Exponential Random Graph Models for Social Networks

Political Networks Conference – Ann Arbor, MI – June 2011

Carter T. Butts (University of California, Irvine)
Zack W. Almquist (University of California, Irvine)
Sean Fitzhugh (University of California, Irvine)
Lorien Jasny (University of California, Irvine)
Nicole Pierski (University of California, Irvine)
Emma Spiro (University of California, Irvine)

Joint with the rest of the Statnet Development Team:
Steven M. Goodreau (University of Washington)
Mark S. Handcock (UCLA)
David R. Hunter (Penn State University)
Pavel N. Krivitsky (Carnegie Mellon University)
Martina Morris (University of Washington)

Basic resources

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

Typographical conventions

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

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

```
# Text after pound signs is a comment
```

All other text represents instructions and guidance.

SECTION 0. GETTING STARTED

Open an R session, and set your working directory to the location where you would like to save this work. You can do this with the pull-down menus (File>Change Dir) or with the command:

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

To install all of the packages in the statnet suite:

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

Or, to only install the specific statnet packages needed for this tutorial:

```
install.packages("network")
install.packages("ergm")
install.packages("sna")
library(network)
library(ergm)
library(sna)
```

After the first time, to update the packages one can either repeat the commands above, or use:

```
update.packages("name.of.package")
```

For this tutorial, we will need one additional package (**coda**), which is recommended (but not required) by **ergm**:

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

SECTION 1. WORLD'S SHORTEST R TUTORIAL

```
a <- 3                                  # assignment
a                                       # evaluation
[1] 3

sqrt(a)                                 # perform an operation
b <- sqrt(a)                            # perform operation and save
b

ls()                                    # list objects in global environment
help(sqrt)                              # help w/ functions
?sqrt                                   # same thing
help.search("square")                   # hip to the square (root)?
??square                                # same thing
help.start()                            # lots of help

rm(a)                                   # remove an object

# Use the File menu to save your current global environment, change working
# directory, exit, etc.
```

SECTION 2: A QUICK INTRODUCTION TO `network`: DATA IMPORT, EXPLORATION, MANIPULATION

2.1 Built-in Datasets

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

2.2 Creating network Objects

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

2.3 Description and Visualization

```
summary(nflo)                                   #Get an overall summary
network.size(nflo)                              #How large is the network?
network.edgecount(nflo)                         #How many edges are present?
plot(nflo,displaylabels=T,boxed.labels=F)       #Plot with names
```

2.4 Adding, deleting, testing edges

```
g <- network.initialize(5)                      #Create an empty graph
g[1,2] <- 1                                     #Add an edge from 1 to 2
g                                               #Examine the result
g[1,2] <- 0                                     #Remove the edge from 1 to 2
g                                               #It's gone!
```

2.5 Testing adjacency

```
nflo[9,3]                                       #Medici to Barbadori?
nflo[9,]                                        #Entire Medici row
nflo[1:4,5:8]                                   #Subsets are possible
nflo[-9,-9]                                     #Negative numbers _exclude_ nodes
```

2.6 Add vertex attributes

```
nflo
nflo %v% "woo" <- letters[1:16]                 # Same as set.vertex.attribute
nflo %v% "woo"                                  # Same as get.vertex.attribute
```

2.7 Sample **sna** *Routines*

```
library(sna)                                    #Load the sna library
betweenness(flo)
isolates(nflo)
kcycle.census(nflo,mode="graph",maxlen=5)
```

SECTION 3. FITTING A BASIC ERG MODEL; THE *ERGM* COMMAND AND *ERGM* OBJECT.

Make sure the `statnet` package is attached:

```
library(statnet)
```

or

```
library(ergm)
library(sna)
```

The ergm package contains several network data sets that you can use for practice examples.

```
data(package="ergm")                    # tells us the datasets in our packages
data(florentine)                        # loads flomarriage & flobusiness data
flomarriage                             # Let's look at the flomarriage data
plot(flomarriage)                       # Let's view the flomarriage network
```

Remember the general ergm representation of the probability of the observed network, and the conditional log-odds of a tie:

$P(Y=y) = \exp[\theta' g(y)] / k(\theta)$	# Y is a network, $g(y)$ is a vector of network stats # θ is the vector of coefficients, $k(\theta)$ is a normalizing constant
$\text{logit}(P(Y_{ij}=1)) = \theta' \Delta(g(y))_{ij}$	# Y_{ij} is an actor pair in Y, $\Delta(g(y))_{ij}$ is the # change in g(y) when the value of Y_{ij} is toggled on

