful **R** tutorials: http://cran.r-project.org/other-docs.html
statnet webpage: http://statnet.org
statnet help: statnet_help@statnet.org
Workshop web site: http://polnet2011.ncasd.org/

Typographical conventions

Text in **Courier bold** represents code for you to type.

Text in Courier regular represents comments or **R** output.

All other text represents instructions and guidance.

SECTION 0. GETTING STARTED

**** The instructions in Section 0 match those on the workshop web page (http://polnet2011.ncasd.org/). If you have already installed the required software, there is no need to do so again. ****

0.1. Download and install the latest version of **R**.

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

0.2. Download and attaching **statnet** *and associated packages:*

a. Open **R**.
b. Install the **statnet** installer. At the **R** cursor >, type:

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

c. Now, and in the future, you can install/update **statnet** at any point using the installer that comes with **statnet**. Step (b) 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.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.

d. Attach **statnet** to your **R** session by typing:

```
library(statnet)
```

0.3. Download and install supplemental packages:

a. We recommend installing some additional, non-**statnet** package that are employed in selected exercises; you do not have to install these packages to use **statnet** in general, but some specific functions (or other analyses shown here) do expect them. To do so, at the **R** cursor type:

```
install.packages("coda")
```

0.4. Set a specific working directory for this tutorial if you wish.

a.If you are using Windows, go to File > Change Dir and choose the directory.
b.Otherwise, use the setwd directory command:

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

SECTION 1. SIMPLE DESIGN-BASED INFERENCE WITH RANDOM WALK SAMPLES

1.1 Load the workshop data file

```
load("polnet2011.Rdata")      # If you get an error, make sure it's in your working directory!

plot(ah50,vertex.col="grade") # Test network from public version of AddHealth (symmetrized)
ah50
```

1.2 Draw a random walk sample

```
args(netwalker)                          # The "netwalker" function lets us simulate RW sampling
set.seed(10)
samp100 <- netwalker(ah50,100,seed=1)        # Draw a small sample - n=100
samp1000 <- netwalker(ah50,1000,seed=1)      # Draw a larger sampler - n=1000
samp2000 <- netwalker(ah50,2000,seed=1)      # Draw an even larger sampler - n=2000

names(samp100)                           # Two components: the sampled nodes, and their degrees
samp100$sample                           # Examine the sample
samp100$degree                           # Examine the degrees of the sampled nodes

length(unique(samp100$sample))           # We are sampling with replacement - not every case is
length(unique(samp1000$sample))          # unique!
length(unique(samp2000$sample))
```

1.3 Convergence assessment

```
#Let's get some covariate information for the sampled nodes...
ssex100 <- (ah50 %v% "sex")[samp100$sample]
ssex1000 <- (ah50 %v% "sex")[samp1000$sample]
ssex2000 <- (ah50 %v% "sex")[samp2000$sample]

#Load coda, and convert our sample sequences to MCMC objects
library(coda)
draws100 <- as.mcmc(cbind(samp100$degree,ssex100))
draws1000 <- as.mcmc(cbind(samp1000$degree,ssex1000))
draws2000 <- as.mcmc(cbind(samp2000$degree,ssex2000))

#Trace and density plots....
plot(draws100)
plot(draws1000)
plot(draws2000)

#Geweke convergence diagnostics
geweke.diag(draws100)
geweke.diag(draws1000)
geweke.diag(draws2000)

#Cumulative quantile plot
cumuplot(draws100)
cumuplot(draws1000)
cumuplot(draws2000)
```

1.4 Obtaining sample weights

```
N <- network.size(ah50)                          # For convenience, store the population size
deg <- degree(ah50,gmode="graph")                # Get the actual degrees

#Estimate the degree sum (Salganik and Heckathorn method)
```

```
ds100 <- N*100/sum(1/samp100$degree)
ds1000 <- N*1000/sum(1/samp1000$degree)
ds2000 <- N*2000/sum(1/samp2000$degree)
(c(ds100,ds1000,ds2000)-sum(deg))/sum(deg)     # Absolute relative error

#Using the degree sum, compute sample weights
sp100 <- samp100$degree/ds100
sp1000 <- samp1000$degree/ds1000
sp2000 <- samp2000$degree/ds2000
```