We begin with the simplest possible model, the Bernoulli or Erdös-Rényi model, which contains only an edge term.

```
flomodel.01 <- ergm(flomarriage~edges)          # fit model
flomodel.01                                     # look at the model

Newton-Raphson iterations:  5

MLE Coefficients:
 edges
-1.609

summary(flomodel.01)                            # look in more depth

==========================
Summary of model fit
==========================

Formula:   flomarriage ~ edges

Newton-Raphson iterations:  5

Maximum Likelihood Results:
      Estimate Std. Error MCMC s.e. p-value
edges  -1.6094     0.2449        NA  <1e-04 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

For this model, the pseudolikelihood is the same as the likelihood.

    Null  Deviance: 166.355  on 120  degrees of freedom
 Residual Deviance: 108.135  on 119  degrees of freedom
          Deviance:  58.221  on   1  degrees of freedom

AIC: 110.13    BIC: 112.92
```

How to interpret this model? The log-odds of any tie occurring is:

= -1.609 * change in the number of ties

= -1.609 * 1 # for all ties, since the addition of any tie to the # network changes the number of ties by 1!

Corresponding prob. = exp(-1.609) / (1+ exp(-1.609))

= 0.1667 # what you would expect, since there are 20/120 ties

Let's add a term often thought to be a measure of "clustering" -- the number of completed triangles

```
set.seed(1664)          #Seeding the RNG for demo purposes; otherwise, your output will vary
flomodel.02 <- ergm(flomarriage~edges+triangle)

Iteration 1 of at most 3:
the log-likelihood improved by 0.001723

Iteration 2 of at most 3:
the log-likelihood improved by 0.01581

Iteration 3 of at most 3:
the log-likelihood improved by 0.1805

This model was fit using MCMC.  To examine model diagnostics and check for degeneracy, use the
mcmc.diagnostics() function.

summary(flomodel.02)


==========================
Summary of model fit
==========================

Formula:   flomarriage ~ edges + triangle

MCMC sample of size 10000

Monte Carlo MLE Results:
         Estimate Std. Error MCMC s.e. p-value
edges     -1.6913     0.3284     0.038  <1e-04 ***
triangle   0.4076     0.3080     0.012   0.188
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Log-likelihood was not estimated for this fit.

To get deviances, AIC, and/or BIC from fit `flomodel.02` run

  > flomodel.02<-logLik(flomodel.02, add=TRUE)

to add it to the object or rerun this function with eval.loglik=TRUE.
```

Again, how to interpret coefficients?

Conditional log-odds of two actors forming a tie is:

-1.691 * change in the number of ties + 0.408 * change in number of triangles

if the tie will not add any triangles to the network, its log-odds. is -1.691.
if it will add one triangle to the network, its log-odds is -1.691 + 0.408 = -1.283
if it will add two triangles to the network, its log-odds is: -1.691 + 0.408*2 = -0.875

the corresponding probabilities are 0.156, 0.217, and 0.294.

Let's take a closer look at the ergm object itself:

```
class(flomodel.02)                    # this has the class ergm
```

```
[1] "ergm"
```

```
names(flomodel.02)                        # let's look straight at the ERGM obj.
```

```
  [1] "coef"           "sample"         "sample.miss"    "iterations"
 [5] "MCMCtheta"      "loglikelihood"  "gradient"       "covar"
 [9] "samplesize"     "failure"        "mc.se"          "burnin"
[13] "interval"       "network"        "newnetwork"     "theta.original"
[17] "mplefit"        "parallel"       "null.deviance"  "etamap"
[21] "formula"        "constraints"    "prop.weights"   "offset"
[25] "drop"
```

By default, ergm does not estimate the log-likelihood for models fit using MCMC; that is because this calculation involves the use of additional simulation, and can be time consuming. In order to obtain the log-likelihood, we use the `logLik` function. (It only has to be done once per object - thereafter, the likelihood information is stored in the model object.)

```
flomodel.02 <- logLik(flomodel.02, add=TRUE)    # Compute the log-likelihood, and update
summary(flomodel.02)                            # Voila!  Fit information is now present!
```

Model elements can also be selected directly:

```
flomodel.02$coef                          # The $ allows you to pull an element out from
flomodel.02$mle.lik                       # a list
flomodel.02$formula
```

We've seen how to add a dependence term to a model. Now, let's consider covariate effects. In the case of the Florentine families, marriage ties were part of a larger struggle over power and wealth. Could family wealth be related to the marriage network?

```
wealth <- flomarriage %v% "wealth"      # the %v% extracts vertex attributes from a network
wealth
plot(flomarriage, vertex.cex=wealth/25) # network plot with vertex size proportional to wealth
```

We can test whether edge probabilities are a function of wealth:

```
flomodel.03 <- ergm(flomarriage~edges+nodecov("wealth"))
summary(flomodel.03)
```

```
==========================
Summary of model fit
==========================

Formula:   flomarriage ~ edges + nodecov("wealth")

Newton-Raphson iterations: 4

Maximum Likelihood Results:
               Estimate Std. Error MCMC s.e. p-value
edges          -2.594929   0.536056        NA  <1e-04 ***
nodecov.wealth  0.010546   0.004674        NA  0.0259 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

For this model, the pseudolikelihood is the same as the likelihood.

    Null  Deviance: 166.355  on 120  degrees of freedom
 Residual Deviance: 103.109  on 118  degrees of freedom
          Deviance:  63.247  on   2  degrees of freedom

AIC: 107.11    BIC: 112.68
```

Yes, there is a significant positive association between familial wealth and the probability of a tie. (Note that here the probability of a tie conditionally related to the total wealth of the families involved. You can also model the probability of a tie as a function of wealth *differences*, or more exotic things. See `?"ergm-terms"` for more information!)

****

Now, let's try a model or two on directed data. We begin with attributions of "liking" among Sampson's Monks:

```
data(samplk)                                    # Load the data set
ls()                                            # See the object
samplk3
plot(samplk3)
sampmodel.01 <- ergm(samplk3~edges+mutual)      # Is there a statistically significant tendency
summary(sampmodel.01)                           # for ties to be reciprocated ("mutuality")?
```

Although the cases we have seen so far are fairly small, we can work with larger cases. Here's another:

```
data(faux.mesa.high)                            # Not an actual school - but it looks like one!
mesa <- faux.mesa.high
plot(mesa)
mesa
plot(mesa, vertex.col="Grade")
legend("bottomleft",fill=7:12,legend=paste("Grade",7:12),cex=0.75)

fauxmodel.01 <- ergm(mesa ~edges + nodematch("Grade",diff=T) +
     nodematch("Race",diff=T))

summary(fauxmodel.01)
```

Note that two of the coefficients are estimated as –Inf (the nodematch coefficients for race Black and Other). Why is this?

```
table(mesa %v% "Race")                          # Frequencies of race
mixingmatrix(mesa, "Race")
```

So the problem is that there are very few students in the Black and Other race categories, and these students form no homophilous (within-group) ties. The empty cells are what produce the –Inf estimates. Technically, this means that the MLE here does not exist: ever more negative coefficients would give us ever higher likelihoods (albeit with diminishing marginal returns), and no finite value is a maximum. In such situations, `ergm` inserts the limit to which the optimization process approaches.

Time to consider some missing data:

```
missnet <- network.initialize(10,directed=F)
missnet[1,2] <- missnet[2,7] <- missnet[3,6] <- 1
missnet[4,6] <- missnet[4,9] <- NA
missnet
plot(missnet)
ergm(missnet~edges)
```

The coefficient equals -2.590. This is the log-odds of the probability .0698. Our network has 3 ties, out of the 43 nodal pairs (10 choose 2 minus 2) whose dyad status we have observed. 3/43 = 0.0698.

```
set.seed(1707)
ergm(missnet~edges+degree(2))

missnet[4,6] <- missnet[4,9] <- 0

set.seed(1707)
ergm(missnet~edges+degree(2))
```

The two estimates for the degree2 coefficient differ considerably. In the first case, there is one node we know for sure has degree 2, two that may or may not, and seven that we know for sure do not. In the latter, there is one node that has degree 2, and nine that do not.