1.5 Estimating graph properties using Hansen-Hurwitz

```
#Mean degree
1/(N*100)*sum(samp100$degree/sp100)
1/(N*1000)*sum(samp1000$degree/sp1000)
1/(N*2000)*sum(samp2000$degree/sp2000)
mean(deg)                                          # True value

#Average neighborhood gender diversity (Herfindahl Index)
neighdiv <- sna::gapply(ah50,1,ah50%v%"sex",
    function(z){if(length(z)>0){z<-table(z); sum((z/sum(z))^2)}else 0})
1/(N*100)*sum(neighdiv[samp100$sample]/sp100)
1/(N*1000)*sum(neighdiv[samp1000$sample]/sp1000)
1/(N*2000)*sum(neighdiv[samp2000$sample]/sp2000)
mean(neighdiv)                                     # True value

#Average ego net transitivity
entrans <- sapply(ego.extract(ah50),
    function(z){if(NROW(z)>2){gtrans(z[-1,-1])} else 1})
1/(N*100)*sum(entrans[samp100$sample]/sp100)
1/(N*1000)*sum(entrans[samp1000$sample]/sp1000)
1/(N*2000)*sum(entrans[samp2000$sample]/sp2000)
mean(entrans)                                      # True value
```

SECTION 2: NETWORK INFERENCE WITH `sna`

2.1 Exploratory visualization

```
gplot(kfr[1,,],displaylabels=T)          # 1st observer's POV - can change
gplot(kfr[2,,],displaylabels=T)          # 2nd observer's POV

# Can scale based on mis-matching rates using hdist with normalize=TRUE
kfr.dist<-hdist(kfr,normalize=TRUE)
plot(cmdscale(kfr.dist),type="n")        # Note the large cluster near 0,0
text(cmdscale(kfr.dist),label=1:21)
abline(h=0,v=0,lty=3)
```

#For more information....

```
?gplot
?hdist
?cmdscale
```

2.2 Network inference

```
# Let's start by defining some priors for the Bayesian network inference
# model.  We'll use an uninformative network prior, together with weakly
# informative (but diffuse and symmetric) priors on the error rates.  Read
# the man page ("?bbnam") to get more information about how the routine works.
np<-matrix(0.5,21,21)        # 21 x 21 matrix of Bernoulli parameters (since n=21)
emp<-sapply(c(3,11),rep,21)  # Beta(3,11) priors for false negatives
epp<-sapply(c(3,11),rep,21)  # Beta(3,11) priors for false positives
hist(rbeta(100000,3,11))     # This gives you a sense of what the priors look like!

# Now, let's take some posterior draws for the friendship network, using
# various models (warning: slow)
kfr.post.fixed<-bbnam.fixed(kfr,nprior=np,em=3/(3+11),ep=3/(3+11))
kfr.post.pooled<-bbnam.pooled(kfr,nprior=np,em=emp[1,],ep=epp[1,])
kfr.post.actor<-bbnam.actor(kfr,nprior=np,em=emp,ep=epp)

# Examine the results - note the difference that heterogeneity makes!
summary(kfr.post.fixed)
summary(kfr.post.pooled)
summary(kfr.post.actor)
plot(kfr.post.fixed)
plot(kfr.post.pooled)
plot(kfr.post.actor)

# Look at some of the error stats....
hist(as.vector(1-kfr.post.actor$em-kfr.post.actor$ep))      # Overall informativeness
mean(as.vector(1-kfr.post.actor$em-kfr.post.actor$ep)<0)    # Neg. inf. rate
mean(as.vector(kfr.post.actor$em-kfr.post.actor$ep)>0)      # Pr(em>ep)
plot(as.vector(kfr.post.actor$em),as.vector(kfr.post.actor$ep),xlim=c(0,1),
  ylim=c(0,1),cex=0.25)                                  # Plot em and em simultaneously
abline(1,-1,col=2)                                       # Color bounds on informative region
abline(h=0,v=0,col=2)
# With bias, most accurate persons not always in the center....
plot(cmdscale(kfr.dist),pch=19,col=rgb(1-apply(kfr.post.actor$em,2,mean),
  1-apply(kfr.post.actor$ep,2,mean),0),
  cex=3*(1-apply(kfr.post.actor$em+kfr.post.actor$ep,2,mean)))
cor((1-apply(kfr.post.actor$em+kfr.post.actor$ep,2,mean)),apply(kfr.dist,1,mean))

# Show some posterior predictive network properties
hist(gden(kfr.post.actor$net))                           #Density histogram
hist(grecip(kfr.post.actor$net,measure="edgewise.lrr"))  #Reciprocity
temp<-log(gtrans(kfr.post.actor$net)/gden(kfr.post.actor$net))
hist(temp)   # Log-odds multiplier measure of transitivity (>0 implies trans)

# Alternate approach: consensus method (MLE version)
kfr.conc<-consensus(kfr,method="romney.batchelder")
```

```
# Compare the results from the two estimation methods
hist(apply(kfr.post.actor$net,1,function(x){gcor(x,kfr.conc)})) #Posterior cor
gcor(apply(kfr.post.actor$net,c(2,3),median),kfr.conc)  #Point estimate
gplot(kfr.conc,displaylabels=T)
gplot(apply(kfr.post.actor$net,c(2,3),median),displaylabels=T)

# For more information....
?rbeta
?bbnam
?gden
?grecip
?gtrans
?apply
?gcor
?rgb
```