SECTION 4. NETWORK TERMS AVAILABLE FOR *ERGM* MODELING and SIMULATION

Terms are the expressions (e.g. “triangle”) used on the right-hand size of :

calls to **ergm** (to fit an ergm model)
calls to **simulate** (to simulate graphs from an ergm model fit)
calls to **summary** (to obtain measurements of network statistics on a dataset)

4.1. Terms provided with statnet

For a list of available terms that can be used to specify an ERGM, see Appendix B, or type:

```
help("ergm-terms")
```

For a more complete discussion of these terms see the "Specifications" paper in *J Stat Software* v. 24. (link is available online at www.statnet.org)

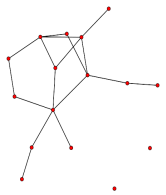
4.2. Coding new terms

We have recently released a new package (**ergm.userterms**) and tutorial aimed at making it much easier than before to write one"s own terms. The package is available on CRAN, and installing it will also download the tutorial (ergmuserterms.pdf). We are also hoping for the tutorial to appear soon in the *Journal of Statistical Software*. Note that writing up new **ergm** terms requires some knowledge of C and the ability to build R from source (although the latter is covered in the tutorial).

SECTION 5. NETWORK SIMULATION: THE *SIMULATE* COMMAND AND *NETWORK.SERIES* OBJECTS.

Once we have estimated the coefficients of an ERGM, the model is completely specified. It defines a probability distribution across all networks of this size. If the model is a good fit to the observed data, then networks drawn from this distribution will be more likely to "resemble" the observed data. To see examples of networks drawn from this distribution we use the **simulate** command:

```
set.seed(1707)
flomodel.03.sim <- simulate(flomodel.03,nsim=10) # simulate 10 networks from the model.

class(flomodel.03.sim)
[1] "network.series"

names(flomodel.03.sim)
[1] "formula"  "networks" "stats"    "coef"

length(flomodel.03.sim$networks)
[1] 10

flomodel.03.sim$networks[[1]]           # double brackets pulls an element
                                        # out of a list by position #
Network attributes:
   vertices = 16
   directed = FALSE
   hyper = FALSE
   loops = FALSE
   multiple = FALSE
   bipartite = FALSE
   total edges= 17
     missing edges= 0
     non-missing edges= 17

Vertex attribute names:
    priorates totalties vertex.names wealth

plot(flomodel.03.sim$networks[[1]])
```

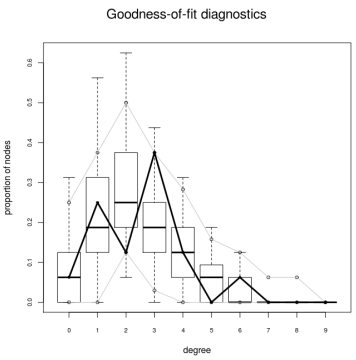


Voilà. (Of course, yours will look somewhat different.)

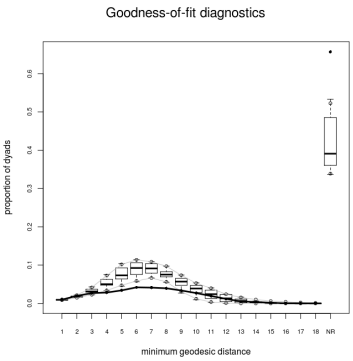
SECTION 6. EXAMINING THE QUALITY OF MODEL FIT – *GOF*.

ERGMs are generative models – that is, they represent the process that governs tie formation at a local level. These local processes in turn aggregate up to produce characteristic global network properties, even though these global properties are not explicit terms in the model. One test of whether a model "fits the data" is therefore how well it reproduces these global properties. We do this by choosing a network statistic that is not in the model, and comparing the value of this statistic observed in the original network to the distribution of values we get in simulated networks from our model.

```
set.seed(1707)
flomodel.03.gof <- gof(flomodel.03~degree)

flomodel.03.gof
plot(flomodel.03.gof)
```




```
mesamodel.02 <- ergm(mesa~edges)
set.seed(1707)
mesamodel.02.gof <- gof(mesamodel.02~distance,nsim=10)
plot(mesamodel.02.gof)
```



(for a good example of model exploration and fitting for the Add Health Friendship networks, see Goodreau, Kitts & Morris **Demography** 2009)

SECTION 7. DIAGNOSTICS: TROUBLESHOOTING AND CHECKING FOR MODEL DEGENERACY

The computational algorithms in **ergm** use MCMC to estimate the likelihood function. Part of this process involves simulating a set of networks to approximate unknown components of the likelihood.

When a model is not a good representation of the observed network the estimation process may be affected. In the worst case scenario, the simulated networks will be so different from the observed network that the algorithm fails altogether. This can occur for two general reasons. First, the simulation algorithm may fail to converge, and the sampled networks are thus not from the specified distribution. Second, the model parameters used to simulate the networks are too different from the MLE, so even though the simulation algorithm is producing a representative sample of networks, this is not the sample that would be produced under the MLE.

For more detailed discussions of model degeneracy in the ERGM context, see the papers in *J Stat Software* v. 24. (link is available online at www.statnet.org)

We can use diagnostics to see what is happening with the simulation algorithm, and these can lead us to ways to improve it. We will first consider a simulation where the algorithm works. To understand the algorithm, consider

```
fit <- ergm(flobusiness~edges+degree(1), interval=1, burnin=1000,seed=1)
```

This runs a version with every network returned. Let us look at the diagnostics produced:

```
mcmc.diagnostics(fit, center=F)
```

Let's look more carefully at a default model fit:

```
fit <- ergm(flobusiness~edges+degree(1))
```

To see the diagnostics use:

```
mcmc.diagnostics(fit, center=F)
```

Now let us look at a more interesting case, using a larger network:

```
data("faux.magnolia.high")
magnolia <- faux.magnolia.high
plot(magnolia, vertex.cex=.75)

fit <- ergm(magnolia~edges+triangle,seed=1)
mcmc.diagnostics(fit, center=F)
```

Very interesting. You could have gotten some more feedback about this during the fitting, by using:

```
fit <- ergm(magnolia~edges+triangle,seed=1,verbose=T)
```

You might try to increase the MCMC sample size:

```
fit <- ergm(magnolia~edges+triangle,seed=1,verbose=T,MCMCsamplesize=20000)
mcmc.diagnostics(fit, center=F)
```

Now, try it again with a sample size of 50,000.

How about trying the more robust version of modeling triangles -- GWESP? (For a technical introduction to GWESP see Hunter and Handcock; for a more intuitive description and empirical application see Goodreau Kitts and Morris 2009)

```
fit <- ergm(magnolia~edges+gwesp(0.5,fixed=T),seed=1)
mcmc.diagnostics(fit)
```

Still degenerate, but maybe getting closer?

```
fit <- ergm(magnolia~edges+gwesp(0.5,fixed=T)+nodematch("Grade")+nodematch("Race")+
     nodematch("Sex"),seed=1,verbose=T)

pdf("diagnostics.pdf")    #Use the recording function if possible, otherwise send to pdf
mcmc.diagnostics(fit)
dev.off()

fit <- ergm(magnolia~edges+gwesp(0.25,fixed=T)+nodematch("Grade")+nodematch("Race")+
     nodematch("Sex"),seed=1)

pdf("diagnostics.pdf")
mcmc.diagnostics(fit)
dev.off()

args(ergm)

fit <- ergm(magnolia~edges+gwesp(0.25,fixed=T)+nodematch("Grade")+nodematch("Race")+
     nodematch("Sex"),seed=1,MCMCsample=50000,interval=1000,verbose=T)
pdf("diagnostics.pdf")
mcmc.diagnostics(fit)
dev.off()
save.image()
```

Success! Of course, in real life one might have a lot more trial and error.

SECTION 8. ADDITIONAL FUNCTIONALITY

8.1. Additional functionality

The **statnet** suite of packages currently contains many additional features not covered in this tutorial:

- analysis of bipartite networks (**network** package)
- curved exponential family estimation and simulation (**ergm** base package)
- latent space and latent cluster analysis (**latentnet** package)
- network permutation models (**netperm** package)
- MLE estimation for degree distributions (negative binomial, Poisson, scale-free, etc.) (**degreenet** package)
- simulation of bipartite networks with given degree distributions (**networksis** package)
- tools for fitting relational event models (**relevent** package)
- hierarchical ERGMs (**hergm** package)

Any of these additional packages can be downloaded from CRAN. For more detailed information on these features, please visit the **statnet** webpage (http://statnet.org).

8.2. Additional functionality in development:

- dynamic network estimation and simulation (**ergm** package) – expected later 2011
- hierarchical versions of relational event models (**relevent** package) – expected later 2011
- ERGMs for valued ties (**ergm** package) – expected later 2011
- model-based simulation of complete networks from egocentric network data – expected early 2012

8.3. Statnet Commons: The development group

APPENDIX A: The many references of ergm and network

You will see the terms **ergm** and **network** used in multiple contexts throughout the documentation. This is common in R, but often confusing to newcomers. To clarify:

ergm

- **ERGM:** the acronym for an Exponential-family Random Graph Model; any statistical model for relational data that is written in exponential family form.
- **ergm package**: one of the constituent packages within the **`statnet`** suite
- **ergm function:** a function within the ergm package; fits an ERGM to a network object, creating an ergm object in the process.
- **ergm object:** a class of objects produced by a call to the ergm function, representing the results of an ERGM fit to a network.

network

- **network:** a set of actors and the relations among them. Used interchangeably with the term graph.
- **network package:** one of the constituent packages within the **`statnet`** suite; used to create, store, modify and plot the information found in network objects.
- **network object:** a class of object in **`R`** used to represent a network.

Appendix B: Table of existing statnet terms

Term	Undir?	Dir?	Bip?	Required Args	Optional Args
Basic terms					
edges	X	X	X		
density	X	X	X		
mutual		X			
asymmetric		X			
meandeg	X	X	X		
Nodal attribute terms					
nodecov (aka nodemain)	X	X	X	attrname	
nodefactor	X	X	X	attrname	
nodeifactor		X		attrname	
nodeofactor		X		attrname	
nodeicov		X		attrname	
nodeocov		X		attrname	
nodemix	X	X	X	attrname	
nodematch (aka match)	X	X	X	attrname	
absdiff	X	X	X	attrname	
absdiffcat	X	X	X	attrname	
b1factor			X	attrname	base
b2factor			X	attrname	base
smalldiff	X	X	X	attrname, cutoff	
Relational attribute terms					
edgecov	X	X	X	network or attrname	
dyadcov	X	X	X	network or attrname	
hamming	X	X	X	network or attrname	
hammingmix		X		network or attrname	base
Degree terms					
degree	X			vec. of degrees	by
idegree		X		vec. of degrees	by
odegree		X		vec. of degrees	by
b1degree			X	vec. of degrees	by
b2degree			X	vec. of degrees	by
gwdegree	X			decay	fixed
gwidegree		X		decay	fixed
gwodegree		X		decay	fixed
gwb1degree			X	decay	fixed
gwb2degree			X	decay	fixed
isolates	X	X	X		
concurrent	X				by
b1concurrent			X		by
b2concurrent			X		by
degcor	X				
degcrossprod	X				
adegcor	X				
rdegcor	X				
indegreepopularity		X			
outdegreepopularity		X			
Star terms					
kstar	X		X	vec. of star sizes	attrname
istar		X		vec. of star sizes	attrname

ostar		X		vec. of star sizes	attrname
b1star			X	vec. of star sizes	attrname
b2star			X	vec. of star sizes	attrname
b1twostar			X	b1attrname, b2attrname	base
b2twostar			X	b1attrname, b2attrname	base
b1starmix			X	k, attrname	base, diff
b2starmix			X	k, attrname	base, diff
m2star		X			
altkstar	X	X	X	lambda	fixed
Cycle and triangle terms					
triangle (aka triangles)	X	X			attrname
ctriple (aka ctriad)		X			attrname
ttriple (aka ttriad)		X			attrname
tripercent	X	X			attrname
cycle	X	X		vec. of cycle sizes	
localtriangle	X	X		network or attrname	
balance	X	X			
triadcensus	X	X		triad types to include	
intransitive		X			
nearsimmelian		X			
simmelian		X			
simmelianties		X			
transitive		X			
transitiveties		X			attrname
Actor-specific effects					
receiver		X			base
sender		X			base
sociality	X				attrname, base
Shared partner terms					
esp (edgewise shared ptnrs)	X	X		vec. of partner #s	
dsp (dyadwise shared ptnrs)	X	X	X	vec. of partner #s	
nsp (nonedge shared ptnrs)	X	X		vec. of partner #s	
gwesp	X	X		alpha	fixed
gwdsp	X	X	X	alpha	fixed
gwnsp	X	X		alpha	fixed
Paths					
towpath	X	X	X		
threepath	X	X	X		